



DISSERTATION / DOCTORAL THESIS

Titel der Dissertation / Title of the Doctoral Thesis

Application Integration Patterns and their Compositions: Foundations, Formalizations, Solutions

verfasst von / submitted by
Daniel Ritter

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of
Doktor der Technischen Wissenschaften (Dr. techn.)

Wien, 2019 / Vienna, 2019

Studienkennzahl laut Studienblatt /
degree programme code as it appears on the student
record sheet:

A 786 880

Dissertationsgebiet laut Studienblatt /
field of study as it appears on the student record sheet:

Informatik

Betreut von / Supervisor:

Univ.-Prof. Dipl.-Math. Dr. Stefanie Rinderle-Ma

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Vienna, _____
Date

Signature

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen / Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Wien, am _____
Datum

Unterschrift

Abstract

Recent trends in technology, business and society like cloud and mobile computing lead to increasingly distributed applications and processes requiring integration. Enterprise Application Integration (EAI) offers integration capabilities using message passing in the form of integration scenarios and constitute a centerpiece of current IT architectures. Due to the complex nature of EAI, the integration scenario logic builds on abstractions derived by best practices, collected as Enterprise Integration Patterns (EIPs), denoting the current foundation of developing and modeling EAI logic and solutions.

However, due to increasing distribution, not only the number of communication partners, but also the data volume and velocity requirements grow beyond processing capabilities of current systems. Further, personal digitalization trends like social media computing introduce increasing amounts of non-textual data beyond current variety capabilities and add trust requirements into the functional correctness and reliability of technology that is now used by non-technical users. These problems cannot be solved with current informally described integration pattern foundations or implementations.

This thesis defines formal foundations for EAI that enable the responsible development of integration solutions, help businesses to trust their correctness, and show how to build effective and efficient integration solutions.

First, EAI foundations in the form of EIPs are revisited regarding their actuality and comprehensiveness in the context of emerging trends, and captured as an extended pattern catalog. Patterns serve as a basis for the formalization of integration logic and their compositions as integration scenarios. Then the latter are studied for correctness-preserving optimization strategies, ensuring the efficiency of integration logic on the composition level. A prototype realization allows for the formal analysis of integration scenarios, thus addressing the trust challenge. To address the volume, velocity and variety challenges more efficient pattern solutions are conceptually developed using new technology trends and evaluated using a newly elaborated EIP benchmark.

In summary, this thesis builds a formal foundation for EAI and shows how this can be used to meet the trust, volume, velocity and variety requirements of current integration systems.

Kurzfassung

Technologische Fortschritte in den Bereichen Cloud und Mobile Computing haben nicht nur einen großen Einfluss auf unser tägliches Leben, sondern auf Unternehmensanwendungen und Geschäftsprozesse, die durch die fortschreitende Modularisierung und Verteilung immer mehr Daten austauschen. Die Anwendungs- und Prozessintegration, engl. Enterprise Application Integration (EAI), ermöglicht die Integration durch Nachrichtenaustausch und hat sich dadurch zu einem zentralen IT Thema in Unternehmen entwickelt. Um die Komplexität der Entwicklung von Integrationsszenarien handhabbar zu machen, wurden bewährte Vorgehensweisen als Integrationsmuster gesammelt und stellen die Grundlage für die Entwicklung und Modellierung von EAI Lösungen dar.

Durch die wachsende Verteilung hat sich jedoch nicht nur die Anzahl der Kommunikationspartner erhöht. Die aktuellen EAI Systeme kommen durch das gestiegene Datenvolumen und steigenden Leistungsanforderungen an ihre Grenzen. Hinzu kommt die Digitalisierung des täglichen Lebens vieler technisch unerfahrener NutzerInnen zum Beispiel durch Soziale Medien. Dadurch gewinnen nicht nur heterogenere, Multimediadatenformate an Bedeutung, sondern das Vertrauen in die Korrektheit und Verlässlichkeit der Technologie rückt in den Vordergrund. Diese Probleme können weder durch die aktuell informell beschriebenen Integrationsmuster noch deren in die Jahre gekommenen Implementierungen gelöst werden.

Im Rahmen dieser Arbeit werden die formalen Grundlagen der Anwendungsintegration definiert, wodurch eine verantwortungsbewusste Entwicklung von Integrationslösungen möglich wird. Das schafft nicht nur Vertrauen in deren Funktionsfähigkeit, sondern unterstützt bei der Entwicklung effizienterer Lösungen.

Dafür werden die EAI Grundlagen in Form der Integrationsmuster bezüglich ihrer Aktualität und Abdeckung im Rahmen der besprochenen Fortschritte untersucht und der aktuelle Musterkatalog erweitert. Dieser bildet die Grundlage für die formale Definition von Integrationsmustern und deren Komposition zu Integrationsszenarien. Durch die Formalisierung der Integrationsszenarien können Optimierungsstrategien entwickelt werden, welche deren Korrektheit nach der Verbesserung gewährleisten. Die prototypisch untersuchten Möglichkeiten der formalen Analyse von Integrationsszenarien stellt eine Lösung für das Vertrauensproblem dar. Für die besprochenen Datenverarbeitungsprobleme und die steigende Heterogenität der Datenformate werden effizientere Integrationslösungen basierend auf den Mustern konzipiert und mit einem eigens dafür entwickelten Benchmark analysiert.

Diese Arbeit entwickelt formale Grundlagen der Anwendungsintegration und zeigt wie dadurch das Vertrauen in aktuelle Integrationssysteme gestärkt und deren Effizienz bezüglich des wachsenden Datenvolumens und Leistungsanforderungen sowie heterogener Datenformate gesteigert werden kann.

Preface

This thesis originated during my work as a member of SAP Cloud Platform Integration (SAP CPI) and the research group Workflow Systems and Technology (WST) at the University of Vienna. The motivation for the work on “Application Integration Patterns and their Compositions” stems from practical experiences at SAP CPI and the growing need for solid theoretical foundations towards a trustworthy Enterprise Application Integration and more efficient integration solutions.

I would like to acknowledge a number of people who helped me considerably in writing this dissertation. First of all, I am indebted to my supervisor Prof. Dr. Stefanie Rinderle-Ma for her courage accepting the proposed topic and her encouraging me in this research on formal application integration foundations and efficient solutions. There are many things to be thankful for, however, I would like to especially mention her thoughtful guidance and constant support, our fruitful discussions and her vast knowledge in research methodologies to make this thesis a success. Her friendliness and the opportunity to contribute through teaching made me feel welcome in the WST research group. Finally, her flexibility and patience allowed for a stable and successful research with excellent results in this complex, industry-academic research project.

Moreover, I thank Prof. Dr. Avigdor Gal, a long-term, honourous member of the distributed and event-based systems community, in which I had the pleasure to grow into in the past years, who accepted to act as a reviewer.

Special thanks go to Dr. Norman May, my colleague and mentor at SAP, for proof-reading this thesis, co-authoring many of the contributions, his constant support, encouragement and guidance through all aspects of this work.

A thesis is even more rewarding with a friendly and collaborative atmosphere, and thus I thank my academic collaboration partners, especially Dr. Fredrik Nordvall Forsberg, Dr. Andrey Rivkin and Prof. Dr. Marco Montali for the joint work, many fruitful discussions, proof-reading this thesis, encouragement, and friendship. It was a real pleasure and honor working with them.

Furthermore, I thank Gunther Rothermel and Dr. Achim Kraiß from SAP for the funding and encouragements, as well as Christian Stenzel, Thomas Willhalm, Roman Dementiev, and especially the late Detlef Poth (†2018) from Intel for providing the latest FPGA hardware, used for one of our pattern solutions. Finally, I thank my co-authors and former colleagues at SAP Jan Sosulski, Jonas Dann, Dr. Kai Sachs, and especially my former teammate Manuel Holzleitner for innumerable coffee breaks and fruitful discussions.

I also thank the whole WST team for feedback, support and making my visits and teaching a memorable experience (especially, but not limited to Monika Hofer-Mozelt, Karolin Winter, Dr. Georg Kaes, Dr. Kristof Böhmer, Florian Stertz, Conrad Indiono, Dr. Walid Fdhila, Manuel Gall, and Dr. Jürgen Mangler).

My best thanks go to my wife Rahel. Rahel’s love, patience and support have made this work possible. She encouraged me through the bad times, and shared the good times with me. The birth of our baby daughter helped me to come to an end. This work is dedicated to Rahel.

Publications

Core:

1. Daniel Ritter and Jan Sosulski. Modeling exception flows in integration systems. In Proceedings of the 18th IEEE International Enterprise Distributed Object Computing Conference (EDOC), pages 12–21. IEEE, 2014.
2. Daniel Ritter and Manuel Holzleitner. Integration adapter modeling. In Proceedings of the 27th International Conference on Advanced Information Systems Engineering (CAiSE), pages 468–482. Springer, 2015.
3. Daniel Ritter and Stefanie Rinderle-Ma. Toward a collection of cloud integration patterns. Technical Report, CoRR, abs/1511.09250, 2015.
4. Daniel Ritter. Towards more data-aware application integration. In Proceedings of the 30th British International Conference on Databases (BICOD), pages 16–28. Springer, 2015.
5. Daniel Ritter, Norman May, Kai Sachs, and Stefanie Rinderle-Ma. Benchmarking integration pattern implementations. In Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems (DEBS), pages 125–136. ACM, 2016.
6. Daniel Ritter and Jan Sosulski. Exception handling in message-based integration systems and modeling using BPMN. International Journal of Cooperative Information Systems, 25(02):1650004, 2016.
7. Daniel Ritter, Norman May, and Stefanie Rinderle-Ma. Patterns for emerging application integration scenarios: A survey. Information Systems, 67:36–57, 2017.
8. Daniel Ritter. Database processes for application integration. In Proceedings of the 31st British International Conference on Databases (BICOD), pages 49–61. Springer, 2017.
9. Daniel Ritter, Jonas Dann, Norman May, and Stefanie Rinderle-Ma. Hardware accelerated application integration processing: Industry paper. In Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems (DEBS), pages 215–226. ACM, 2017.
10. Daniel Ritter and Stefanie Rinderle-Ma. Toward application integration with multimedia data. In Proceedings of the 21st IEEE International Enterprise Distributed Object Computing Conference (EDOC), pages 103–112. IEEE, 2017.
11. Daniel Ritter. Hardware accelerated application integration: challenges and opportunities. In Proceedings of the Second International Workshop on Active Middleware on Modern Hardware, ACTIVE@Middleware, page 15, 2017.

12. Daniel Ritter, Norman May, Fredrik Nordvall Forsberg, and Stefanie Rinderle-Ma. Optimization strategies for integration pattern compositions. In *Proceedings of the 12th ACM International Conference on Distributed and Event-based Systems (DEBS)*, pages 88–99. ACM, 2018.
13. Daniel Ritter and Stefanie Rinderle-Ma and Marco Montali and Andrey Rivkin and Aman Sinha. Formalizing application integration patterns. In *Proceedings of the 22nd IEEE International Enterprise Distributed Object Computing Conference (EDOC)*, pages 11–20. IEEE, 2018.
14. Daniel Ritter, Stefanie Rinderle-Ma, Marco Montali, Andrey Rivkin, and Aman Sinha. Catalog of formalized application integration patterns. Technical Report, CoRR, abs/1807.03197, 2018.
15. Daniel Ritter, Fredrik Nordvall Forsberg, Stefanie Rinderle-Ma, and Norman May. Catalog of optimization strategies and realizations for composed integration patterns. Technical Report, CoRR, abs/1901.01005, 2019.

Complementary:

1. Daniel Ritter. Experiences with business process model and notation for modeling integration patterns. In *Proceedings of the 10th European Conference on Modelling Foundations and Applications (ECMFA)*, pages 254–266. Springer, 2014.
2. Daniel Ritter. Using the business process model and notation for modeling enterprise integration patterns. Technical Report, CoRR, abs/1403.4053, 2014.
3. Daniel Ritter. What about database-centric enterprise application integration? In *Proceedings of the 6th Central-European Workshop on Services and their Composition (ZEUS)*, pages 73–76, 2014.
4. Daniel Ritter and Jan Bross. Datalogblocks: relational logic integration patterns. In *Proceedings of the 25th International Conference on Database and Expert Systems Applications (DEXA)*, pages 318–325, 2014.
5. Daniel Ritter. Compilation of BPMN-based integration flows. In *Proceedings of the 7th Central-European Workshop on Services and their Composition (ZEUS)*, pages 1–9, 2015.
6. Daniel Ritter and Manuel Holzleitner. Toward resilient mobile integration processes. In *Proceedings of the 21st International Conference on Business Information Systems (BIS)*, pages 278–291. Springer, 2018.

Contents

1	Introduction	1
1.1	Enterprise Application Integration	2
1.2	Challenges and Research Gap	4
1.2.1	Emerging Application Integration	4
1.2.2	New Challenges for Enterprise Application Integration	6
1.2.3	Research Gap	8
1.3	Research Questions and Methodology	10
1.4	Contributions	12
1.5	Outline of this Thesis	15
I	Foundation	17
2	Application Integration Foundations	19
2.1	Integration Patterns, Compositions and Mining	21
2.1.1	Integration Patterns	21
2.1.2	Integration Pattern Composition	23
2.1.3	Building Blocks of Integration System Architectures	25
2.1.4	The Pattern Engineering Process	26
2.2	Pattern Identification: From Best Practices to Patterns	28
2.2.1	Pattern Identification Methodology	28
2.2.2	Deductive Analysis: Literature Review	29
2.2.3	Inductive Analysis: System Review	38
2.3	Pattern Authoring and a Pattern Catalog	45
2.3.1	Design of a Pattern Catalog	45
2.3.2	Example Integration Pattern Authoring	50
2.4	Quantitative Analysis	54
2.4.1	Integration Scenarios	54
2.4.2	Scenario Analysis	56
2.5	Conclusions	58
3	On Formalizing Integration Patterns and their Compositions	61
3.1	Formal Pattern Semantics	62
3.1.1	Formalization Requirements Analysis and Design Choices	65
3.1.2	Petri Nets with Time and Transactional Data	68
3.1.3	Evaluation: Comprehensiveness, Correctness, and Case Studies	89
3.1.4	Conclusions	100
3.2	Composing Patterns	101

3.2.1	Structural Pattern Analysis	104
3.2.2	Graph-based Pattern Compositions	106
3.2.3	Timed Db-Nets with Boundaries and Synchronization	109
3.2.4	Translating Contract Graphs to Timed DB-nets with Boundaries	114
3.2.5	Evaluation: Case Studies	122
3.2.6	Conclusions	127
3.3	Related Work	129
3.3.1	Pattern Formalization	130
3.3.2	Pattern Composition Formalization	131
3.4	Discussion	132
4	Optimization Strategies for Pattern Compositions	135
4.1	Optimization Strategies and Design Choices	139
4.1.1	Identifying Optimization Strategies	139
4.1.2	Process Simplification	140
4.1.3	Data Reduction	140
4.1.4	Parallelization	141
4.1.5	Discussion and Design Choices	141
4.2	Optimization Strategy Formalization	142
4.2.1	OS-1: Process Simplification	144
4.2.2	OS-2: Data Reduction	145
4.2.3	OS-3: Parallelization	148
4.2.4	Optimization Correctness	150
4.3	Evaluation	153
4.3.1	Abstract Cost Model	153
4.3.2	Quantitative Analysis	154
4.3.3	Case Studies	158
4.4	Related Work	160
4.5	Conclusions	162
II	Realization	165
5	Benchmarking Patterns	167
5.1	Integration Pattern System Implementations	170
5.1.1	Apache Camel	170
5.1.2	Vectorization: Micro-batching Integration Processing	172
5.2	Integration Scenario Analysis	178
5.2.1	Integration Scenario Styles	178
5.2.2	Analysis of Real-World Applications	179
5.2.3	Summary	182
5.3	Integration Pattern Benchmark	182
5.3.1	Benchmark Design	182
5.3.2	Pattern Design Choices	185
5.3.3	Message Transformation Patterns	187
5.3.4	Message Delivery Semantics	188
5.3.5	Benchmark Implementation	189
5.4	Evaluation	191
5.4.1	Benchmark Setup	191

5.4.2	Benchmark Results	191
5.5	Related Work	195
5.6	Conclusions	198
6	Pattern Solutions	201
6.1	Vectorization: Database-centric Pattern Solutions	205
6.1.1	Transaction Processing and Big Data Management Systems	206
6.1.2	Database Transition System	210
6.1.3	Evaluation	218
6.1.4	Conclusions	221
6.2	Specialization: Hardware-Accelerated Pattern Solutions	221
6.2.1	Field-Programmable Gate Arrays	224
6.2.2	Dataflow Integration System on FPGAs: Patterns to Circuits	227
6.2.3	Message Processing	234
6.2.4	Evaluation	236
6.2.5	Conclusions	245
6.3	Evolution: Multimedia Pattern Solutions	246
6.3.1	Image Processing and Semantic Knowledge Representation	249
6.3.2	Literature and Application Analysis	253
6.3.3	Multimedia EAI Concepts	257
6.3.4	Evaluation	265
6.3.5	Conclusions	268
6.4	Related Work	269
6.4.1	Data-centric Pattern Solutions	269
6.4.2	Hardware Pattern Solutions	270
6.4.3	Multimedia Pattern Solutions	272
6.5	Discussion	274
7	Conclusions	281
7.1	Summary	281
7.2	Outlook	285
	Bibliography	326
A	Correctness-Preserving Reduce Interaction Optimizations	327
A.1	Ignore Failing Endpoints	327
A.2	Reduce Requests	328
A.3	Correctness Considerations	328

List of Figures

1.1	Different perspectives on <i>integration</i>	3
1.2	IT trends after 2004 in the context of application integration	5
1.3	IT trends after 2004 an their relationship to application integration	6
1.4	Conceptual EAI system overview with challenges	8
1.5	Overview of contributions and outline	15
2.1	The enterprise application integration patterns from 2004	22
2.2	Pattern composition in EIP icon notation	24
2.3	Pattern composition in BPMN	25
2.4	Integration system architecture with corresponding pattern categories	26
2.5	Pattern engineering process	27
2.6	Design science methodology used for systematic pattern identification	28
2.7	Distribution of topics mentioned in literature over time	30
2.8	Solutions for NFAs not covered by the EIPs by system vendor.	42
2.9	Conversational approach for exactly-once-in-order messaging	51
2.10	Scenarios using original EIPs	55
2.11	New capability categorization	56
2.12	Integration system architecture with unaltered, extended and novel pattern categories	58
3.1	<i>Responsible</i> pattern formalization process	64
3.2	EIP requirement categories	66
3.3	Aggregator pattern variant as a timed db-net	68
3.4	Db-net layers	69
3.5	Db-net taxi booking process example	72
3.6	Timed-arc Petri net example	75
3.7	Load balancer realization as a timed db-net	79
3.8	Message translator realization as a timed db-net	80
3.9	Splitter realization and sample <i>split</i> subnet realization as a timed db-net	80
3.10	Content-based router and content enricher realization as a timed db-net	81
3.11	Resquencer realization as a timed db-net	82
3.12	Circuit breaker realization as a timed db-net	83
3.13	Throttler and delayer realizations as a timed db-net	84
3.14	Finite db-net execution traces of a content-based router and a load balancer in timed db-net	86
3.15	A partial execution of a content enricher timed db-net	87
3.16	Sample <i>translation</i> subnet realization	89
3.17	Timed db-net comprehensiveness	90

3.18	Message translator as a timed db-net (in CPN Tools)	91
3.19	Splitter as a timed db-net (CPN Tools)	91
3.20	Content enricher as a timed db-net (in CPN Tools)	92
3.21	Aggregator as a timed db-net (in CPN Tools)	93
3.22	Flawed content-based router (in CPN Tools)	94
3.23	Replicate material from SAP Business Suite (in BPMN)	95
3.24	Replicate material scenario translated into its timed db-net representation (schematic)	96
3.25	Pattern composition as hierarchical timed db-net (in CPN Tools)	96
3.26	Predictive maintenance — create notification scenario as modeled by a user (in BPMN)	97
3.27	Create notification scenario translated into timed db-nets (schematic)	98
3.28	Create notification pattern composition as hierarchical timed db-net before simulation (in CPN Tools)	99
3.29	Create notification pattern composition as hierarchical timed db-net after simulation (in CPN Tools)	100
3.30	Responsible pattern composition process (process-perspective)	102
3.31	Condensation based on pattern boundaries	103
3.32	From pattern contract graphs to timed db-nets with boundaries (methodological)	104
3.33	Structural integration pattern categories: Start, end, message processor, fork, join	105
3.34	IPCG from the material replication scenario	109
3.35	Example net with boundaries	111
3.36	Translation templates for the start and end pattern categories	115
3.37	Translation templates of unconditional for and join pattern categories	116
3.38	Translation of an conditional fork pattern category	117
3.39	Translation of a message processor pattern category	117
3.40	Translation of a merge pattern category	118
3.41	Translation of an external call pattern category	118
3.42	1:1 pattern contract construction	120
3.43	1:o pattern contract construction	121
3.44	Example of a message translator construction	122
3.45	Join router construction	123
3.46	Complete integration pattern contract graph of the replicate material scenario	124
3.47	Material replicate scenario as a timed db-nets with boundaries	125
3.48	Material replicate scenario simulation	126
3.49	Integration pattern contract graph of the predictive maintenance scenario	127
3.50	Predictive maintenance scenario as a timed db-nets with boundaries	128
3.51	Predictive maintenance scenario simulation	129
4.1	IPCG from the material replicate scenario after a “sequence to parallel” optimization	137
4.2	Responsible pattern composition optimization process (process-perspective)	138
4.3	Redundant sub-process rule	144
4.4	Combine sibling patterns rule	145
4.5	Rules for early-filter and early-mapping	146
4.6	Rules for early-aggregation and early-claim check	147
4.7	Rules for early-split	148
4.8	Rules for sequence to parallel variants	149
4.9	Heterogeneous sequence to parallel	150
4.10	Timed db-net translation of IPCGs before and after applying the “combine sibling patterns” rewrite rule	151

4.11	Data inputs, outputs and external resources of a pattern	153
4.12	Pattern composition evaluation pipeline	156
4.13	Pattern reduction per scenario	156
4.14	Unused elements in integration scenarios	157
4.15	Parallelization scenario candidates	158
4.16	“Sequence to parallel” optimization candidates	158
4.17	Country-specific invoicing	160
4.18	Invoice processing after application of strategies OS-1–3	161
5.1	Apache Camel system architecture	171
5.2	Apache Camel exchange	172
5.3	Message processing in Apache Camel by abstract example	173
5.4	Vectorized processing with Apache Camel	174
5.5	Message routing and transformation patterns mapped to Datalog	174
5.6	Overview of integration scenario types <i>ST1–ST6</i>	179
5.7	Extended PDGF-based message creation	183
5.8	EIPBench execution phases	190
5.9	Content-based router (size scaling)	194
5.10	Message throughput of the content-based router (concurrent users as threads) . .	195
5.11	Message throughput of the content-based router (batch scaling)	196
6.1	CCT scenario with focus on the integration processing	202
6.2	Solution proposals for the three Vs in related domains	203
6.3	Timed db-net realizations	205
6.4	Database transaction states	207
6.5	Transaction processing example	208
6.6	Big data management system architecture sketch	209
6.7	Multi-relational message collection	210
6.8	Timed db-net realization of a vectorizing content-based router	213
6.9	Transactional process model	213
6.10	Structural pattern category: message processor	214
6.11	Structural pattern categories: fork and join	215
6.12	Message processing model	216
6.13	Automatic detection of transactions for more complex integration scenarios . . .	218
6.14	Database transition system design flow	218
6.15	Pattern throughput benchmark and process latencies for CCT scenario	220
6.16	FPGA hardware for EAI processing	223
6.17	CCT scenario with focus on the integration processing	224
6.18	Combinational and asynchronous sequential logic	225
6.19	Logic islands and with I/O blocks	227
6.20	FPGA layout with interspersed BRAM blocks and DSP units	228
6.21	Basic integration aspects on hardware	229
6.22	Router and load balancer patterns	230
6.23	Splitter and aggregator patterns	231
6.24	Translator and content enricher patterns	233
6.25	Conceptual view on pattern templates and hierarchical format processing	234
6.26	Template throughput for predicate and expression template patterns	238
6.27	Template throughput for composed patterns	238

6.28	Resource usage on the FPGA chip (floorplan) for the predicate template and the remaining system components	241
6.29	Concurrent user measurements	242
6.30	Message size baseline measurements	243
6.31	Connected cars scenario performance	244
6.32	Multimedia sub-process for social media emotion harvesting (excerpt)	247
6.33	Challenges of user-centric interaction on semantic message contents	248
6.34	Digitalization of an analog example image	250
6.35	Example RDF graph	251
6.36	Excerpt of archetypal and intermediate face expression profiles and emotion representation of a <i>VirtualHuman</i>	252
6.37	Literature analysis: image processing in industries over time	254
6.38	Multimodal operation classification	257
6.39	Conceptual object model	259
6.40	Evolution of the logical representation of a splitter	260
6.41	Evolution of the logical representation of an aggregator	261
6.42	Message translator and content filter: re-coloring	261
6.43	Content enricher: add shape	262
6.44	Feature selector (interaction)	263
6.45	Detector region (improvement)	264
6.46	Conceptual EAI system architecture with multimedia extensions	265
6.47	Vendor libraries as part of an integration system	266
6.48	Multimedia message throughput of the social marketing scenario	268
6.49	Message throughput of the social marketing scenario	269
6.50	Abstract pattern building block	275
6.51	Timed db-net with ontology-based data access	277
7.1	Conceptual EAI system overview — the big picture	286
A.1	Rules for ignore failing endpoints	327
A.2	Rules for reduce requests	328

List of Tables

2.1	Solutions for trends and non-functional aspects	31
2.2	System review — horizontal search	39
2.3	Original EIPs used in systems (Boomi to Apache Flume)	40
2.4	Original EIPs used in systems (Apache Nifi to Webmethods)	41
2.5	New integration patterns for NFAs in the context of system implementations from <i>security</i> to <i>processing</i>	47
2.6	New integration patterns for NFAs in the context of system implementations from <i>conversations</i> to <i>composition</i>	48
2.7	Common pattern formats: enterprise integration patterns, cloud computing pat- terns, design patterns	51
3.1	Formalization requirements	67
3.2	Formalization requirements (summary)	101
4.1	Optimizations in related domains — horizontal search	139
4.2	Optimization strategies in context of the objectives	140
4.3	Abstract costs of relevant patterns	155
5.1	Integration scenarios grouped by their integration styles and examples: from CBR to AGG	180
5.2	Integration scenarios grouped by their integration styles and examples: from MT to UDF	181
5.3	Message routing patterns with microscale factors	186
5.4	Message transformation patterns with microscale factors	187
5.5	Message delivery semantics with microscale factors	189
5.6	Message throughput of content-based routing and message transformation pattern benchmarks compared to the baseline	192
5.7	Throughput benchmarks for message delivery semantics	193
5.8	EIPBench in the context of related work	197
6.1	EIPBench pattern benchmarks	219
6.2	Test hardware comparison	236
6.3	FPGA pattern benchmarks	237
6.4	Resource occupation	240
6.5	Resource occupation CBR-A, SP-B	240
6.6	Multimedia data induced shift of core EAI characteristics	246
6.7	Industry and application analysis results	255
6.8	Integration pattern multimedia aspects (re-calculated as <i>recal</i>)	258

6.9	Model coverage compared to vision API definitions	267
6.10	Lessons learned from related work	271

Chapter 1

Introduction

Contents

1.1	Enterprise Application Integration	2
1.2	Challenges and Research Gap	4
1.3	Research Questions and Methodology	10
1.4	Contributions	12
1.5	Outline of this Thesis	15

In an increasingly digitalized and modularized world, the integration of applications, processes and devices becomes more relevant for companies [GKRH13] as well as for non-technical users, for whom digital innovations have become companions in their daily life [HPG⁺12]. Examples of such integration scenarios include the flexible extension of existing applications by cloud offerings through integration of *Hybrid Applications* (i.e., combined cloud and on-premise or company internal application) [VT17, For16a], the incorporation of context-aware device data into cloud applications and business networks [RH18], and the integration with social and multimedia channels [AAB⁺17].

For traditional application systems and packaged applications (e.g., *Enterprise Resource Planning* (ERP), *Customer Relationship Management* systems (CRM)) integration shall allow for cases like secure and flexible extension and adaptation to changing requirements such as legal regulations (e.g., for business trends like *postmodern ERP* [GKRH13]). In addition, the growing number of communication partners and heterogeneity (e.g., of message formats) becomes critical for integration through the advent of cloud applications [BBG11, FLR⁺14], companies collaborating via business networks [FGH⁺98], mobile applications and computing (e.g., [Rot05]), and the integration of devices or the *Internet of Things* (IoT) [AIM10] (e.g., vehicle logistics [WHL98], predictive maintenance [Mob02]). Due to the integration of social media applications (e.g., smart farming [BH15], social computing [Sch94, WCZM07]), the message format variety extends to non-textual data (e.g., multimedia data) [AAB⁺17]. In aftermath, the increasing data workloads and volume as well as integration qualities like velocity of message processing become more relevant differentiators for integration vendors. Moreover, the resulting integration scenarios and solutions become more complex compared to current on-premise integration, which makes a comprehensive solution even more desirable. While most of the integration logic is now operated outside of its

creator’s control (e.g., on edge or end-user devices, cloud platforms) [DWC10, KJM⁺11], it shall be possible to handle exceptional cases and responsibly develop functionally correct integration solutions. *Enterprise Application Integration* (EAI) [Lin00, Lin03, RMB02, Kel02] capabilities and systems shall address these recent trends, and thus allow for the explicit definition of integration semantics in a responsible manner for trust as well as efficient processing (cf. [FG13, AAB⁺17]).

However, for such a trustworthy EAI, comprehensive and sound application integration foundations are required that allow for a formal definition and reasoning (e.g., through experimental validation or formal verification) about integration scenarios and their improvements (e.g., optimization). Although EAI builds on a foundation in the form of *Enterprise Integration Patterns* (EIPs) [Hoh02, WB02, HW04] that abstract from common, complex integration domain aspects, the patterns are only informally described as best practices, as found in recent studies [FG13] and acknowledged by the EIP authors [ZPHW16]. Unfortunately, that leaves integration systems without an underlying formal foundation, and thus without formally defined integration patterns (incl. execution semantics) and their compositions, essentially integration scenarios. In fact, while the EIPs from 2004 are still relevant, they are by far not comprehensive and adapted to the new trends, again acknowledged by the EIP authors [ZPHW16]. While present in most if not all of the current integration system implementations (e.g., [IA10, SAP19a]), their informal description neither allows to use them for modeling due to missing execution semantics, nor to validate or even verify the correctness of current or new integration solutions [FG13]. Furthermore, the formalization of compositions of integration patterns as integration scenarios is absent (as will be shown in this thesis). Consequently, the user can neither check the functional correctness of an integration scenario nor predict, whether an improvement (e.g., optimization strategy) preserves correctness in advance. For trending deployments on edge devices, mobile phones and cloud platforms, this can impact not only business applications and processes, but also non-technical users that rely on the integration capabilities in their everyday lives.

In practice, vendors of classical EAI system (e.g., [Tib17, Sof17]) and more recently cloud integration systems (e.g., [DEL17, IBM17, SAP19a]), henceforth simply referred to as integration systems, have reacted with a plethora of vendor-specific and ad-hoc solutions for modeling integration scenarios, which are not grounded on such a formal foundation. As a consequence, up to now, none of the integration system vendors provides formally grounded analysis and optimization techniques. Moreover, while related data-centric domains like database and data processing or event processing already reacted to critical challenges from the discussed trends (volume, velocity and variety [KWG13]), integration system vendors still struggle to provide solutions up to the task. Ideally, it should be possible to specify functionally correct integration scenarios with techniques for formal analysis and the definition of correctness-preserving improvements, as well as instantiations into efficient runtime systems.

Thus, while business application vendors struggle to define their modular, but integrated core of their intelligent enterprise (e.g., [Tha06]) and end-users rely on integration capabilities in their daily lives, the underlying integration fabric is facing major problems that are currently unsolved.

1.1 Enterprise Application Integration

As a result of the mentioned technical, business and social trends, a transition from monolithic, packaged applications to modular, distributed, heterogeneous applications takes place (e.g., postmodern ERP [GKRH13]). While classical EAI systems constitute a centerpiece of current IT architectures [Lin00, HW04], the impact of these trends further increases the need for integration. To better understand what the term *integration* means in the context of EAI, we classify current integration approaches in Figure 1.1 by differentiating between several perspectives: application

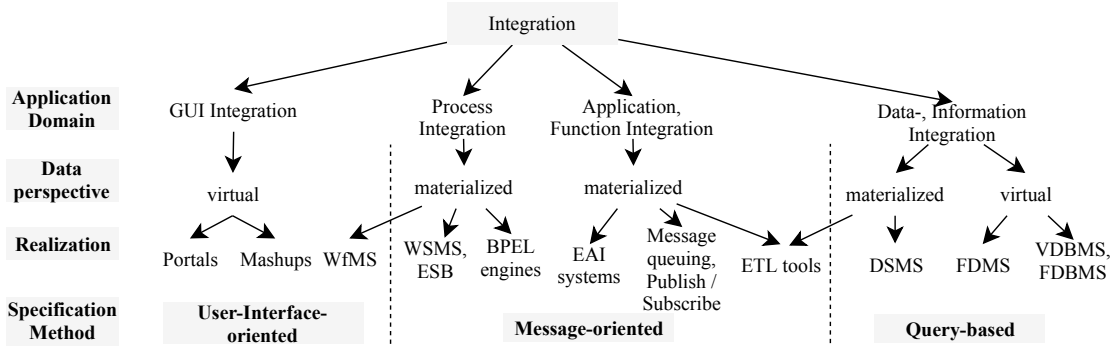


Figure 1.1: Different perspectives on *integration* (similar to [Böh11])

domain, data perspective, realization, and specification method, similar to [Böh11]. Thereby, we explicitly exclude orthogonal integration aspects such as schema matching or master data management, since these areas are typically not specifically designed for a certain integration approach. Regarding the *application domain* in the context of the *specification method*, we differentiate the user-interface-oriented *GUI Integration*, from message-oriented or application-tier *Process Integration* (e.g., [EH06, Kle13]) and *Application Integration* (cf. EAI), and the more database-tier, query-based *Data Integration* (e.g., [Len02, DHI12]) and *Information Integration* (e.g., [And81]). The term *message-oriented* or *messaging* denotes a data exchange via messages, and is thus different from other common integration styles such as *remote procedure call*, *shared database* and *file transfer* (e.g., see classification by Hohpe and Woolf [HW04]). While EAI systems are based on messaging, the latter two are predominantly found in data and information integration. The *data perspective* allows for the separation of integration approaches with respect to *materialized* and *virtual* data. The *realization* provides a concrete system aspect that represents the different integration approaches (incl. EAI). Subsequently, the term *integration* is discussed in the context of their specification methods.

GUI Integration denotes the integrated visualization of or access to heterogeneous and distributed data sources (e.g., in the form of *Portals* [Dia01, Fir07]). In contrast *Mashups* (e.g., [AT08]) are dynamically composed web content for small applications that focus more on content integration. Both approaches use a virtual, hierarchical network topology.

The categories *Application Integration* and *Process Integration* refer to a loosely coupled integration type, where messaging is used to decouple sender and receiver endpoints. While both approaches physically propagate and store — and thus materialize — data, application integration refers to the integration of heterogeneous, non-compliant system or applications, and process integration refers to a more business process oriented integration of homogeneous services (e.g., Web services [ACKM04]). In other words, application integration is more data-centric in terms of the efficient message routing and transformation, and process integration is more focused on procedural aspects in the sense of controlling the overall business process. In this context, there exist manifold realizations like ETL tools, *Message-oriented Middleware* (e.g., message queuing, *Publish-Subscribe*) [BCSS99, Cur04, BJ87, EFGK03] and EAI systems for application integration and *Business Process Execution Language Engines* (BPEL [Cor02]), *Web Service Management Systems* (WSMS [SMWM06]) and *Workflow Management Systems* (WfMS [GHS95, VDAVHvH04]), for process integration. Despite working on materialized data, the latter has a strong focus on user-centric tasks and interaction, and is thus categorized as user-interface oriented. Note that there were attempts to converge these system categories in the

past [Sto02, HAB⁺05], in the form of overlapping functionalities [Sto02]. For example, process integration standards such as BPEL are also partially used to specify application integration tasks (e.g., experimental evaluation by Scheibler et al. [SRL10]), however, that never gained practical relevance, due to the different concepts and processing models.

Finally, *Data and Information Integration* refers to database-centric integration approaches, where huge amounts of data are replicated and integrated, typically from one database to another [Len02, And81]. In this integration area, the data can be either virtual or materialized depending on its location [DD99]. Virtual integration denotes a (global) virtual view over distributed data sources, for which the data is not physically consolidated. Examples for virtual integration realizations are *Virtual Database Management Systems* (VDBMS) [KE11] and *Federated DBMS* (FDBMS) [SL90, KE11], which both use a hierarchical topology together with an event model of ad-hoc queries for dynamic integration. In contrast, materialized integration realizations physically store or exchange data for synchronization or consolidation purposes. An example for this integration type is *Data Stream Management Systems* (DSMS) [ACÇ⁺03, GÖ03, MWA⁺03], which allow for complex, continuous queries over the data. The related data integration via *ETL tools* [SHT⁺77, Vas09] follows a data-driven or time-based event model. Although these approaches conceptually use hub-and-spoke or bus topologies, ETL tools often use message-oriented data exchange and processing, and is thus categorized as such.

This thesis exclusively considers materializing, messaging integration approaches with a strong focus on application integration and EAI systems. However, the proposed solutions are applicable to process integration and partially information integration as well.

1.2 Challenges and Research Gap

Despite EAI being ubiquitous and of enormous relevance for companies [Gar16, For16b], the application integration foundations by means of EIPs, and EAI systems, have never fully been set into context to current trends. We identify IT trends and application scenarios which emerged after the collection of integration patterns by Hoppe and Woolf [HW04] from 2004, and study the performance of current EAI systems that implement the EIPs. Some of these trends, e.g., *Cloud* and *Mobile* computing, IoT, *Microservices*, and *API Management*, were even recently acknowledged by the EIP authors [ZPHW16]. The subsequently discussed trends, challenges and research gaps are analyzed in detail in Chapter 2 and the performance of current EIP implementations, assessed in the pattern benchmark in Chapter 5. The identified user-facing modeling aspects are briefly discussed, however, left for future work.

1.2.1 Emerging Application Integration

One major source for identifying new trends is the yearly published “Emerging Technologies Hype Cycle” report between 2005 and 2017 by Gartner [Gar17]. We focused on the most relevant trends for application integration today, i.e., we excluded trends like machine learning and analytics in the analysis presented in this work. The results are depicted in Figure 1.2. Both our literature review in Section 2.2.2 and our system review in Section 2.2.3 are consistent with the trends identified by the Gartner reports because both academic research as well as concrete systems address these trends.

Broadly speaking, the early years (2005 to 2007) are dominated by technology trends like *Service-oriented Architecture* (SOA) and *Event-driven Architecture* (EDA) styles (cf. WSMS and DSMS in the classification in Figure 1.1). Moreover related technologies like Microservices are mentioned by Gartner in 2017 [Gar17], and API Management by Forrester for 2016 to 2018 [For16c].

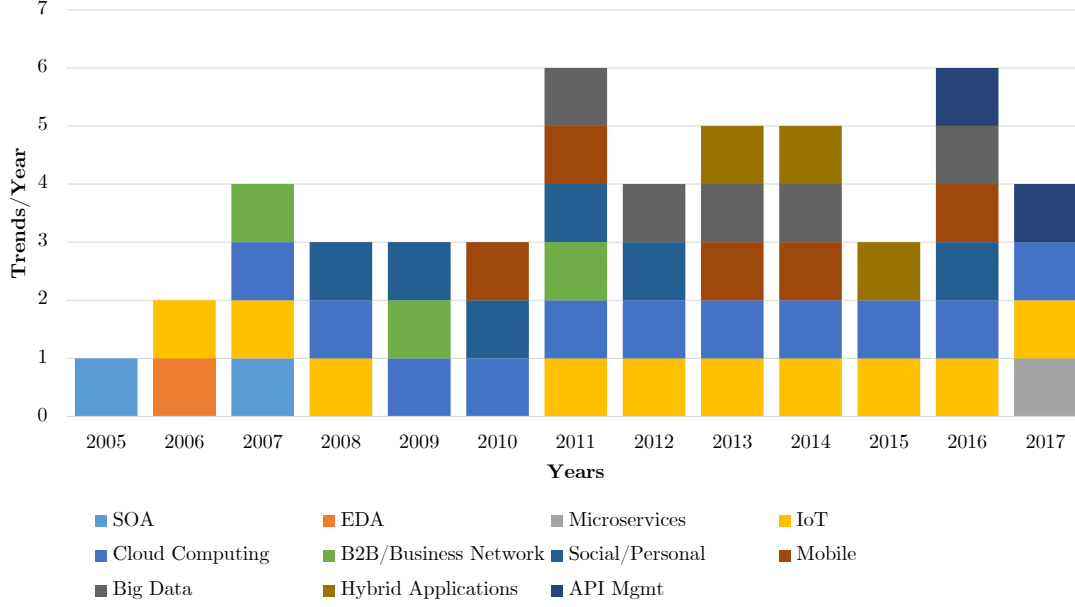


Figure 1.2: IT trends after 2004 in the context of application integration

The Cloud Computing trend became prominent in 2007 and subsequently led to trends like Hybrid [VT17, For16a] and Multicloud [BRC10, Pet13] Computing, from 2013 to 2015 and the move from B2B to cloud-based business networks. Early developments in the Internet of Things (IoT) became influential even before 2006 to 2008 even though devices were not yet affordable and widespread. However, with the advent of Mobile Computing in 2010, mobile and IoT devices and applications (since 2012) started to play a role for application integration. As countless devices and applications generated an increasing amount of data, *Big Data* (from 2011) became influential and a challenge not only for integration systems. Finally, humans increasingly organized themselves in social media with its momentum from 2008 to 2012, which evolves to personal computing, supported by wearable and mobile devices and applications.

In Figure 1.3 we associate the trends mentioned in Figure 1.2 with aspects of application integration. While some of the nodes represent the trends (i.e., without application and integration system), the edges denote required interaction and (transitive) communication, which also gives hints on existing as well as new integration scenarios for the different combinations. Node spanning trends are denoted by *dashed-line* nodes.

It is noteworthy that for hybrid application integration both on-premise to cloud as well as cloud to cloud communication become relevant, e.g., for migration of on-premise applications to the cloud. This raises technical issues like security but also robustness in the presence of errors or unavailable communication partners. Furthermore, cloud, on-premise and mobile applications generate communication traffic of an ever increasing scale with respect to the amount of data but also the number of communication partners. In this cloud setup organizations replace the bilateral RPC-style communication by asynchronous, message-based interactions which are mediated by integration systems. When the applications in an EAI scenario are partly hosted by cloud providers, *monitoring* becomes more challenging because the interfaces available for monitoring are often be limited. We also review these and other *Non-Functional Aspects* (NFAs)

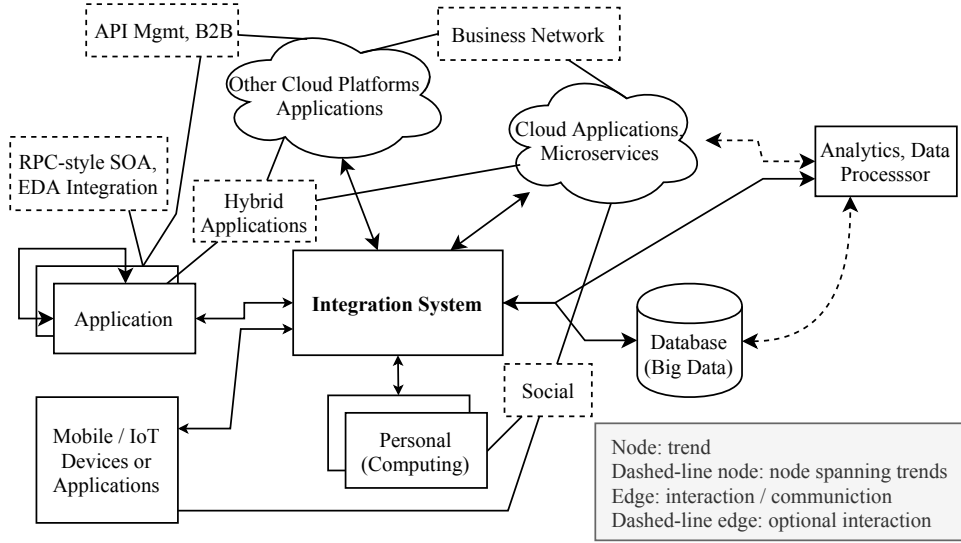


Figure 1.3: IT trends after 2004 and their relationship to application integration

in our literature review in [Section 2.2.2](#), our system review in [Section 2.2.3](#) and in the context of real-world integration scenarios in [Section 2.4](#).

1.2.2 New Challenges for Enterprise Application Integration

The classical EAI [[Lin00](#), [Lin03](#), [RMB02](#), [Kel02](#)] grounded on EIPs [[WB02](#), [Hoh02](#), [HW04](#)] provides solutions for challenges like the variety problem for textual message protocols until 2004 and beyond. As the EIP authors recently claimed, the patterns are still relevant, however, do not address the new challenges stemming from recent trends discussed before [[ZPHW16](#)]. Subsequently, we discuss the new challenges (Cx) for application integration addressed in this thesis in the order they are addressed in this thesis.

- C1 (Actuality, Comprehensiveness) While the EIP authors claim the actuality of the existing patterns [[ZPHW16](#)], they acknowledge missing pattern categories that recently arose through the trends or gained importance. Missing pattern categories concern *security*, *error handling* (incl. fault tolerance), *multimedia data*, *conversations*, *storage*, and *composition*, while extensions of existing categories are necessary for *processing* and *system management* (i.e., monitoring). Therefore, an extensive and systematic study is needed that tests actuality and strives for comprehensiveness through an extension of the EIP pattern language.
- C2 (Formalization, Trust) Even with an actual and comprehensive pattern language, the patterns and their execution semantics are still described by example. The current lack of formally defined patterns, compositions and improvements prohibits formal analysis and correctness guarantees (e.g., simulation, model checking) during specification or modeling and optimization, and thus *trust* in integration solutions (cf. Fahland and Gierds [[FG13](#)]). Moreover, the trending out-of-hand operation models (e.g., cloud, mobile / edge computing), demand a *responsible development* of integration solutions (e.g., [[Sta14](#)]). Therefore, suitable formalisms have to be selected or developed, which allow a formal specification of the patterns, compositions, and optimizations.

- C3 (Optimization, Complexity) The trends lead to increasingly complex integration scenarios as we will see on several occasions during this thesis. This is due to the growing number of heterogeneous communication partners, new processing models (e.g., synchronous streaming, persistent vs. asynchronous messaging), and evolving message formats. For the integration scenarios to remain tractable, suitable optimization strategies have to be identified and formalized that make the message processing more efficient and reduce the complexity of the scenarios.
- C4 (Justification) To judge the effect of a scenario improvement or a more efficient pattern implementation or solution, a method of justification has to be defined (e.g., in the form of a benchmark). However, currently no such method is available for measuring and comparing integration pattern implementations. Therefore individual pattern semantics (microscaling) have to be tested together with general aspects like increasing volume or concurrency (macroscaling), and compared for characteristic EAI workloads.
- C5 (Volocity) We conveniently combine the data-centric processing challenges of volume and velocity to what we call “volocity”. Although fast and high volume message processing has been a challenge and solutions have been proposed before (e.g., data reduction, parallelization optimization strategies), the new trends fundamentally challenge these solutions. For example, the conventional EAI system architectures on von Neumann architectures have problems when routing and transforming data (cf. [Chapter 5](#)). Therefore, new pattern solutions should be evaluated among the currently trending technologies for volocity cases, as well as justified and compared using a pattern benchmark.
- C6 (Variety) EAI addresses the requirement of integrating applications by solving the *variety* problem for heterogeneous, textual message formats, however, not for multimedia data and not for multimodal processing (combined textual and multimedia data). In this context a uniform configuration of integration scenarios (between pattern foundation and configuration or modeling level) is challenging. Therefore, the impact of multimedia data on EAI has to be evaluated and a multimodal processing and configuration model has to be defined.

To make the challenges more tangible, they are set into context of an integration system reference architecture composed from [[Lin00](#), [Lin03](#), [RH15](#)] depicted in [Figure 1.4](#). The diagram depicts the main aspects of an application integration system from [Figure 1.3](#), along the main challenges C1–C6 that arise in the context of the changes since 2004.

Challenge C1 (“Acuality, Comprehensiveness”) is at the center of [Figure 1.4](#), since it addresses the fundamental need of a comprehensive collection of patterns that are actually used. Such a catalog can probably not be complete, since new trends and thus new challenges will come up over time. Further, a comprehensive pattern catalog is required to approach challenge C2 (“Formalization, Trust”), which covers the formalization of patterns, their execution semantics and compositions. Only with the multitude of pattern characteristics of a comprehensive catalog, suitable formalisms can be selected, applied and evaluated. One application domain of such formalism on composition level is the formal treatment and study of correctness-preserving (cf. C2) optimization strategies, which are used to approach challenge C3 (“Optimization, Complexity”) as well as improved processing of high volume and velocity from challenge C5. This concludes the challenges with respect to the EAI foundation that denote kind of an intermediate layer between the design and runtime.

The runtime challenges target the realization of integration patterns as pattern solutions. There, a solution for challenge C4 (“Justification”) lays the ground for the evaluation of novel, more efficient pattern solutions demanded in challenge C5 (“Volocity”), which go beyond composition level improvements through optimization strategies. Finally, challenge C6 (“Variety”), i.e.,

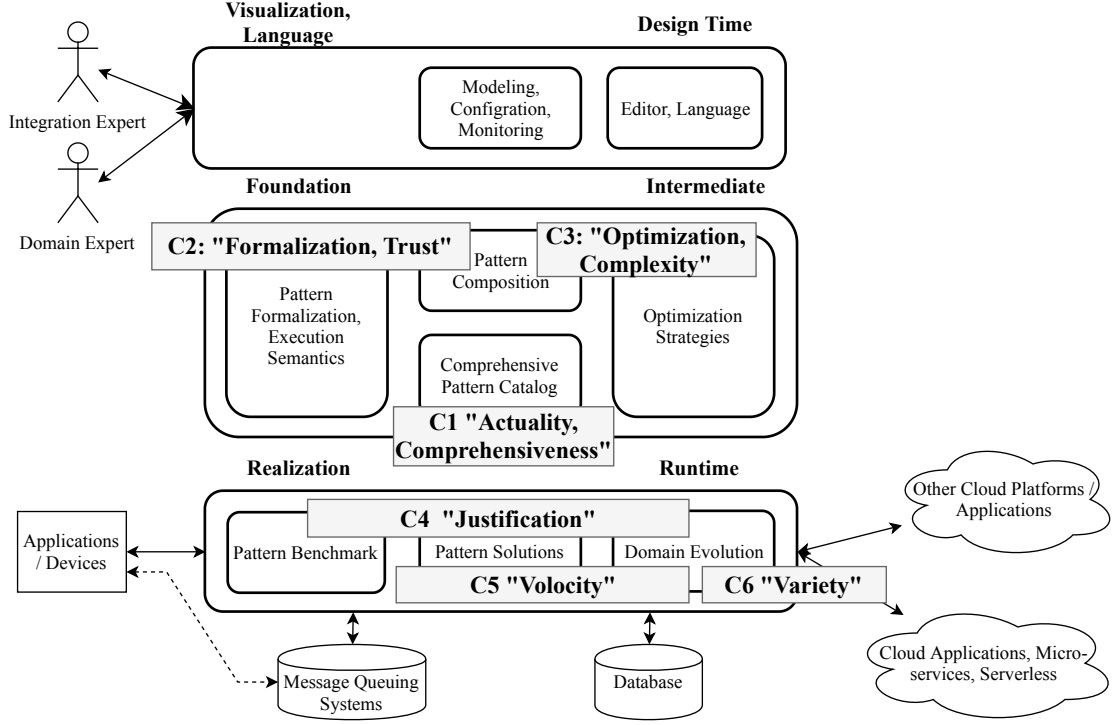


Figure 1.4: Conceptual EAI system overview with challenges

together with C5, constitutes a domain evolution of the classical EAI toward an emerged application integration definition.

The language design (e.g., for modeling, configuration, monitoring) and the visualization in suitable editors is out of scope of this thesis. However, Figure 1.4 illustrates that an attempt in that direction will find a well-defined EAI foundation.

1.2.3 Research Gap

Fulfilling challenges C1–C6 is essential for a trustworthy and efficient Enterprise Application Integration. However, the approaches presented in the EAI literature have disregarded these aspects so far, which denotes the current research gap. The combined trend and literature reviews show that for the trends and Non-Functional Aspects (NFAs) different solutions have been proposed, mainly missing formal foundations (i.e., patterns, formalization), and modeling from the literature study, as well as inefficient pattern solutions and missing justification capabilities.

Formal Foundations

Formal foundations in the form of formalized integration patterns and their compositions (essentially integration scenarios), denote the basis for *responsible development* of integration solutions.

Patterns Patterns are the predominant solution proposed in literature (cf. Table 2.1 in Chapter 2). Consequently, this work aims to close gaps by providing patterns for NFAs not present so far (cf. C1). Still pattern descriptions would be necessary in the context of the following

trends and NFAs. At first, *multimedia* functions are under-represented. Due to the access to the end user, multimedia becomes more and more interesting for all kinds of applications (e.g., sentiment analysis, monitoring in different domains like medicine or agriculture). For application integration, this targets the volume (cf. C5), variety and interoperability problems (cf. C6). The resulting increase of heterogeneity of media formats and communication partners (e.g., cloud applications, mobile devices, camera phones) demands revisiting the EIPs in the context of multimedia operations and their semantic aspects. Consequently, the increasing message sizes require the evaluation of optimization techniques (e.g., message indexing), and more efficient processing styles like *streaming*, which the EIP authors also acknowledge [ZPHW16], or data-aware integration pattern solutions (e.g., [Rit15b]). In general, to address the Big Data challenges of volume and velocity (cf. C5) requires corresponding benchmarks for pattern as well as for end-to-end system implementations, which are currently missing (cf. C4). As additional NFAs, only few of the *conversation patterns* are supported. For instance, Section 2.2.3 shows that conversation patterns can provide alternatives to improve the current processing and might become useful in more complex application or device interactions (e.g., device mesh [Gar17]). The monitoring of integration scenarios across multiple platforms (e.g., mobile, on-premise, cloud) — including aspects like raising indicators in case of an event — remains a challenge. This also hints on further work required for Mobile Computing and IoT, e.g., standardized protocols, conversation or interaction patterns (incl. data collection, device reconfiguration), and energy efficiency. Finally, as new trends and NFAs might constantly arise, their analysis with respect to pattern support becomes a continuous task (cf. C1).

Formalization Starting from the pattern view, formalization is an important step to precisely specify the semantics of the pattern realizations, as also acknowledged by Fahland and Gierds [FG12, FG13], i.e., formalization constitutes an important step towards the implementation and execution of the patterns in integration scenarios (cf. C2). As shown in Table 2.1 (on page 31) formalization approaches have been predominantly proposed in the context of Service-oriented Architectures (SOA) for validating and optimizing compositions by, for example, mapping them to Petri nets. Notably, a more formal definition of integration pattern compositions (also suggested by the EIP authors [ZPHW16]) is required. This would allow for currently missing structural validations, e.g., using Petri nets — as identified in Chapter 2, as well as semantic, runtime validations and optimizations on static scenario as well as dynamic, workload data (cf. C3). First work on the latter was conducted by Böhm et al. [BHP⁺11], however, it has to be revisited in the context of the trends and NFAs as well as new technical capabilities (e.g., machine learning of / for workload patterns, routing conditions, condition orderings). Furthermore, cloud, mobile and device computing raise new questions about responsible development of integration scenarios and correctness-preserving optimization strategies. In general, there is still an enormous potential for elaborating formalizations for both trends and NFAs, specifically, as a non-comprehensive set of patterns has been proposed by now. A follow-up research topic is how to implement patterns and pattern compositions in solutions using formal models (cf. overlap of C2, C5–C6).

Modeling

Except few works in the SOA domain providing modeling support for compositions (e.g., [HA10]), no attention has been paid to model integration-specific aspects so far. For compositions, business process modeling notations such as *Business Process Model and Notation* (BPMN) can be used, however, the integration-specific aspects exceed the modeling capabilities. Nevertheless, conveying information on the integration scenarios to users is of utmost importance for, e.g., maintenance and adaptations of these scenarios. Therefore, patterns might help to form the basis for different

modeling and visualization proposals. It could be envisioned to base integration scenarios on existing business process modeling languages (e.g., BPMN) in order to keep the mental map of users, but to enhance them with integration-specific icons. In our recent work [RR15], a first idea of equipping BPMN with integration icons is depicted for the SAP Cloud Integration eDocument use case. In general, NFAs like *security* and *multimedia* have to be further analyzed. Therefore, new visual configuration editors, e.g., allowing to “query by sketch” conditions, for integration scenarios would provide a more adequate, non-textual configuration. In addition, editors and visual data science (incl. machine learning) tools for scenario-based runtime monitoring, which are capable of dealing with large amounts of data, could lead to smarter (cross-) integration platform administration of integration scenarios. In this context, the development of visual modeling notations, new editors together with extensive user studies become necessary.

Efficient Pattern Solutions

When considering performance measures like message throughput and processing latency (cf. C5–C6), current integration systems show enormous weaknesses even in microscaling cases for simple routing patterns (cf. results in Chapter 5, e.g., the content-based router’s route branching and complex condition evaluation velocity) and macroscaling cases (e.g., increasing volume or concurrent invocations). However, the required method for justification (e.g., pattern benchmark) with configurable micro- and macroscaling, accounting for increasing volume and velocity as well as growing number of communication partners, is currently missing (cf. C4). Finally, the impact and challenges of multimedia data on EAI have not been studied so far (cf. C6). While this will not be limited to efficient pattern solutions (cf. pattern research gap), it will make the development of social media integration scenarios difficult.

Summary

In this thesis we aim to provide solutions for the research gap of a comprehensive pattern collection (i.e., addressing challenge C1), a formal pattern foundation (i.e., C2–C3), more efficient pattern solutions (i.e., C5–C6) and means for their justification (i.e., C4). The formal results of this thesis — especially the pattern composition formalism — might serve as a foundation for modeling approaches, however, we remind that design time aspects in terms of language design and modeling are out of scope, and thus we leave the modeling gap for further research.

1.3 Research Questions and Methodology

The most important aspects of the research gap concern the foundations of application integration in the form of patterns and their composition (incl. composition improvements by way of optimizations), as well as the efficiency of runtime systems, called *pattern solutions*, and their justification. Based on the pattern research gap, this thesis answers the following main research questions (*RQx*):

RQ1 To which extent are the current conceptual foundations of application integration in the form of the EIPs still sufficient for the new challenges and how can they be updated?

RQ1-1 Are the patterns from 2004 still sufficient in the context of new trends and challenges?

RQ1-2 Does the existing pattern language require extensions?

Research question RQ1-1 addresses the *actuality* of the patterns in the context of recent trends, and thus cover the claim of the EIP authors that the patterns are still valid today [ZPHW16]). In case existing EIPs have to be adapted or new patterns are required for the trends, question RQ1-2 targets a pattern identification and update process aiming to provide a *comprehensive* pattern collection. Together they should give an answer to overarching question RQ1 regarding the EAI foundation.

Assuming actual and comprehensive foundations as integration patterns, the formalization research gap is approached by those research questions:

RQ2 How to formulate integration requirements and scenarios in a usable, expressive and executable integration language?

RQ2-1 What is a suitable formalism for defining execution semantics of existing and new patterns?

RQ2-2 What is a sound and comprehensive formal representation of integration patterns that allows for formal validation of integration scenarios and reasoning?

RQ2-3 What are relevant optimization strategies, and how can they be formally defined on pattern compositions?

To answer the general research question RQ2, sub-question RQ2-1 sets out to formalize these foundations starting on a pattern level. Thereby a suitable formalization has to be selected and used to formally define the execution semantics of all collected patterns. In the context of question RQ2-2, which addresses the need for sound integration scenarios, formal analysis, simulation and validation techniques have to be already considered on the pattern semantics level. Together, the first two sub-questions allow for a *responsible development* of integration scenarios and solutions, and build the ground for a formal definition and treatment of *correctness-preserving* optimization strategies on the scenario level, addressed in RQ2-3. Thus, answers to RQ2 help to close the formalization gap for EAI towards *trustworthy* application integration.

Finally, the efficient pattern solutions and justification gap is addressed by the following research questions:

RQ3 Which related concepts and technology trends can be used to improve integration processing and how can the resulting integration solutions be practically realized and compared?

RQ3-1 Which related concepts and technology trends can be used to improve integration processing and how can this be practically realized?

RQ3-2 How can the benefits and improvements of pattern implementations be measured, validated and compared?

Data volume, velocity and variety challenges in EAI are shared with related domains like database management or data processing, and complex event or stream processing. Consequently, the overarching question RQ3 intentionally includes *related concepts* and their solutions, which answers to question RQ3-1 should consider. To assess specific solutions, the development of a method of justification is targeted by RQ3-2.

More detailed sub-questions are presented later in different sections of the thesis.

The overall process of answering the stated research questions has been guided by a *Design Science Research* approach (DSR) dating back to Simon [Sim96], which is tailored to research in information systems [Wie14]. In contrast to the natural sciences, which describe and observe natural objects and phenomena in order to understand reality, technology is conceptual as well,

however, embodied as in implementations or artifacts. In this way, information technology is the technology used to acquire and process information in support of human purposes, and thus instantiated as complex organizations of hardware, software, procedures and data in an information system [MS95]. While this targets questions like “does it work?” or “is it an improvement?”, natural science is concerned with how and why things are [MS95]. Walls et al. [WWES92] concluded that IS research builds on the behavioral science paradigm that has its roots in natural science. That means, design science research can be used to develop and justify theories within the context of information systems [HMPR04]. In design science research, the design process is divided into the following iterative steps: awareness of problem (proposal), suggestion (tentative design), development (artifact), evaluation (justification, performance measures) conclusion (results) [PTRC07]. A DSR artifact (e.g., design concept, prototype) [HC10] has to be evaluated in what is called deduction as part of the cognitive research process [VK15]. There, a tentative design is performed (design process) as prototype, which is studied and measured during the evaluation phase (design product) [HMPR04, WWES92].

Practically, these phases slightly differ in terms of research activities necessary within each phase depending on the research question [KGM12]. For example, research questions like RQ1 (and partially RQ3) that target a comprehensive study of the currently observable state and proposed hypotheses, will have to combine deductive and inductive research methods. Deductive methods are often subject to quantitative or exploratory research that targets a better understanding of or familiarizing with a topic by building hypotheses from an already existing theory and tests it by analyzing data [VK15]. With that, deductive actions allow for the derivation of more concrete design decisions, activities which eventually lead to an instantiated artifact and finally methods leading to a comprehensive evaluation concept allowing generalizations. In contrast, inductive actions allow for conclusions about the underlying design principles and theories, and thus denote rather qualitative research. A representative amount of data relevant to the topic or research question is gathered (e.g., discover integration patterns), patterns in the data are identified, and theories are developed that explain these patterns. This way, the inductive method fosters a deeper understanding, produces new knowledge and allows for broad generalizations from specific observations. Thus, deductive methods allow for testing of inductively generated theories.

This thesis pursues a mix of design science and in some cases explanatory research approaches. While gaining an overview of the current state of EAI together with the research sub-questions RQ1-1 and RQ1-2, as well as parts of RQ2-3 (i.e., relevance of existing optimization strategies) and RQ3-1 (i.e., impact of multimedia on EAI) will require elements of quantitative or exploratory and qualitative research methods, such as prototypes for the analysis and case studies (cf. [PRTV12]). The other research questions can be answered through design science research methods like design and evaluation of artifacts, again through instantiations in the form of prototypical implementations (e.g., for formalized patterns, compositions, optimization, benchmark, solutions), but also proofs (e.g., formal analysis), illustrative examples and case studies (e.g., real-world cloud integration scenarios). Note that artifacts like the development of concepts and their prototypical implementation in this thesis (e.g., for all aspects of questions RQ2 and RQ3) is in fact a constructive creation of artificial artifacts (something inherent to design science research).

1.4 Contributions

The main outcomes of this thesis are several DSR artifacts, the majority of which have already been presented in conference and journal papers. The following list presents these artifacts, the chapter in which they are presented in this work, and where these results have been published. A substantial portion of the contents in these chapters is based on the respective publications.

1. Systematic *literature review* concerning the research gaps, open challenges, and state-of-the-art in Enterprise Application Integration, denoting the overall related work (Section 1.2.3 and Chapter 2, respectively)
 - Daniel Ritter, Norman May, and Stefanie Rinderle-Ma. Patterns for emerging application integration scenarios: A survey. *Information Systems*, 67:36–57, 2017. [RMRM17]
Figures and tables from this paper are used similarly in Figures 1.2, 1.3, 1.5 and 2.7 and Table 2.1.
2. Assessment of the actuality of existing (answering research question RQ1-1) and identification of new integration or messaging patterns (cf. question RQ1-2), mostly, but not limited to integration adapters, exception handling, conversations and security as comprehensive *pattern catalog* (cf. RQ1; Chapter 2)
 - Daniel Ritter and Jan Sosulski. Modeling exception flows in integration systems. In *Proceedings of the 18th IEEE International Enterprise Distributed Object Computing Conference (EDOC)*, pages 12–21. IEEE, 2014. [RS14]
 - Daniel Ritter and Manuel Holzleitner. Integration adapter modeling. In *Proceedings of the 27th International Conference on Advanced Information Systems Engineering (CAiSE)*, pages 468–482. Springer, 2015. [RH15]
Figures from this paper are used similarly in Figures 2.4 and 2.12
 - Daniel Ritter and Stefanie Rinderle-Ma. Toward A collection of cloud integration patterns. *CoRR*, abs/1511.09250, 2015. [RR15]
 - Daniel Ritter and Jan Sosulski. Exception handling in message-based integration systems and modeling using BPMN. *International Journal of Cooperative Information Systems*, 25(02):1650004, 2016. [RS16]
 - Daniel Ritter, Norman May, and Stefanie Rinderle-Ma. Patterns for emerging application integration scenarios: A survey. *Information Systems*, 67:36–57, 2017. [RMRM17] (also listed above)
Figures and tables from these paper are used similarly in Figures 2.8 to 2.11 and Tables 2.2 to 2.7.
3. *Pattern formalization, formalized pattern catalog* and *prototype* as foundation of application integration systems (cf. RQ2-1; Section 3.1)
 - Daniel Ritter and Stefanie Rinderle-Ma and Marco Montali and Andrey Rivkin and Aman Sinha. Formalizing application integration patterns. In *Proceedings of the 22nd IEEE International Enterprise Distributed Object Computing Conference (EDOC)*, pages 11–20. IEEE, 2018. [RRMM⁺18]
 - Daniel Ritter, Stefanie Rinderle-Ma, Marco Montali, Andrey Rivkin, and Aman Sinha. Catalog of formalized application integration patterns. *CoRR*, abs/1807.03197, 2018. [RRM⁺18]
Figures and tables from these papers are used similarly in Figures 3.2, 3.3, 3.7, 3.9(a), 3.10(a), 3.10(b), 3.17(a), 3.17(b), 3.23, 3.26 and 3.27 and Table 3.1.

The presented prototype was developed in collaboration with Marco Montali and Andrey Rivkin from the University of Bozen-Bolzano.

4. Formalization of *pattern compositions*, representing integration scenarios (Section 3.2), and *prototype*, identification of optimization strategies as *catalog of optimization strategies* and their *formalization* (cf. RQ2-2 and RQ2-3; Chapter 4)

- Daniel Ritter, Norman May, Fredrik Nordvall Forsberg, and Stefanie Rinderle-Ma. Optimization strategies for integration pattern compositions. In Proceedings of the 12th ACM International Conference on Distributed and Event-based Systems (DEBS), pages 88–99. ACM, 2018. [RMFR18]
 - Daniel Ritter, Fredrik Nordvall Forsberg, Stefanie Rinderle-Ma, and Norman May. Catalog of optimization strategies and realizations for composed integration patterns. CoRR, abs/1901.01005, 2019. [RFRM19]
- Figures and tables from this paper are used similarly in Figures 3.34, 3.46, 4.1, 4.3, 4.5 to 4.8, 4.12 to 4.14 and 4.16 to 4.18 and Tables 2.2, 4.2 and 4.3.

The presented type theoretic realization was developed in collaboration with Fredrik Nordvall Forsberg from the University of Strathclyde.

5. Concept and implementation of an integration *pattern benchmark* covering textual data for the evaluation and comparison of novel pattern solutions (cf. RQ3-2; Chapter 5)
 - Daniel Ritter. Towards more data-aware application integration. In Proceedings of the 30th British International Conference on Databases (BICOD), pages 16–28. Springer, 2015. [Rit15b]
 - Daniel Ritter, Norman May, Kai Sachs, and Stefanie Rinderle-Ma. Benchmarking integration pattern implementations. In Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems (DEBS), pages 125–136. ACM, 2016. [RMSRM16]
 - Daniel Ritter, Norman May, and Stefanie Rinderle-Ma. Patterns for emerging application integration scenarios: A survey. Information Systems, 67:36–57, 2017. [RMRM17] (also listed above)

Figures and tables from these papers are used similarly in Figures 5.5 to 5.11 and Tables 5.1 to 5.8.
6. Novel *pattern solution* concepts and several *prototypes* with respect to *volume* and *velocity* (cf. RQ3-1; Sections 6.1 and 6.2)
 - Daniel Ritter. Database processes for application integration. In Proceedings of the 31st British International Conference on Databases (BICOD), pages 49–61. Springer, 2017. [Rit17a]
 - Daniel Ritter, Jonas Dann, Norman May, and Stefanie Rinderle-Ma. Hardware accelerated application integration processing: Industry paper. In Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems (DEBS), pages 215–226. ACM, 2017. [RDMRM17]
 - Daniel Ritter. Hardware accelerated application integration: challenges and opportunities. In Proceedings of the Second International Workshop on Active Middleware on Modern Hardware, ACTIVE@Middleware, page 15, 2017. [Rit17b]

Figures and tables from these papers are used similarly in Figures 6.1, 6.7, 6.9, 6.11 to 6.13, 6.15, 6.16 and 6.21 to 6.31 and Tables 6.2 to 6.5.
7. *Variety* concept, *prototype* of a pattern solution, and *benchmark* extension for multimedia data (Section 6.3)
 - Daniel Ritter and Stefanie Rinderle-Ma. Toward application integration with multimedia data. In Proceedings of the 21st IEEE International Enterprise Distributed Object Computing Conference (EDOC), pages 103–112. IEEE, 2017. [RRM17]

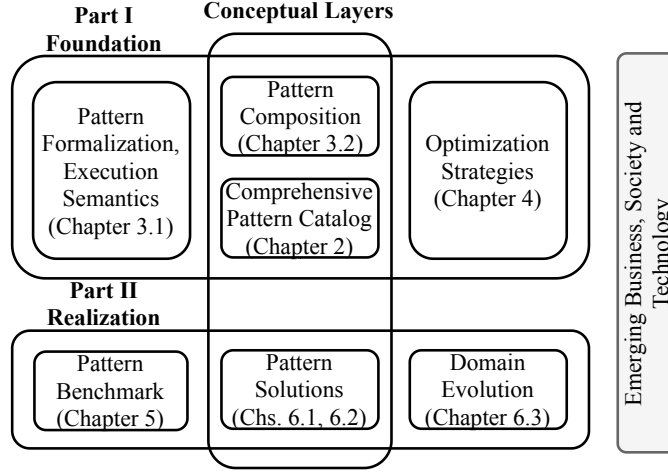


Figure 1.5: Overview of contributions and outline

Figures and tables from this paper are used similarly in Figures 6.33, 6.37 to 6.39 and 6.46 to 6.49 and Tables 6.6 to 6.9.

The main contributions of this thesis target the collection of new best-practices in EAI in form of patterns, the sound and complete formalization of the foundations in form of patterns, their compositions and improvements (Part I), and the study of pattern realizations as solutions (Part II) shown in Figure 1.5. Together, these contributions build the first *formal, trustworthy, multimodal* foundations of enterprise application and process integration that allow for *responsible* development of *efficient* integration solutions.

Out of scope Except for the multimedia data variety problem [RRM17], we do not address the currently vacant research field of application integration modeling and configuration (cf. modeling research gap), which we leave for future work. We refer interested readers to first solution attempts using process modeling [Rit14a, Rit14b, RS14, RH15, RS16], which we use in this thesis for the illustration of example integration scenarios.

1.5 Outline of this Thesis

The structure of this thesis follows the conceptual layers of EAI in Figure 1.5.

In the first part of this thesis (Part I), Chapter 2 investigates and outlines the current EAI foundations in the form of integration patterns, identifies, assesses and fills the gaps. The resulting comprehensive collection of integration patterns is fundamental for this thesis.

Consequently, Chapter 3 uses the pattern catalog as knowledge base for the analysis of their characteristics for the formalization of the patterns (cf. Section 3.1) and their compositions (cf. Section 3.2). Towards a trustworthy EAI, the compositions are linked to the pattern formalism to allow for a formal analysis on the overall execution semantics of integration scenarios. Leveraging and practically applying these results, Chapter 4 comprehensively studies the applicability of existing optimization and formalizes them compatible to the formalism used for pattern compositions, thus gaining correctness-preserving improvements of such compositions.

The second part of this thesis (Part II), more practically targets an efficient implementation of the formal results, called pattern solutions. Therefore, Chapter 5 provides means of justification

of pattern solutions in the form of a pattern benchmark, grounded on the analyzed pattern characteristics (called *microscaling*) and challenges from the general trends discussed in this thesis (called *macroscaling*).

As a more efficient implementation of our formalized pattern prototype in [Section 3.2](#), [Section 6.1](#) designs a pattern solution that approaches the data volume problem by an integration vectorization technique close to the actual data (i.e., parallel, micro-batch processing in relational databases). Then, [Section 6.2](#) leverages modern hardware for high velocity messaging through a specialization of EAI concepts close to the network. Finally, [Section 6.3](#) studies the impact of multimedia data on EAI overall (i.e., patterns, processing, configuration). Therefore the pattern benchmark is extended by multimedia data and specifications. Together, the pattern solutions show possible directions of the evolving EAI domain.

[Chapter 7](#) summarizes the results of this thesis and gives an outlook on future work (e.g., modeling language based on formalized pattern compositions).

Note that the chapters follow a common structure of introduction (not shown), preliminaries, analysis, solution, evaluation, related work and conclusion. However, for better understanding, [Chapters 3](#) and [4](#) deviate by moving the preliminaries closer to the corresponding solution, and the related work in [Chapters 3](#) and [6](#) is combined and moved to the end of the chapters. For the latter, the chapters end with a discussion of the conclusions in their respective sections.

Part I

Foundation

Chapter 2

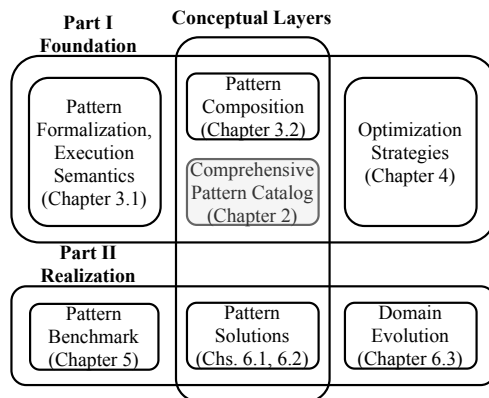
Application Integration Foundations

Contents

2.1	Integration Patterns, Compositions and Mining	21
2.2	Pattern Identification: From Best Practices to Patterns	28
2.3	Pattern Authoring and a Pattern Catalog	45
2.4	Quantitative Analysis	54
2.5	Conclusions	58

The limits of my language mean the limits of my world.
 Ludwig Wittgenstein, 1922 [Wit13]

In this chapter we study the impact of recent trends (e.g., technological, business and social) on the existing foundations of *Enterprise Application Integration* (EAI). These foundations



are embodied by the *Enterprise Integration Patterns* (EIPs) from 2004 [Hoh02, WB02, HW04] that were derived by EAI best practices at that time, and can be considered as the building blocks, when implementing or using an application integration system (e.g., by modeling integration scenarios) according to Fahland and Gierds [FG13, p. 1] and the EIP authors [ZPHW16, p. 18]. Inspired by the influential work on architectural patterns by Alexander [Ale77], one can argue that the discipline of computer science started with the identification and adoption of timeless knowledge from practical experiences in the form of patterns (e.g., in software engineering [GHJV95],

software architecture [BMR⁺96, Fow02], and more recently service interaction [BDTH05], conversations [Hoh06], and cloud computing [FLR⁺14]). These patterns capture the core of a solution to a reoccurring problem (e.g., [Ale77, GHJV95, Fow02]).

Similarly, the EIPs denote a structured documentation of abstract integration domain concepts as a catalog of interrelated patterns, representing a pattern language [ZPHW16]. However, considering Wittgenstein’s observation about “language limitations” [Wit13] and the influential trends since 2004, the *actuality* of these fundamental building blocks as well as the need for *extensions* have to be revisited. While the pattern authors recently claimed that the patterns are still relevant, they acknowledged that contemporary concepts are missing (e.g., exception handling, stateful conversations) [ZPHW16, p. 14, pp. 17–18]. The concern that recent best practices are missing is also shared by Scheibler [Sch10, p. 28], and thus, we address the general sub-research questions of RQ1 “*To which extent are the current conceptual foundations of application integration in the form of the EIPs still sufficient for the new challenges and how can they be updated?*” for *pattern actuality* and *pattern identification*, respectively:

RQ1-1: *Are the patterns from 2004 still sufficient in the context of new trends and challenges?*,

RQ1-2: *Does the existing pattern language require extensions?*,

which we study driven by the following derived hypotheses (Hx) in a *Design Science Research* (DSR) methodology (see Section 1.3):

- H1: The existing EIPs do not suffice for all application scenarios after 2004 (\rightarrow RQ1-1),
- H2: Current system implementations support patterns beyond EIPs (\rightarrow argument in favor of our study),
- H3: Some trends are handled in an (yet) immature and ad-hoc fashion (\rightarrow premise / justification for new patterns),
- H4: Solutions not in EIPs can be found in real-world integration scenarios for the trends (\rightarrow RQ1-2).

More concretely, hypothesis H1 aims to answer question RQ1-1 and hypotheses H2 and H3 provide arguments in favor of the study and justify the premise for new patterns, before hypothesis H4 settles an answer to RQ1-2. Answering these questions will not only help us to understand the domain in the context of new trends and challenges (cf. research challenge C1 “*Actuality, Comprehensiveness*”), but also provide comprehensive foundations (with new focus areas), addressing the *pattern research gap* in Section 1.2.

Therefore, the existing integration patterns from 2004 [Hoh02, WB02, HW04] are briefly introduced, and a pattern identification, authoring and application process from the pattern mining domain (e.g., [Har99, Han12, FBBL14]) is introduced in Section 2.1. Along this process, a new pattern identification approach is developed in Section 2.2 that allows for several results from the application integration domain since 2004: (a) research gap analysis, (b) systematic mining of reoccurring solutions (i.e., best practices), and (c) assessment of the relevance of the existing patterns. First, the research gap (a) results from a systematic literature study, which denotes the deductive part of our new mining approach. It reviews approaches closely related to application integration and integration patterns, which are set into context to the new trends since 2004. We recall that some of the results were already discussed in Chapter 1 as research gaps. Then, the results are combined with a systematic system review to inductively derive evidences for new solutions (b). The system review focuses on Non-Functional Aspects or requirements (NFAs) — related to the trends — identified during the literature review, thus leading to a list of potentially missing functionality. At the same time, the system review allows for checking the usage of the existing patterns in current systems (c). In the pattern authoring step in Section 2.3, the identified solutions are again described as patterns that extend the existing pattern language. The resulting extended pattern language is quantitatively evaluated with respect to its comprehensiveness,

novelty and relevance in [Section 2.4](#) through a study of real-world integration scenarios related to the trends and their usage of the new patterns.

Henceforth, we refer to the EIPs from 2004 as **original** and the more recently identified patterns as **extended**. Together they denote an integral part of the foundation of this thesis (cf. chapter overview), since they are instrumental for the further formal definition of a comprehensive, contemporary EAI foundation. As such, they serve as source of requirements for the formalization of single patterns, their composition as well as application (e.g., in optimization strategies). Moreover, a benchmark tailored to evaluating patterns as well as novel pattern implementations and systems are built on top of these pattern foundations.

Parts of this chapter have previously been published in the proceedings of EDOC 2014 [[RS14](#)] (new exception handling patterns), CAiSE 2015 [[RH15](#)] (new integration adapter patterns), a technical report [[RR15](#)] (extended pattern language), and as articles in the IJCIS 2016 [[RS16](#)] (new exception handling patterns) and IS 2017 [[RMRM17](#)] journals (pattern identification, authoring, and evaluation).

2.1 Integration Patterns, Compositions and Mining

First, we briefly introduce the existing integration patterns and their basic composition capabilities. Then we describe the pattern mining approach in the form of a pattern identification, authoring and application process.

2.1.1 Integration Patterns

Today, the state-of-the-art foundations of EAI and its system implementations still date back to the collection of EIPs by Hohpe and Woolf [[HW04](#)] in 2004. These patterns ground EAI on a message-oriented integration style with the characteristics of decoupling senders from receivers, and solving the connection and variety problems for textual message formats. More precisely, the message-oriented foundations are based on the *Pipes-and-Filters* architecture style [[PW92](#), [BMR⁺96](#), [Meu95](#)] (also known as dataflow), which became an architecture style [[BMR⁺96](#)] for the processing and exchange of data, and later an integration pattern [[HW04](#)]. The enterprise integration pattern catalog combines Hohpe’s and Woolf’s previous, independent work on “enterprise integration patterns” [[Hoh02](#)] and “patterns of system integration with enterprise messaging” [[WB02](#)], respectively, and actually denotes a pattern language¹ (i.e., a matrix of connected patterns [[Zdu07](#)]), which shall help a user during the decision process, when designing and building an integration system, as recently acknowledged by the EIP authors [[ZPHW16](#)]. Thereby, the pattern descriptions provide a practical, but structured documentation of expert knowledge according to a pattern format (e.g., name, problem / driving question, forces, solution, related patterns) that can be comprehensible for non-experts.

The **original** 61 messaging patterns from 2004 are listed and set into context to EAI system domain-aspects in [Figure 2.1](#)², and thus attempting a first look into the “black box” integration system from [Figure 1.3](#) (on [page 6](#)). The pipes-and-filters style allows for the use of direct communication as well as *message queuing* [[HW04](#)] for the wiring of the communication and integration architecture. They are categorized according to the EAI system domain-aspects, for which they provide an abstraction, by chronologically following the flow of a message and are briefly introduced and described subsequently.

¹We use the terms *pattern catalog* and *pattern language* synonymously, despite their subtle differences.

²© The EIP pattern icon, the pattern name, the problem and solution statements and the sketch are available under Creative Commons Attribution license.

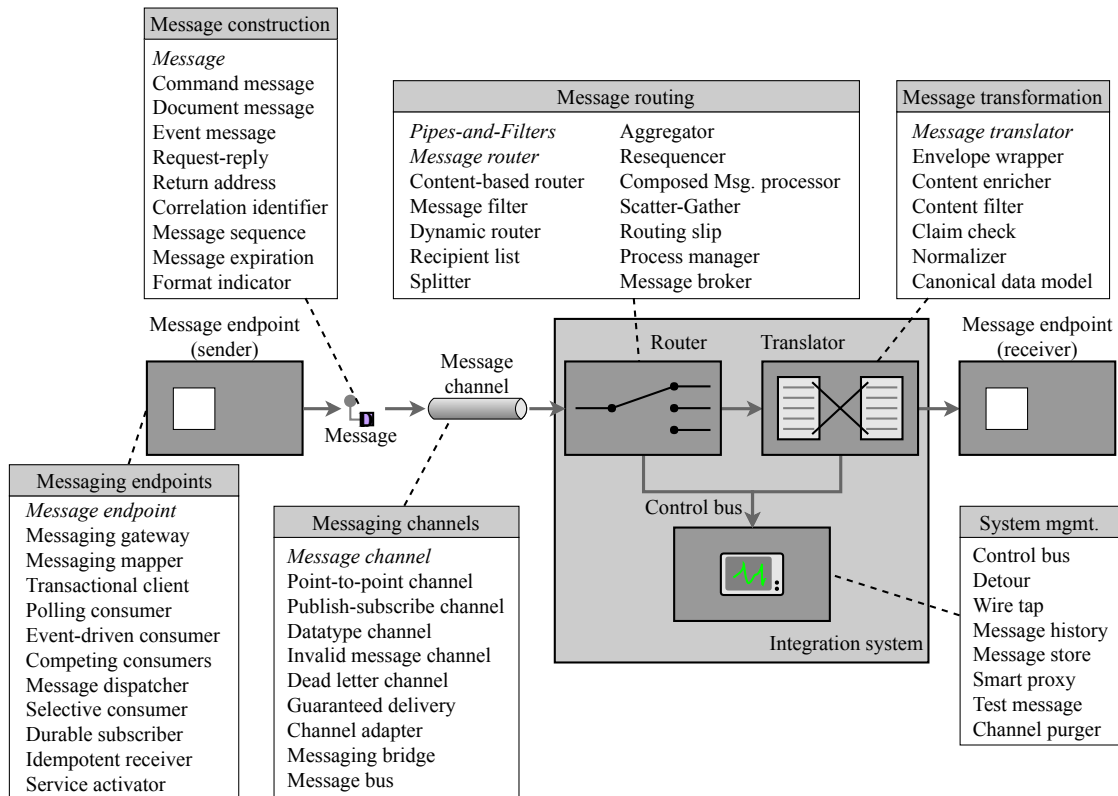


Figure 2.1: The enterprise application integration patterns from 2004 [HW04]

Message Construction The messaging patterns are grounded in the *messaging* integration style (e.g., in contrast to *shared database*) [RMB02, HW04], and thus the basic EAI domain-aspect is a *Message*. This category describes the different types of messages, which are usually required by an integration solution. A message is described as packets of data, which are exchanged via pipes (i.e., *Message Channels*) and processed by filters (i.e., *Message Routing*, *Message Transformation*). The message types range from a simple *Command Message* without usable information, over an *Event Message* with limited formats and data, to a *Document Message* with arbitrary, textual formats. A message can contain additional information like a *Return Address* and a *Correlation Identifier* (e.g., to identify the sender, the request in an asynchronous communication, respectively), or a *Message Sequence* (e.g., to decompose large messages to smaller chunks), which might be required during the message processing.

Message Endpoints The message exchange usually happens between abstract participants, called *Endpoints*. As indicated in Figure 1.3 (on page 6), endpoints abstract from different kinds of applications (e.g., business, cloud, mobile, social), services (e.g., SOA, EDA), and devices (e.g., mobile, sensor). The endpoints are connected to the communication channels, and thus act as *sender* or *receiver* of messages, as shown in Figure 2.1.

Message Channels The Message Channels facilitate the exchange of messages between one to many senders and one to many receivers (e.g., *Point-to-Point Channel*, *Publish-Subscribe Channel*). These channels transport messages of certain format (e.g., *Datatype Channel*), guarantee certain service qualities (e.g., best effort or at least once delivery [FLR⁺14, RH15]), and deal with delivery

failures (e.g., *Dead Letter Channel*, *Invalid Message Channel*). In addition, even more abstract concepts like *Messaging Bridge* (i.e., connecting multiple messaging systems) or meta concepts like *Message Bus* (i.e., denoting a *Message-oriented Middleware* [BCSS99, Cur04]) are part of this category.

Message Routing The routing patterns describe different ways to determine the recipient(s) of a message. The sender does not need to know the actual receiver, which essentially denotes the aspect of *decoupling* allowed by the patterns. Besides the simple structural routing from pipe to filter, the routing requires access to the message’s data (e.g., *Content-based Router*, *Recipient List*) or even changes the message for a correct routing (e.g., by decomposing one message into several messages using a *Splitter*, or combining several, related messages to one with an *Aggregator*).

Message Transformation The *variety* and potential incompatibility of the textual formats of heterogeneous endpoints require message transformations from the senders’ formats into formats understood by the recipients of a message. The transformation patterns provide a solution to this second decoupling aspect, allowing the endpoints to remain unchanged. Within the integration system, the message processing patterns require a standardized access to the data (i.e., common message format) in the form of a “lingua franca”, which is provided by the *Canonical Data Model*, thus solving the variety problem for textual message formats. The solutions contain changes to the data (e.g., *Content Filter*, *Content Enricher*), contextualization of the data (e.g., *Envelope Wrapper*) as well as transport of references with a claim to stored messages that can be materialized using the claim (e.g., *Claim Check*).

System Management Loose coupling as well as distributed applications are more difficult to administrate than a monolithic system. Therefore, a basic set of *System Management* patterns allows for the monitoring (e.g., *Control Bus*, *Wire Tap*, *Detour*), tracing (e.g., *Message History*), and testing of the system (e.g., *Test Message*).

2.1.2 Integration Pattern Composition

Most of the single patterns denote atomic building blocks to compose more complex integration logic. Other patterns like *Request-Reply* or *Composed Message Processor* are compositions of the atomic patterns [HW04]. The atomic and composed patterns can be wired by direct communication links and Message Channels to *integration scenarios* in a pipes-and-filter style, also called *pattern compositions*. As indicated in Figure 2.1, the filters are single patterns that are visually represented by a dedicated, explanatory icon (e.g., Router, Translator, Control Bus). The resulting basic pattern compositions denote an icon notation, introduced by [HW04], which mainly represents the control flow of the pattern execution, however, with data flow indications (e.g., message icons, channels).

Example 2.1. The synchronous communication between two endpoints can be realized by the Request-Reply pattern, shown in Figure 2.2(a)³. A requesting participant constructs a request message *Request*, which is transported by a Message Channel *Request channel* to a responding participant, returning the corresponding *Response* message to the request via the reply channel. The requesting participant does not need to wait for the reply, before continuing the processing.

While the Request-Reply pattern describes the (a)synchronous interaction of endpoints, by combining messages and channels, Figure 2.2(b)³ shows a more complex composition of atomic patterns representing a Composed Message Processor pattern. This message processor splits an incoming message *msg*, into its parts and routes them individually to subsequent processors. Since the concept of an arbitrary filter (e.g., user-defined function) does not exist in the original EIPs,

³© The EIP pattern icon, the pattern name, the problem and solution statements and the sketch are available under Creative Commons Attribution license.

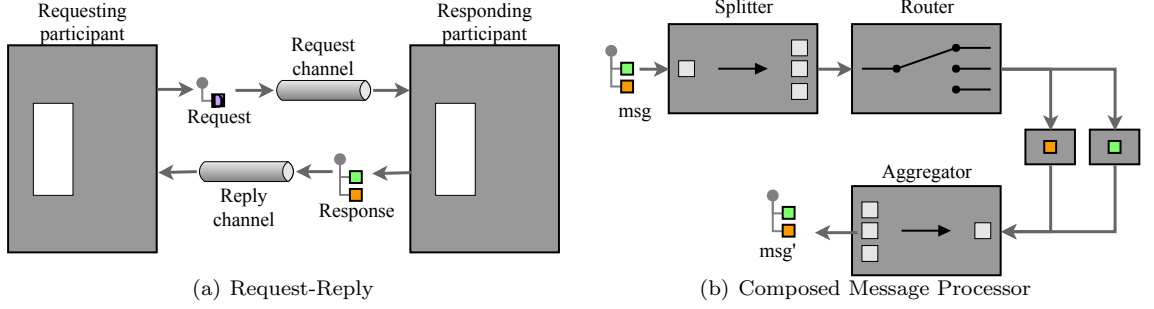


Figure 2.2: Pattern composition in EIP icon notation (notation from [HW04])

these processors are indicated by unnamed filters. After the processing, the resulting and related message parts are aggregated according to an aggregation strategy and based on a completion condition (not shown). The output is a new message msg' . ■

The two composed pattern examples in EIP icon notation illustrate the capabilities of describing integration scenarios on an architecture documentation level. However, recent studies by Fahland and Gierds [FG13, FG12] as well as this work (see Section 2.3) suggest that the EIPs do not denote a modeling language. The pattern authors acknowledged that the EIPs rather represent guidelines, when using an integration system [ZPHW16]. Due to only little and yet insufficient work on a modeling language (e.g., with respect to execution semantics), we fall back to one approach based on the *Business Process Model and Notation* (BPMN) [Gro10] to illustrate the initial integration scenario examples in this work. The patterns [Rit14b] and their compositions [Rit14a] are represented by BPMN *Collaboration Diagrams*, which allow for a more detailed description of the control and data flows and other (novel) EAI concepts like those developed in this work like integration adapters [RH15] and exception handling [RS14, RS16] (see Section 2.3).

Example 2.2. Figure 2.3 shows the BPMN representation of the two patterns from Figure 2.2 in a more expressive way. Thereby, the Request-Reply pattern is denoted by Figure 2.3(a), by a *Requesting participant* and a *Responding participant*. While the control flow is equally denoted by BPMN *Sequence Flows*, the data flow uses a combination of BPMN *Message Flow* and *Data Object* with *Data Association* elements that are grounded in BPMN data definitions unlike in the EIP icon notation. The synchronous behavior is given by a BPMN *Service Task*, called *Synch. Call*, which receives the request and forwards it to the receiver. The response message is returned and any errors are handled by a BPMN *Intermediate Error Event*. Similarly, the Composed Message Processor in Figure 2.3(b) allows for superior concepts compared to the EIP icon notation like annotations for multiple-messages forwarded by the Splitter, conditional and default *Sequence Flows*, and different BPMN *Activity* types like the BPMN *Script Task*. Furthermore, the Aggregator is a complex pattern that can be abstracted as BPMN *Call Activity*, denoting that more complex logic is linked. ■

Despite the richer and seemingly suitable syntax of the BPMN-based approach (compared to the EIP icon notation), the BPMN execution semantics differ from those of the patterns (e.g., in terms of message processing [Rit14a] and representation of integration adapters [RH15]). For further details we refer to our previous work on modeling EIPs [Rit14a, RH15, RS14, RS16].

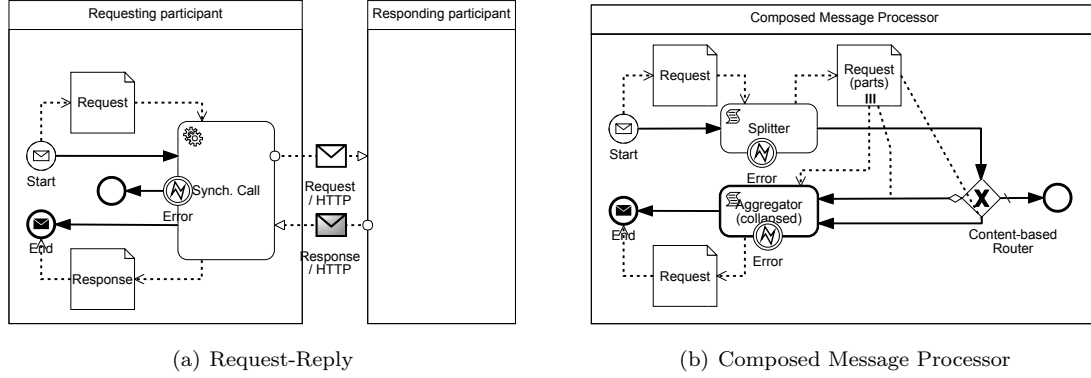


Figure 2.3: Pattern composition in BPMN (notation from [Rit14a])

Hence, in this work, the BPMN diagrams are used to illustrate certain integration scenarios and their aspects, but cannot be seen as a sound and complete integration modeling language, and thus left for future work.

2.1.3 Building Blocks of Integration System Architectures

The notion of the patterns as “building blocks” of integration systems is supported by Trowbridge et al. [TRH⁺04], who describe how to design and build an integration architecture on integration patterns. However, these architectural patterns (e.g., portal, process and data integration) are not to be confused with the EIPs, on which most of the current integration system implementations build on. For example, the open source integration system Apache Camel described by Ibsen and Anstey [IA10] largely follows the patterns as building blocks, and similar to Scheibler [Sch10], attempts to make the EIPs configurable and executable.

From these system implementations, a common integration system architecture was captured in [RH15], which is set into context to the EIPs, where possible, in Figure 2.4. The message endpoint patterns are depicted as *sender systems* and *receiver systems* that can engage in different kinds of conversations (e.g., cf. conversation patterns by Hohpe [Hoh06]) and connect to the integration system through integration adapters that are partially in the message construction patterns. These adapters are differentiated by the direction of the data flow as *consumer adapters* or *producer adapters* (vendor-specific variants in terminology possible). The tasks of the adapters are physical Message Channel, protocol (e.g., HTTP, FTP, JMS) and security handling, and format conversions (e.g., XML, JSON, CSV) from the external to the integration system internal format, the Canonical Data Model [RH15]. For security and operations of the adapter (e.g., monitoring, scheduling), it has to access a *secure / key store* and *operational database*, respectively. The constructed messages are distributed via a Point-to-Point Channel or Message Queuing patterns, which require access to the operational database for monitoring and service quality assurance. An integration runtime executes compositions of message routing and transformation patterns, henceforth also referred to as *message processors*. The message processor compositions are depicted as *integration programs* (also integration processes), while we call the end-to-end integration scenario *pattern composition*, since it consists of patterns. A message processor requires access to the operational resources according to their characteristics. A small amount of these operational capabilities are in the System Management patterns.

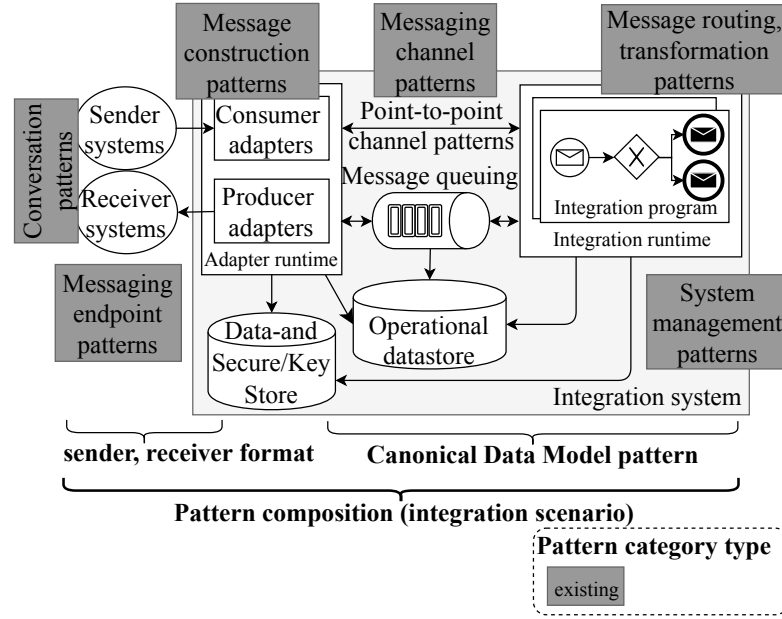


Figure 2.4: Integration system architecture with corresponding pattern categories (dark grey)

In contrast to patterns as a modeling language, the EIPs denote an approximation of practical integration system implementations, and thus can be regarded as their building blocks or at least conceptual guidelines, when implementing and using integration systems. However, logical next questions with respect to the claim by the pattern authors that “the patterns are still used” [ZPHW16] (actuality) and “whether further best practices arose after 2004” are left unanswered. Therefore, a systematic study of existing and a mining of new patterns is required (e.g., in the form of a pattern engineering process).

2.1.4 The Pattern Engineering Process

In the computer science “pattern community”, the pattern identification, authoring and application processes are defined and discussed by communities like the “Pattern Languages of Programs” (PLoP) conference, where the EIP authors presented their first pattern catalogs in 2002 [WB02, Hoh02], as well as the “Transactions on Pattern Languages of Programming” (TPLoP) journal and regional communities like “European Conference on Pattern Languages of Programs” (EuroPLoP). *Authoring* denotes the process of describing and linking a pattern to others and *application* stands for the implementation of a pattern in a certain context [FBBL14], this is also called a *pattern solution*. Since the identification and authoring of patterns is by large a manual process, common guidelines for a *pattern engineering process* (i.e., essentially *mining* patterns for a domain) exist in these communities to ensure a continuous / iterative, systematic and rigorous procedure (e.g., [Har99, Han12, FBBL14]). The resulting patterns are essentially structured textual documents, describing abstract problem-solution pairs of design problems recurring in a specific context [FBBL14].

Due to the existence of the EIP pattern catalog from 2004, this catalog is extended rather than defining a new one. For the assessment of its actuality and potential extensions, we rely on the continuous pattern engineering process of pattern identification, subsequent authoring

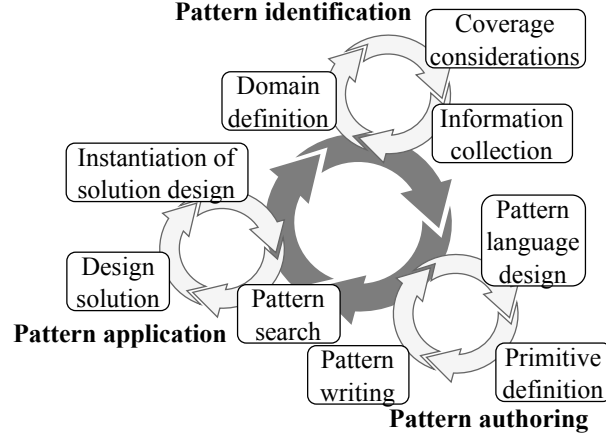


Figure 2.5: Pattern engineering process (adapted from [Han12, FBBL14])

and application as shown in Figure 2.5. The process essentially constitutes an adaptation of the procedures by Fehling et al. [FBBL14], used in the context of cloud computing patterns, and more general work on pattern mining by Hanmer [Han12]. The distinct phases are subsequently described from pattern identification over pattern authoring to pattern application.

Pattern Identification In the first phase, the relevant information of a domain is structured and collected. In the *domain definition* step, the information sources are identified, which are then constrained to a subset of only the relevant sources in the *coverage considerations* step. The information from these sources are captured for found solutions in the *information collection* step according to an information format, which are further refined and structured (not shown).

Pattern Authoring The information collected during the previous phase is used to identify similarities between existing solutions. Based on these similarities, the patterns can be authored. Therefore, in the *pattern language design* phase, a format for the pattern language has to be specified. For example, we evolve the existing EIP language design by the results of a comparative study of different pattern languages in computer science. The pattern format homogenizes the information and allows for a structured access by its users. In the *primitive definition* step, the descriptions of the pattern primitives are revised and extended. In our case, the existing patterns are matched with respect to potential alternative usages and new aspects. Eventually, the patterns are properly authored during the *pattern writing* step based on the captured knowledge from the previous steps. In this step, we additionally assess the relevance of the existing patterns by checking their coverage in the information sources. Then the consistency of the patterns within the pattern language is revised (not shown).

Pattern Application The application of the patterns can be performed independently, done in a later part of this work. The application phase starts with the *pattern search* step that helps to find a suitable pattern for the task at hand. The solution design is tailored to the technological domain in the *design solution* step, to which the resulting implementation of the pattern is deployed in the *instantiation of solution design* step. The variance of these implementations might provide additional information considered in the pattern identification phase.

Although it is unknown whether Hohpe and Woolf followed such a pattern engineering process, when describing the EIPs, we apply the identification and authoring phases in the subsequent sections, while the pattern application phase is performed in Chapter 6.

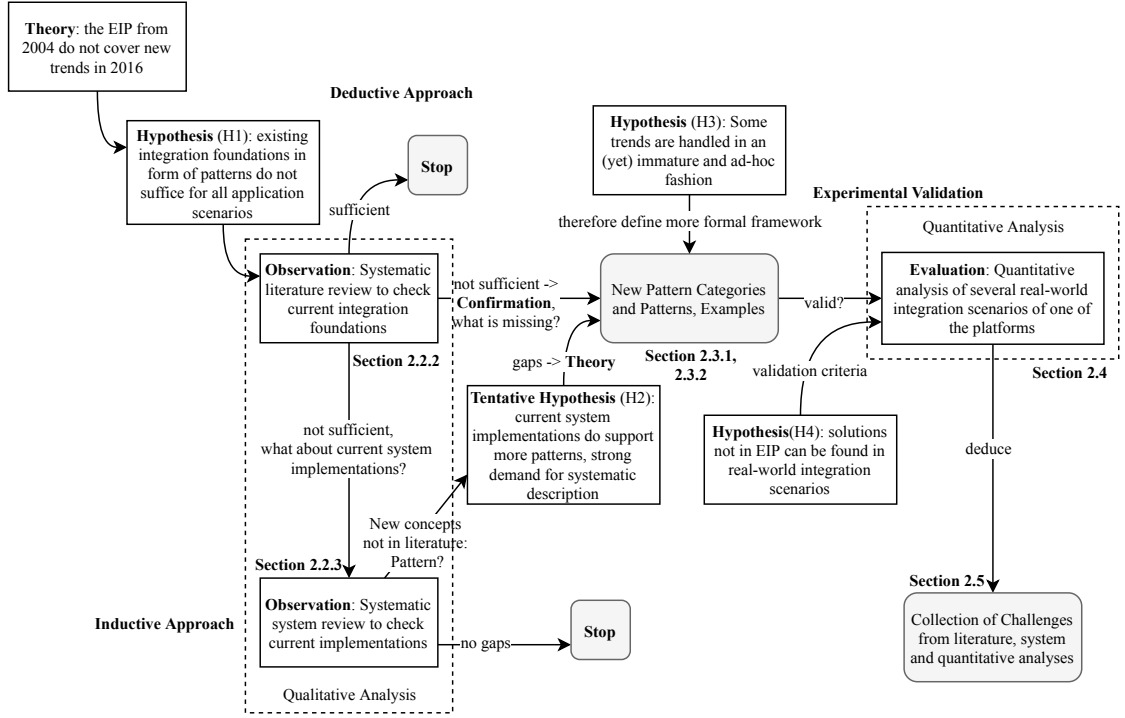


Figure 2.6: Design science methodology used for systematic pattern identification

2.2 Pattern Identification: From Best Practices to Patterns

This section focuses on the identification part of the introduced pattern engineering process and focuses on the analysis of the research gap, the assessment of the relevance of the existing patterns and the mining of new integration patterns following trends since 2004.

2.2.1 Pattern Identification Methodology

The proposed engineering approach relies on the design science methodology (see Section 1.3) as a rigid method to collect and evaluate new trends mentioned above, to summarize research which adds new patterns to the original EIPs, and to evaluate these new patterns in the context of real world application scenarios. Figure 2.6 depicts the research method applied for a systematic structuring and execution of the pattern identification phase.

Our fundamental **theory** and motivation for the pattern engineering is: *The original EIPs from 2004 do not completely cover new trends in 2016 and beyond.* From this we derive **hypothesis (H1)**, i.e., *the existing EIPs do not suffice for all application scenarios after 2004.* This hypothesis is tested based on two observation artifacts, i.e., a systematic literature review in Section 2.2.2 and a systematic system review in Section 2.2.3. Based on the literature review we analyze whether new trends and application scenarios can be seen after 2004 and which solutions are provided. The system review aims at analyzing available systems regarding their support for integration. Interpreting the literature and system reviews then leads us to the tentative **hypothesis (H2)** that *current system implementations support patterns beyond EIPs* which results in a *strong*

demand for systematic description. In order to address the detected gaps we propose new pattern categories and patterns. In **hypothesis (H3)** we argue that *some trends are handled in an (yet) immature and ad-hoc fashion*, and thus require a structuring in the form of patterns. These artifacts are then evaluated based on a quantitative analysis of several real-world integration scenarios following the **hypothesis (H4)** *solutions not in EIPs can be found in real-world integration scenarios for the trends*. Finally, resulting research directions are described.

2.2.2 Deductive Analysis: Literature Review

We conduct a literature review in order to evaluate the hypothesis *H1: existing integration foundations in the form of patterns do not suffice for all application scenarios* as set out in Figure 2.6. The hypothesis raises two questions to be investigated in the literature review, i.e., i) “are there any topics after 2004 not yet covered by the original EIPs?” and if yes ii) “do existing approaches provide solutions to these topics?”.

The literature review is based on the guidelines described by Kitchenham [Kit04]. The primary selection of references was conducted using google scholar (scholar.google.com) on 2016-10-4. The search string was

allintitle: integration patterns

excluding patents and citations. As a general baseline, only papers after 2004 are considered as the main theory behind this study is that *the EIPs from 2004 do not cover trends in 2016*. Hence the time range was set to 2005 – 2016. Overall this resulted in 525 hits. On these hits, the following selection criteria were applied:

- relation to computer science, enterprise application integration, service integration, data integration, system integration
- availability of the document
- written in English
- published in peer-reviewed venues

Altogether, 52 papers were selected as relevant. These 52 papers were further analyzed whether they contribute as observations to the hypotheses. This resulted in removing 23 papers from the primary literature list (for example, papers were excluded that focus on data integration). Then a vertical search was conducted in forward and backward direction, resulting in 43 papers, including one paper that was added based on expert knowledge. After analyzing these papers, 34 were included in the secondary literature list. Overall, this results in 63 papers for the secondary literature list.

Processing of Selected Literature – Topics and Trends

At first, all papers from the secondary literature list were analyzed with respect to the topics they mention. Comparing the harvested topics with the trends identified in the introduction gives an answer to question i) “are there any topics after 2004 not yet covered by the EIPs?”. In this first step it is sufficient that a topic is mentioned. It was not necessary that a solution was provided. As the collected topics are very fine granular and spread widely, they were first grouped according to the trends mentioned in the introduction.

Figure 2.7 depicts the distribution of topic mentions along trends over time. It can be seen that SOA (i.e., RPC-style integration [HW04]) plays a dominant role, particularly in the years 2005 – 2013. During this period, some topics such as mashups, cloud, and EDA were occasionally mentioned. In the last years, i.e., 2014 – 2016 the picture seems to change, turning away from the strong focus on SOA towards topics such as cloud, hybrid, and IoT.

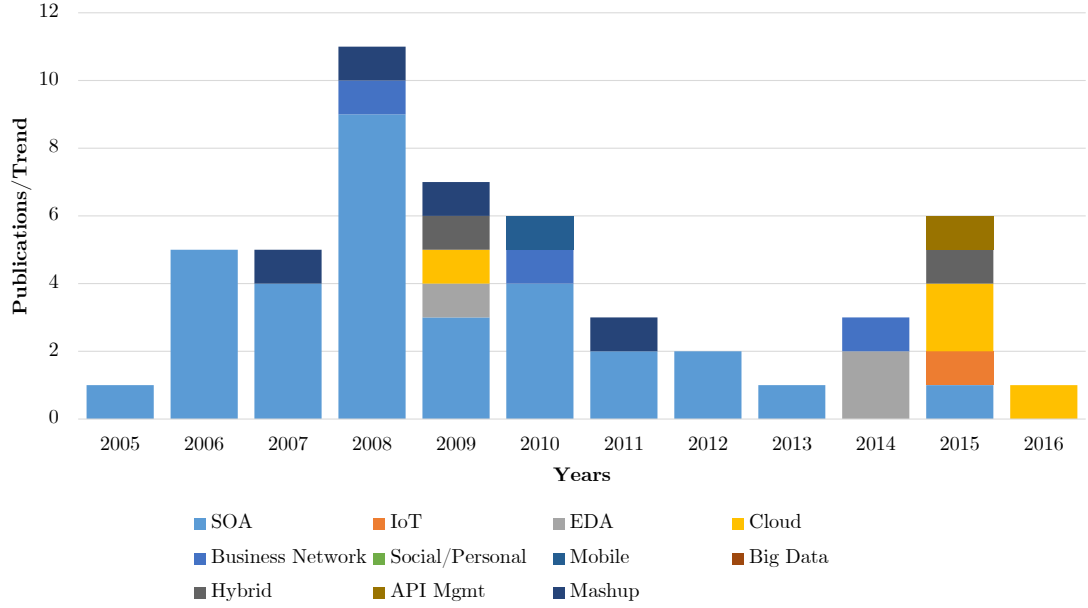


Figure 2.7: Distribution of topics mentioned in literature over time

From Figure 2.7 it can be concluded that some of the trends occurred in the literature after 2004 with a dominant occurrence of SOA. Apparently, since 2014 SOA loses significance, and other trends such as IoT and cloud seem to gain more attention. From the dominance of SOA we also conclude that a more fine-grained analysis of the mentioned topics is meaningful. Hence, in the following, summaries for the analyzed approaches are provided, ordered by the topics and areas they work on. Subsequently, the list of trends will be complemented with Non-Functional Aspects or requirements (NFAs) mentioned by literature that constitute further important topics for EAI since 2005. Moreover, if approaches provide solutions with respect to the different topics, the type of solution will be collected.

Literature Summaries

The approaches identified in the literature search are subsequently summarized. We organize the summaries chronologically by following the timeline from Figure 1.2. In the context of EAI, no work was found on the internet of things, social / personal computing, microservices, and API management, which can be seen as successor of the Service-oriented Architecture trend. In addition to the trends, for each approach we try to derive additional NFAs as well as the proposed solutions. The harvested NFAs and solutions are summarized at the end of the section in Table 2.1 and yield the input for the further analysis.

Service-oriented and Event-driven Architectures According to the timeline, the first EAI solutions after 2005 were provided by Service-oriented and Event-driven Architectures representing mostly RPC-style solutions (i.e., a post shared database and file sharing integration style, compared to *messaging* like EIPs, according to [HW04]).

Service-oriented Architecture. Hentrich and Zdun present patterns that address data integra-

Table 2.1: Solutions for trends and non-functional aspects (parentheses mean partial solution)

Trends	Patterns [HW04]	Formalization [ZPHW16, PJGM12]	Modeling [Rit14a, Rit15c, PJGM12]
Service-oriented Architecture	[HZ06, HZ07, HZ09, ZHVDA06, Zdu08, ASPT15, GDP09, KB12, FSLM06, BDTH05, KH06], security [QYZL05, SP08]	[GDP09], adapters [GMW12, SEG08, KS08, JZGH11, WDOV08], control flow [OVVDA ⁺ 07], interact. [LMSW08]	[HA10]
Internet of Things			
Event-driven Architecture		[PSPP14, PPSP14]	
Cloud computing	[RR15, MSHP15], (migration [AFP16])		
B2B / Business Network			(by example [Rit14a, Rit15c])
Social / Personal Computing			
Mobile Computing	SOA device patterns [MLK10]		
Big Data			
Hybrid Computing	(migration [AFP16, Man09])		
API Management	(for SOA [ASPT15])		
Mashups	[LLX ⁺ 09, LLX ⁺ 11], SOA migration [CAO ⁺ 07]		
NFAs with evidence			
Asynch [HW04]	EIPs [HW04], strategies [GR11]	[FG12, FG13]	
Security [ASPT15, SP08, MSHP15, HOS ⁺ 15, RCVB10, HDX14]	(for SOA [QYZL05, SP08])		
Media [Gar17]			
Synch / Streaming [ZPHW16]			
Conversations [ZPHW16], [Gar17]	[Hoh06], (for SOA [BDTH05])	(for SOA [UP06, LMSW08])	
Error Handling [ZPHW16, MSHP15]	(EIPs [HW04, Hoh05])		
Monitoring [MSHP15, MLZN09, MN10]	[(HW04)]		

tion issues such as incompatible data definitions, inconsistent data across the enterprise, data redundancy, and update anomalies [HZ06]. It is described how to integrate the application-specific business object models of various external systems into a consistent process-driven and service-oriented architecture. In summary, the proposed solution combines SOA with patterns, e.g., refactoring patterns. In [HZ07], the authors propose a pattern language for design issues of business process-driven service orchestrations. The patterns illustrate how these types of service invocation need to be reflected in process models in order to integrate processes with services. Implications regarding the functional architecture are also captured by the patterns. Specifically, the patterns reflect solutions for general business requirements that can be found in SOA engagements. Overall, the paper proposes a solution, more precisely, a pattern language covering, for example, Synchronous Service Activity, Fire Event Activity, and Asynchronous Sub-process Service.

In subsequent work [HZ09] the authors present solutions to Process-driven SOA patterns in the sense of a process integration architecture featuring patterns at Macro Flow (business process) and Micro Flow level (transaction or human), as well as Integration Adapter, Configurable Dispatcher, and Integration Adapter Repository. These patterns correspond to the ones proposed in [HZ06]. Furthermore long-running business processes are distinguished from short-running technical processes. Zdun et al. present a survey of technology-independent patterns that are relevant for SOA and argue towards formalized pattern-based reference architecture model to describe SOA concepts [ZHVDA06]. Finally, Zdun describes a federation model to control remote objects and proposes a solution based on patterns, e.g., broker and software patterns [Zdu08].

Autili et al. discuss challenges posed by the heterogeneity of Future Internet services [ASPT15]. Modern service-oriented applications automatically compose and dynamically coordinate software services through service choreographies described based on BPMN 2.0 Choreography Diagrams. The authors state that currently composition and adaptation is often a manual task, hence, they advocate towards the automatic synthesis of choreography-based systems and describes preliminary steps towards exploiting Enterprise Integration Patterns to deal with a form of choreography adaptation. Concretely, an adapter generator and prototype using spring integration is presented. Example patterns comprise Message Routing Patterns, namely Message Filter, Aggregator, Splitter, and Resequencer. Overall, this work bridges SOA to EAI using EIPs and protocol adapters for services. Moreover, it is planned to integrate EIPs with security patterns and message transformation as future work.

In Gacitua-Decar and Pahl an ontology-based approach to capture architecture and process patterns is presented [GDP09]. Ontology techniques for pattern definition, extension and composition are developed and their applicability in business process-driven application integration is demonstrated. The proposed solution is an architecture framework for SOA-based EAI as well as an ontology-based notion of patterns to link business processes and service architectures. This could be seen as a formalization approach. A SOA service integration framework with a pattern-based modeling approach is presented by Heller and Allgaier [HA10]. It features controlled extensibility of enterprise systems for unforeseen service integration and can be estimated as similar to related B2B Integration and Enterprise Application Integration. The framework leverages structural or behavioral interface mediation techniques. The modeling approach with adaptation patterns and runtime support is demonstrated with a UI integration prototype in the automotive domain. Overall, this work suggests pattern-based modeling as solution. Kaneshima and Braga analyse whether EAI can be conducted by web services and SOA or DB sharing [KB12]. Both solutions are being adopted by organizations, although they present advantages and disadvantages that should be analyzed. This work documents these problems and solutions in the form of patterns like access via Shared Database, direct RPC-style integration via web services, Intermediate Duplication with access via DB or web services. Hence, the proposed solution is

based on SOA and patterns.

Umapathy and Purao transform EIPs to web service implementations using a transformation model called ceipXML [UP06]. The proposed solution comprises conversation models that may be used to implement interactions among Web services as well as a methodology that generates the design elements in the form of conversation policies for Web services. Current integration approaches do not support the end user development requirements for infrequent, situational or ad-hoc integration and collaboration as stated by Zheng et al. [ZCJ11]. The work differentiates between UI, component, business logic, resource and data integration. Gierds et al. define an approach for behavioral adapters based on domain-specific transformation rules that reflect the elementary operations that adapters can perform; synthesize complex adapters that adhere to these rules [GMW12]. The proposed solution comprises a formalization, specification of the elementary activities to model domain knowledge, separating data from control, and a reduction from adapter synthesis to controller synthesis. An adapter is generated to reconcile mismatches (e.g., incompatible protocols) in Sequel et al. [SEG08]. The proposed solution is constituted by a survey of protocol adapter generation (e.g., semi-automated protocol adapter generation). Gudivada and Nandigam deal with EAI using extensible Web services [GN05]. A solution is not directly proposed, but rather a practical implementation. Deng et al. combines SOA and Web service technology to simplify EAI by studying the service-oriented software analyzing and development characteristics [DYZ⁺08]. The approach distinguishes between vertical integration within an enterprise while B2B emphasize on the horizontal integration. Again the paper presents a more practical implementation.

SOA and Mobile Computing. Mauro et al. [MLK10] target design problems of SOA for mobile devices with Service Oriented Device Architecture (SODA). For this SOA design patterns like Enterprise Inventory are analyzed with respect to their applicability to SODA, and new pattern candidates like Service Virtualization are identified. From these candidates new (device) patterns including Auto-Publishing, Dynamical Adapter, Server Adapter, Integrated Adapter, External Adapter are proposed as solutions.

SOA and Mashups. Liu et al. combine several common architecture integration patterns, namely Pipes-and-Filters, Data Federation, and Model-View-Controller to compose enterprise mashups [LLX⁺09]. Moreover, these patterns are customized for specific mashup needs. In [LLX⁺11] enterprise architecture integration patterns (e.g., Pipes-and-Filter, Data Federation, Model-View-Controller) are leveraged in order to compose reusable mashup components. The authors also present a service oriented architecture that addresses reusability and integration needs for building enterprise mashup applications. The proposed solutions focus on SOA and mashups, but no solution to EIPs and new trends is provided. The work by Braga et al. addresses issues of complexity of service compositions with adequate abstraction to give end users easy-to-use development environments [BCDM08]. Abstract formalisms must be equipped with suitable runtime environments capable of deriving executable service invocation strategies. The solution tends towards mashups and modeling as users declaratively compose services in a drag-and-drop fashion while low-level implementation details are hidden. However, the solution could not be clearly identified and is hence not included in Table 2.1 (on page 31). Finally, Cetin et al. chart a road map for migration of legacy software to pervasive service-oriented computing [CAO⁺07]. Integration takes place even at the presentation layer. No solution is provided for EIPs and trends, however, mashups are used as migration strategy to SOA for the Web 2.0 integration challenge.

SOA Security. Qu et al. present six bilateral patterns (Binding, On-demand, Tailor, Composite, Contract and Migration) and four multilateral patterns (Separated, Shared, Mediated and Enhanced) as a solution for integrating new services with Grid security services [QYZL05]. For each pattern, the authors discuss its intent, applicability, participants and consequences. Shah and

Patel analyze the security requirements for global SOA [SP08]. For security concerns, dynamic configuration of handlers, sequence, and identification of handlers is proposed as solution. Fisher et al. provide practical implementations in Java and .NET for interoperable, synchronous, and asynchronous integration [FSLM06]. Hence, the proposed solution consists of implementation details for SOA, WS security examples, and best practices such as a secure object handler adding custom interceptor logic for, e.g., adding digital signatures.

SOA and Business Processes. Ouyang et al. formalize process control flows into BPEL processes by an intermediate translation to Petri nets [OVVDA⁺07]. From the same group, Wang et al. construct and interface adaptation machine that sits between pairs of services and manipulates the exchanged messages according to a repository of mapping rules. For both approaches, the proposed solution is a formalization. Lohmann et al. analyze the interaction between WS-BPEL processes using Petri nets [LMSW08]. Again the proposed solution is a formalization. With a similar goal, Kumar and Shan aim at simplifying the pattern compatibility based on a matrix and rules that enable the simplification of checking compatibility between two or more processes because these prerequisite rules can be applied to each pattern separately [KS08]. The proposed solution is an algorithm and can hence be subsumed as formalization. Mismatch patterns that capture the possible differences between two service (business) protocols to adapt and automatically generate BPEL adapters are presented by Jiang et al. [JZGH11]. They introduce several dependencies such as transformation dependency (incl. message split), synchronization dependency, choice dependency (choice among two or more messages), and priority dependency. The proposed solution is the formalization of mismatches. Barros et al. propose SOA process interaction patterns including Send, Receive, Send/Receive, and Racing Incoming Messages [BDTH05]. Patterns for synchronization problems in the area of process-driven architectures, e.g., Waiting Activity or Timeout Handler, are introduced by Köllmann and Hentrich [KH06]. Vernadat looks at architectures and methods to build interoperable enterprise systems, advocating a mixed service and process orientation and the classification of integration levels, physical system, application, business integration, and enumerates SOA concepts [Ver07]. No specific solution is proposed. Grossmann et al. derive integration configurations from underlying business processes, e.g., activities [GSS07]. Future work names exception handling as a challenge, and thus no solution provided.

Event-driven Architecture (EDA) and SOA. Taylor et al. address the SOA – EDA connection as service network and provide a reference EDA manual [TYPM09]. As no solution is provided, the approach is not included in Table 2.1 (on page 31). A theoretical framework for modeling events and semantics of event processing is provided by Patri et al. [PSPP14]. The formal approach enables modeling real-world entities and their interrelationships and specifies the process of moving from data streams to event detection to event-based goal planning. Moreover, the model links event detection to states, actions, and roles enabling event notification, filtering, context awareness, and escalation. The proposed solution consists of events and formalization.

Cloud Computing, Business Networks, and Hybrid Applications The successor of grid and cluster computing is cloud computing that extends B2B to business networks. Moreover, the coexistence of applications on-premise and in several kinds of cloud platforms is extended towards hybrid applications.

Cloud Computing. Asmus et al. focus on the migration of enterprise applications to the cloud [AFP16]. Integration is considered a key factor influencing cloud deployment. Several migration patterns are described as a basis for enabling enterprise cloud solutions. The following challenges are named in the paper: data volume, network latency, identity and data security management, interoperability (i.e., supporting the trends big data, security, and variety as in multimedia).

Asmus et al. state that “integration pattern can be a starting point in deciding integration options” [AFP16]. The key areas addressed in the approach include on premise, off-premise private cloud, cloud integration, cloud service provider, and external users. The integration patterns refer to process to process and data integration. Overall, the proposed solutions are “patterns and processes-based” methods for an initial evaluation of the risk and effort required to move new and existing applications to a cloud service. In our recent work, a collection of integration patterns derived from requirements of hybrid and cloud applications is presented [RR15], thus proposing a solution for the cloud and patterns. The main challenge described by Merkel et al. is a secure integration [MSHP15]. The approach proceeds in a top-down manner by deriving integration patterns from scenarios and in a bottom-up fashion by deriving patterns from case study requirements. It identifies the need for security (access control, integrity, confidentiality) as well as security constraints (e.g., EU Data Protection Directive) and presents an evaluation based on an architecture with major focus on hybrid and multi-cloud setups. The described patterns are cross-cloud ESB, usage of ESBs, as well as security patterns as architecture components such as LDAP. The approach only works in private clouds. Merkel et al. propose future work on public clouds that involves content encryption, key management, data splitting, computing with encryption functions, anonymization, data masking, and encrypted virtual machines. They mention Cross-Cloud Balancer, Cross-Cloud Data Distributor, and replication patterns as further future work. Other challenges mentioned are cross-cloud monitoring and cloud management. In summary, the proposed solution are new patterns for SaaS integration and centralized as well as decentralized multi-cloud integration.

Business Network. Our recent work provides mappings of EIPs integration semantics and patterns to BPMN-based models as well as an implementation of a business network scenario example [Rit14a, Rit15c]. Both works do not directly propose a solution to the trends depicted in Figure 2.7 (on page 30), but introduce *modeling* as a possible solution in the context of EIPs, thus added as a category to Table 2.1 (on page 31).

Hybrid Applications. A major challenge in hybrid applications is the decision where to host parts of the application. In this regard, Mansor recommends to bear in mind the patterns in the envisioned process [Man09]. The work addresses technical challenges when implementing a hybrid architecture. The proposed solution refers to architectural patterns. A holistic approach for the development of a service-oriented enterprise architecture with custom and standard software packages is presented by Buckow et al. [BGP+10]. The system architecture to be developed is often based on integration patterns for the physical integration of systems. No solution is provided in the context of this work.

Internet of Things and Big data With affordable and widespread mobile sensors and devices comes the Internet of Things and together with the immense amount of data from cloud and mobile computing comes Big data.

Internet of Things (IoT). Heiss et al. collect challenges in cyber-physical systems such as communication quality, interoperability, and massive amounts of data [HOS+15]. As interesting requirements they state “placement” (of integration scenarios), e.g., cloud or on-device, the demand for global optimization, more intelligent devices, networking and cloud and security including data security and privacy etc., decoupling of layers vs. direct data access for on-top applications. Rather than proposing a solution, the industrial and business perspectives on such envisioned platforms are described.

Big data . Ritter and Bross suggest moving-up relational logic programming for implementing the integration semantics within a standard integration system [RB14]. For this EIP semantics is translated to relational logic. For declarative and more efficient middleware pipeline processing

(e.g., parallel execution, set-operations), the patterns are combined with Datalog. The expressiveness of the approach is discussed, and a practical realization by example is provided. Although no direct solution to the trends is provided the approach directs to “data-aware” EIPs.

General EAI approaches From practical EIP implementations to ideas for new patterns, formalization approaches, enabling techniques and domain-specific work, this section rounds off the literature analysis with further EAI challenges.

Practical Aspects. Scheibler and Leymann present a framework for configuration capabilities of EIPs, specifically for code generation based on a model-driven architecture [SL08]. In [SL05], EIPs are implemented in IBM WebSphere. Again no solution for the trends is provided, but a solution to the EIPs through *implementation*. Thullner et al. analyze EIP coverage in open source tools and implement a sample scenario in Apache Camel and Mule [TSS08]. No solution is provided.

EAI Patterns. [Hoh06] presents a pattern language for conversations between loosely coupled services, i.e., *patterns* are suggested as a solution. Gonzales and Ruggia deal with response time and service saturation issues (more requests than can be handled) using an adaptive ESB infrastructure [GR11]. They propose solutions in the form of *strategies*, i.e., Delayer, Defer Requests, Load Balancing, and Cache.

Formalization and Verification. Fahland and Gierds present a conceptual translation of EIPs into Colored Petri nets, hence providing a formal model based on a system specification using EIPs [FG12, FG13]. The Petri net based formalizations can be used to simulate and conduct model checking of pattern compositions. Though the formalization can be understood as a solution, it does not address any new trends beyond EIPs, thus this approach is not contained in Table 2.1 (on page 31). A semantic representation of EIPs for automatic management of messaging resources (e.g., Channels, Filters, Routers) is presented by Patri et al. [PPSP14]. The application is to connect mobile customers to Smart Power Grid companies. Data is accessed in the form of alerts from a complex event processing engine using SPARQL queries. The proposed solution is a formalization for resource management of integration patterns. Basu and Bultan focus on the interaction behavior in asynchronously communicating systems resulting in decidable verification for a class of these systems [BB14]. As the proposed solution (formalization) is not in the context of the trends, it is not included in Table 2.1. Mederly et al. generate a sequence of processing steps needed to transform input message flow(s) to specified output message flow(s) [MLZN09, MN10]. The work takes into account requirements such as throughput, availability, service monitoring, message ordering, and message content and format conversions. Additionally, it uses a set of conditions, input and output messages, and a set of configuration options. Control flow ordering is formalized. The work is excluded from Table 2.1 because it provides no solution, but rather creates parts of integration solutions from the description of what has to be achieved, not how it should be done.

EAI enabling techniques. The following approaches address different enabling technologies. However, neither are the presented approaches related to the trends, nor do they propose concrete solutions. Hence they are not included in Table 2.1. Architectural patterns (e.g., Remote Process Invocation, Batch Data Synchronization, SOA, Publish-Subscribe, P2P, Broker, Pipes-and-Filters, Canonical Data model, Dynamic Router) are contributed by Kazman et al. [KSNK13]. This work constitutes a guideline for IT architects that combines existing patterns. Land et al. integrate the existing software after restructuring or merger, i.e., address the question of how to carry out the integration process [LCL05]. Multiple case studies and recurring patterns for vision processes and an integration process are provided as well. Basic concepts of enterprise architectures including integration and interoperability are summarized by Chen et al. [CDV08].

Domain-specific Approaches. Cranefield and Ranathunga integrate agents with a variety of external resources and services using Apache Camel and the EIP endpoint concept [CR13]. e-Learning as a growing and expanding area with huge number of disparate applications and services is addressed by Rajam et al. [RCVB10]. The approach redefines the Model-View-Controller pattern. It can be further enriched to encapsulate certain non-functional and integration activities such as security, reliability, scalability, and routing of request. As all these approaches do not propose a solution directly connected to EIPs and the trends, they are not included in Table 2.1 (on page 31).

EAI Challenges. A survey to motivate some more challenges in the area of enterprise application integration and to link back to the trends is presented by He and Xu [HDX14]. Further this work examines the architectures and technologies for integrating distributed enterprise applications, illustrates their strengths and weaknesses, and identifies research trends and opportunities for horizontal and vertical integration. Though no solution is proposed, the discovered trends are strengthened, for example, SOA, personal, mobile, and IoT. The survey also addresses NFAs, e.g., security, which are collected and serve as input for Table 2.1. Another survey by Panetto et al. discusses trends and NFAs in enterprise integration [PJGM12]. Moreover, *modeling* and *formalization* (formal methods such as verification) are proposed as challenges, but no concrete solution provided.

Synthesis and Discussion of Non-functional Aspects

The second aspect of our analysis of trends are topics that were named by Gartner [Gar17] and Zimmermann et al. [ZPHW16] as relevant or that were identified during the literature review. However, these topics have a more cross-cutting quality (i.e., relevant for several trends). We call them NFAs, which we appended to Table 2.1 together with the references that supported them as challenges as evidence. They are set into context to important aspects, when working with integration scenarios, namely patterns, formalization and modeling. The focus on patterns comes from the EIPs [HW04] and supported by many related domains, that capture knowledge and best practices in the form of patterns (e.g., SOA, Cloud Computing). Panetto et al. [PJGM12] bring up the formalization (supported by [ZPHW16]) and modeling (supported by [Rit14a, Rit15c]) as additional relevant topics. We now set these topics into context with the references from the literature analysis in Table 2.1.

For the EIPs, we added asynchronous message processing as *Asynch* to cover the solutions in this space, e.g., by [HW04, GR11]. For the NFAs, solutions in the area of formalizations are proposed by [FG12, FG13] for the validation of pattern composition and business processes. Another NFA is *Security*, which was seen as challenge at least by Gartner [Gar17] and in the literature by [SP08, HOS⁺15, RCVB10] (in general), by [HDX14] (performance concerns, real-time integration), and by [MSHP15] (e.g., safe integration, indications that content encryption, key management and more is missing). Autili et al. [ASPT15] mention the need for security integration patterns. The solutions for patterns are limited to SOA with patterns like Secure Service Consumption or Security Handler Information Exchange [QYZL05, SP08].

According to Gartner, multimedia format handling and processing can be seen as a non-functional requirement [Gar17]. This includes image, video and text image formats, which are increasingly produced through mobile devices and, e.g., interacted on social media, becoming of increasing interest for (business) applications.

In the context of the big data challenges of integration systems (from Gartner; e.g., volume, velocity, stability), (synchronous) streaming protocols are seen as one possible solution. The authors of [ZPHW16] mention that patterns as well protocols are currently missing in EAI.

With more and more communication partners that result from the trends in Section 1.2.2,

(stateful) conversational protocols might be required, according to [ZPHW16] and also Gartner (e.g., device meshes). First ideas have been sketched by [Hoh06] with an initial collection of conversation patterns, which should be extended [ZPHW16]. For SOA web service conversation policies [UP06] and interaction patterns [BDTH05] solutions were provided. Formalizations have been proposed in [LMSW08] for the SOA domain with focus on the controllability of a process. The proposed solutions for SOA might be transferred to integration processes as starting point for more general conversation patterns.

To handle erroneous situations during message processing, escalate them and make systems more fault-tolerant, error handling is seen as a major aspect [ZPHW16, MSHP15]. Hohpe et al. [HW04, Hoh05] do only cover Dead Letter Channel as solution and sketch some ideas about the topic. Overall, in the literature, the topic is neither addressed from a pattern, formalization, nor modeling perspective. While [ZPHW16] mentions missing patterns and formalization, Merkel et al. [MSHP15] lists Balancing and Distribution, as well as [HDX14] mentions Fault-tolerance and Message Scheduling as missing aspects. Similarly, the insight into the current state of affairs, called monitoring, for services and cross-cloud are seen as important topics in [MSHP15, MLZN09, MN10].

The monitoring of integration processes as well as cross platform monitoring were only mentioned, however, no solution was provided. The Control Bus, a Wire Tap and the Message History patterns in Hohpe et al. [HW04] denote partial solutions, which can be used to build a monitoring solution on integration process level.

2.2.3 Inductive Analysis: System Review

This section reports on the results of a system review to evaluate **hypotheses H2** *Current system implementations support patterns beyond EIPs*, and **H3** *Some trends are handled in an (yet) immature and ad-hoc fashion* as set out in Figure 2.6 (on page 28). The system review is based on the guidelines described in [Kit04] for a horizontal search including “well-established” commercial application integration systems, more experimental systems from startups, open source systems and public knowledge in the form of a Wikipedia search. The selection of systems was conducted on 2016-10-04, and the results of the horizontal search are summarized in Table 2.2. The NFAs are used to focus the search in those systems.

First, out of 12 systems, seven commercial systems were collected by taking the systems listed in both the Gartner (Leaders, Visionaries, Challengers) [Gar16] and the Forrester (Application Integration) IPaaS list [For16b], leading to the following systems: Dell Boomi [DEL17], IBM Cast Iron [IBM17], Informatica [Inf17], Jitterbit Harmony Cloud Integration [Jit17], Microsoft BizTalk [Mic17], SAP Cloud Platform Integration [SAP19a], and Oracle Cloud Integration [Ora17]. We excluded MuleSoft due to its similarity to Apache Camel [IA10], which we selected as an expert addition from Wikipedia (discussed later).

In addition, two startup systems from the top 20 overall systems were selected due to their number of followers on angel.co⁴, namely Tray.io [Tra17] and Zapier [Zap17]. While the former is striving to build an “Integration Marketplace” for enterprise applications, Zapier is a cloud integration startup.

Out of 13 open source systems of the Github Hadoop Ecosystem⁵, we selected Apache Flume [Apa17a] and Nifi [Apa17b] as data ingestion systems according to the selection criteria (cf Table 2.2). We excluded the application integration systems Talend (also listed as a commercial system), Spring Integration and MuleESB for their similarity to Apache Camel as well as Apache Beam, Apache Sqoop and Spring XD for their similarity to Apache Flume.

⁴Angel.co, visited 02/2017: <https://angel.co/data-integration>

⁵Hadoop Ecosystem on Github, visited 02/2017: <https://hadoopecosystemtable.github.io/>

Table 2.2: System review — horizontal search

Category	hits	selected	Selection criteria	Selected Systems
Commercial	12	7	Gartner and Forrester IPaaS Quadrants	Dell Boomi [DEL17], IBM Cast Iron [IBM17], Informatica [Inf17], Jitterbit [Jit17], MS BizTalk [Mic17], SAP Cloud Platform Integration [SAP19a], Oracle [Ora17]
Startup	20	2	cloud/data integration, B2B, API, #followers	Tray.io [Tra17], Zapier [Zap17]
Open Source	13	2	application integration, data ingestion	Apache Flume [Apa17a], Apache Nifi [Apa17b]
Wikipedia	34	1	enterprise application integration; non-duplicates	Apache Camel [IA10]
Added Sys-tems	n/a	3	expert knowledge	Cloudpipes [Clo17] (startup), Tibco [Tib17], WebMethods [Sof17] (commercial)
Removed Sys-tems	-	-		
Overall	74	15		

The open source integration system Apache Camel [IA10] does not appear in the open source list, however, it was the only non-duplicate from the other lists that has to be selected, since it implements the existing EIPs from [Hoh05] and is a role model for many systems like Spring Integration, or Red Hat’s FuseESB.

The software systems of Tibco [Tib17] and Software AG [Sof17] are wide-spread and influential integration systems for on-premise with a cloud integration offering and are listed among the top for wide integration and deep integration for traditional on-premise by Forrester⁶. Hence we add them as expert selected additions. We add Cloudpipes [Clo17] from the startup list as cloud integration system.

That leaves us in total with 15 systems with a good mix of well-established commercial and startup products, as well as community projects. Since the main focus lies on commercial systems that are known to be less well accessible for a systematic analysis of their features, we focus on the publicly available material (i.e., without registration or login) and try to get more information from underlying open source systems, where possible.

Processing of Selected Systems

EIP Solutions used in System Implementations We start our system review with an analysis of all selected systems with respect to their implementation of EIP solutions. The EIPs describe six pattern categories, namely, Messaging Channels, Message Construction, Message Routing, Message Transformation, Messaging Endpoints and System Management. We focus the analysis on the two pattern categories of message routing and transformation, since they represent the core aspects of integration systems. Furthermore we leave out composed patterns (e.g., composed message processor, scatter-gather), when their single parts are already in the selection. Table 2.3 (from Boomi to Flume) and Table 2.4 (from Nifi to Webmethods) depict the solutions found in the system implementations that could be associated to the routing and

⁶The Forrester Wave: Hybrid2Integration, Q1 2014

Table 2.3: Original EIPs used in systems (Boomi to Apache Flume)

Pattern	Boomi	IBM	Informatica	Jitterbit	BizTalk	SAP	Oracle	Flume
Content-based Router	✓	✓	-	-	✓	✓	✓	(✓)
Message Filter	-	✓	-	-	✓	-	-	✓
Dynamic Router	-	-	-	-	✓	-	-	(✓)
Recipient List	-	-	-	-	-	-	-	-
Splitter	✓	✓	(✓)	✓	✓	✓	✓	✓
Resequencer	-	-	-	-	-	-	✓	-
Routing Slip	-	-	-	-	-	-	-	-
Aggregator	-	-	(✓)	-	✓	✓	(✓)	(✓)
Envelope Wrapper	(✓)	(✓)	(✓)	(✓)	-	-	-	-
Content Enricher	(✓)	-	-	(✓)	(✓)	✓	-	(✓)
Content Filter	(✓)	-	-	(✓)	(✓)	✓	-	(✓)
Claim Check	(✓)	-	-	(✓)	-	(✓)	-	-
Normalizer	(✓)	-	(✓)	✓	(✓)	(✓)	-	(✓)
Message Translator	✓	-	(✓)	✓	✓	✓	-	(✓)

supported ✓, partial (✓), unknown/not supported -.

transformation patterns.

The Apache Camel system seems to be specifically designed around the EIPs, thus supports nearly all EIPs and sticks to the original EIPs naming for the respective solutions, which makes it a benchmark for the others. Most notable deviations are the Envelope Wrapper (i.e., wrap application data inside an envelope, compliant with the messaging infrastructure) and Message Translator patterns (i.e., translate one data format into another one; not in transformation patterns). None of them is directly represented in Camel, however, can be implemented using UDFs (i.e., user-defined functions like Camel Processor) or scripting (e.g., Camel Script), therefore marked as partially covered.

The most common routing pattern solutions are the Content-based Router, the Splitter and the Aggregator. Since the Message Filter is a special case of the Content-based Router and filter can be used to construct the latter, not all systems provide implementations for both of them. The Splitter is sometimes implemented according to the description in the EIPs, however, some vendors decomposed it to its iterative core functionality (e.g., For Each in IBM, Oracle, Cloudpipes). The Aggregator shows many partial solutions that require user-defined functions (e.g., Informatica, Oracle, Tray.io), while only few provide its EIP functionality (e.g., Aggregator in BizTalk, SAP Cloud Platform Integration, Tibco or ContentMerge in Apache Nifi).

The transformation patterns seem to play a major role in the analyzed systems, since most of them are broadly supported. However, there seems to be a tendency to provide UDF capabilities and leave the burden to the user to deal with the semantics.

Finally, the dynamic routing patterns (e.g., Dynamic Router), those patterns that contain the recipient in their content (e.g., Recipient List, Routing Slip), and the Message Resequencer, e.g., used for the exactly-once-in-order service quality [RH15], were sparsely implemented. This leaves the question on their relevance or other components that take over their function.

Summary While some of the EIPs like Content-based Routing or Message Filter, Splitter and Content Enricher can be found in most of the systems, others are rarely implemented (e.g., Resequencer, Routing Slip, Dynamic Router). The analysis of these patterns and their relevance are left for future work, and thus not analyzed further.

Table 2.4: Original EIPs used in systems (Apache Nifi to Webmethods)

Pattern	Nifi	Camel	Tray.io	Zapier	Cloudpipes	Tibco	Webmethods
Content-based Router	✓	✓	✓	-	✓	✓	✓
Message Filter	-	✓	-	✓	✓	-	-
Dynamic Router	-	✓	-	-	-	-	-
Recipient List	✓	✓	-	-	-	-	-
Splitter	✓	✓	-	✓	✓	✓	-
Resequencer	-	✓	-	-	-	-	-
Routing Slip	-	✓	-	-	-	-	-
Aggregator	✓	✓	(✓)	-	-	✓	-
Envelope Wrapper	-	(✓)	-	-	-	-	-
Content Enricher	✓	✓	✓	(✓)	(✓)	✓	✓
Content Filter	✓	✓	(✓)	(✓)	(✓)	✓	✓
Claim Check	-	✓	-	-	-	(✓)	-
Normalizer	-	✓	(✓)	(✓)	(✓)	✓	✓
Message Translator	✓	(✓)	✓	(✓)	(✓)	✓	✓

supported ✓, partial (✓), unknown/not supported -.

New Solutions not covered by System Implementations We now analyze the collected systems with respect to their functionalities according to the harvested NFAs from Section 2.2.2, namely security, media, streaming or more abstract “processing”, conversations, error handling, and monitoring. Comparing the NFAs with the collected system functionalities, while neglecting functionalities covered by the EIPs, gives an answer to the question which topics are required and used in addition. The system functionalities represent an implemented solution as part of an actual integration system.

Figure 2.8 depicts found solutions not covered by the EIPs by NFAs and system vendor. During the analysis new NFAs were identified — not mentioned by Gartner, the EIP authors, or the literature — that seem to play a role in practical terms: stateful integration processes using storage, (pattern) composition (mentioned in Zimmermann et al. [ZPHW16]), and system operations. These three new NFAs were included into the analysis of the other systems as well. All non-related topics are collected as miscellaneous (Misc).

Notably, all identified NFAs are at least partially covered by system implementations, indicating that solutions in the form of conceptual definitions are required (e.g., as patterns). According to Mulesoft [Mul19], the major challenges in cloud integration systems are security and management. The management includes error handling and monitoring, which allow to control the behaviour of the integration scenarios.

The classic application integration addresses the variety problem for textual message formats [Lin00]. With the availability of integration systems for “everybody” (e.g., in the form of a cloud integration system) non-textual formats gain importance.

The trade-off between stateful and stateless message processing is represented by storage capabilities in integration systems, for which nearly all vendors propose a solution and conversations. The stateful approaches could be represented by conversational protocols, which allow to move the state from the integration systems to the communication partners (idea sketched in [Hoh06]). Most of the service qualities (e.g., at-least-once, exactly-once processing) [HW04, RH15] require stateful integration processes. Consequently, this would require changes in the applications. Current systems provide only rudimentary support, if at all.

Finally, a broad variety of miscellaneous topics was collected, e.g., sentiment analysis, natural language translators, but also more general functions like sort, loops, as well as explicit format handling, i.e., marshalling and type conversion.

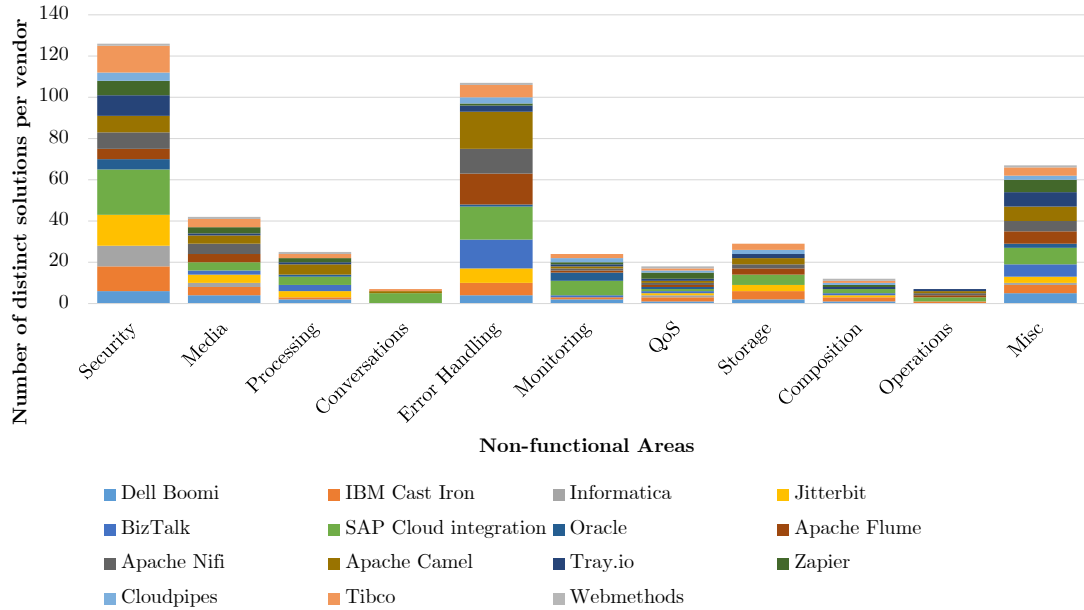


Figure 2.8: Solutions for NFAs not covered by the EIPs by system vendor.

Summary Notably, security and error handling (and monitoring) capabilities are predominantly found. They address the challenges of security and management. Furthermore, solutions for the increasing variety of message formats (cf. multimedia) as well as the volume and velocity handling can be found in the systems are part of new processing types. The storage of data and message semantics like quality of service are relevant for integration scenarios. This leads to the trade-off between stateful vs. stateless integration processes, which briefly address in [Section 2.3.2](#). The (stateful) conversations, which could be part of a solution, are currently under-represented.

System Summaries along NFAs

Security The aspect of confidentiality or message privacy is solved on transport, message and storage levels. The transport level channel encryption can mostly be specified in the inbound and outbound adapters in the form of the transport protocol (e.g., HTTPS, SFTP) and guarantees that the message cannot be read during transmission (e.g., Jitterbit’s Transmission Protection). Once the message is received by the inbound adapter and handed to the subsequent operation in the integration process, message privacy can be applied or reversed. Therefore many vendors provide explicit message encryptors and decryptors (e.g., PGP Encrypt and Decrypt from Dell Boomi, AES_ENCRYPT from Informatica or Encrypt / Decrypt in Apache Nifi), or encrypting adapters (e.g., FileProcessorConnector in Informatica, FileChannel in Apache Flume, WSSProvider in Tibco). The encrypted storage of messages helps to protect the message’s privacy in the store, e.g., can be configured in SAP’s DBStorage and Persist operations. The configuration of the message privacy solutions mostly include encryption algorithms, key lengths and certificates. Similarly, the integrity and authenticity of a message can be ensured on the different levels. Most of the vendors provide configurations for safe and authenticated transport (e.g., using user and password, certificate or token-based authentication). The transport is considered safe if changes of the message can be recognized by the receiver and the authenticity guarantees that

the sender is the expected one. For instance, most of social media endpoints like Twitter and Facebook use token-based OAuth authentication. In addition, many vendors provide explicit message signers and signature verifiers (e.g., Digest / Hash function in IBM, Signer and Verifier in SAP Cloud Platform Integration) as well as safe message storage, e.g., by Jitterbit or SAP Cloud Platform Integration. For the storage, the authenticity seems to be implied, since the cloud platform message or data store is used. The availability of integration scenarios is not only a stability, but also a security concern. Therefore some vendors like IBM, SAP Cloud Platform Integration provide implicit countermeasures, e.g., redundant message stores with high availability and disaster recovery, as well as Apache Flume with explicit MorphlineSolrSinks and Kafka Channel configurations. Finally, changes to the message are tracked for auditing purposes. This is made explicit as Audit Trails in Jitterbit and Oracle or Service Auditing in WebMethods.

Media The literature review in [Table 2.1](#) (on [page 31](#)) shows that there are no solutions for multimedia processing in application integration or related domains (e.g., SOA, EDA). The system analysis does only provide few, superficial solutions. For instance, textual representation of binary content is explicitly configurable in most of the systems (e.g., Base64 Encode / Decode in Dell Boomi and IBM, Encoder / Decoder in SAP Cloud Platform Integration). These encodings play a major role when communicating with remote applications, but also when calling services (e.g., user-defined operations) using textual message protocols. In addition, most of the vendors allow user-defined operations in the form of scripting capabilities (e.g., Script, Processor in Apache Camel, Expressions in Informatica). With that, more complex operations can be performed like the compression of — usually bigger — multimedia messages. Despite that, pre-defined compression operators can be found in, e.g., Dell Boomi, Jitterbit, Apache Nifi, which allow to configure the compression type (e.g., zip). The explicit support of scripting seems to be a general trend, when representing transformation patterns (see [Section 2.2.3](#)). This could either mean that the implementations are too diverse to formulate a general solution or indicate that the topic was not considered yet. The support of explicit image processing operations seems to be limited to Nifi’s `ResizeImage` and `ExtractImageMetadata` functions as well as IBM’s `Read MIME` activity. The only real multimedia operation is the image resizing, since the metadata simply provide a format specific capture of the defined image’s meta tags.

Processing While the literature review does not show solutions for message processing, especially not for “data-aware” or Big Data processing, the systems implement solutions. The canonical solution for processing larger amounts of data is to scale-out to multiple processing units, constituting parallel subprocesses. The parallel processing of one message in subprocesses using a broadcast can be done, e.g., in BizTalk with Create concurrent flows, SAP Cloud Platform Integration Gateways, or Apache Camel Multicast. Furthermore, the explicit configuration of parallel processing within an integration process (i.e., not process parallelization) is supported by, e.g., Dell Boomi using the Flow Control properties, Jitterbit Parallel Processing, Tibco Non-inline subprocesses and Critical Section as BizTalk Scope batch property. Alternatively, a more data-centric approach, however, impacting the latency of the process, is micro-batching [[Rit15b](#)]. Vendors like Dell Boomi and Jitterbit also support batch processing of messages using the Flow Control properties or Chunking configurations. The processing of message streams allows the system to handle larger amounts of data than the integration system resources would allow. This more connection oriented approach was identified by Zimmermann et al. [[ZPHW16](#)] as a missing pattern category in the context of synchronous message processing. An explicit streaming support is provided, e.g., by Jitterbit Streaming Transformation and Apache Camel. However, not all integration operations or adapters are (conceptually) capable of streaming.

Conversations Gartner [Gar17] as well as Zimmermann et al. [ZPHW16] mention the importance of conversations for messaging. These conversations are similar, however, stand in contrast to the choreography (e.g., [Pel03, ASPT15]) and interaction patterns for services [BDTH05] in SOA because they denote more complex tasks than sending and receiving data or messages. They target complex (stateful) conversations as partially covered in [Hoh06]. Some of the systems allow a timed redelivery of messages in a non-error case (e.g., SAP Cloud Platform Integration, Apache Camel). This feature is similar to the Contingent Requests pattern in [BDTH05]. For a conversation, an acknowledgement mechanism would be required similar to [Hoh06]. One technique of reducing the number of requests to an endpoint is request caching. In Tibco, request caching can be configured by specifying time slices and operations in the Caching Stage. The SAP Cloud Platform Integration system allows to map synchronous to asynchronous communications and vice versa. This becomes necessary when the endpoints’ message exchange mechanisms do not fit.

Error Handling Error handling is a crucial aspect of integration scenarios [ZPHW16] for the control and fault tolerance aspects. In the literature we found solution attempts [HW04, Hoh05] like the Dead Letter Channel pattern for the collection of failing messages, while the systems implement various, more sophisticated solutions. The fundamental topic for dealing with errors in integration scenarios is the handling of exceptions. Therefore, most of the systems provide a “catch-all” capability (e.g., Catch All in IBM), which sometimes even come with an exception subprocess for more advanced handling (e.g., Exception Subprocess in SAP Cloud Platform Integration). In addition, vendors like Dell Boomi, IBM, SAP Cloud Platform Integration and Tibco provide more fine-granular scoping of exception handlers, e.g., down to the single operation. More advanced topics include escalation, fault-tolerance and eventually prevention techniques. Most notably, the systems support escalation mechanism like (partial) abortion of complex processes (e.g., incl. parallel processing) and raising indicators for alerting, as well as message redelivery on exception for tolerance, and message validation, load balancing (see [GR11]) and flow control to prevent errors. More recent work [RS14, RS16] — not found in the literature review — covers all of the found system solutions as patterns and shows their composition. Furthermore, it introduces the concept of compensations (e.g., for undo operations), which was not found within the reviewed system implementations.

Monitoring The monitoring of integration scenarios gains importance especially within integration platforms hosted by a third party and across those platforms in cloud and mobile computing. The major monitoring aspects found in the systems can be distinguished into UI components that show important aspects of the system, called monitors, and a rather event-based registration on the instance level. For example, Dell Boomi supports message change events by Find Changes, which can be extended to Field Tracking. Oracle supports the latter with Business Identifiers. That means, user-defined events on technical and business level can be tracked via conditional events. Examples for monitors can be found in most of the systems across all parts of an integration scenario (i.e., from adapter or channel, over component, down to message monitors). The monitors can be fed by built-in and user-defined message interceptors (e.g., in Apache Flume and Camel), which allow scenario specific monitoring. When integrating hybrid applications, most systems provide central, cloud monitors instead of local ones.

Storage An integration system requires persistent stores and queues to be operable, e.g., for System Management and monitoring. In addition, message delivery semantics (e.g., reliable messaging like “exactly-once-in-order”) [RH15], secure messaging, and legal aspects (e.g., “Which messages were received and processed?”) must be ensured. In the literature only simple

messaging related storage are mentioned like the *Message Store* [HW04], for storing messages during processing, and the *Claim Check* [HW04] to store (parts of) the message during processing and re-claim them later. Consequently, several system vendors identified the need for additional storage capabilities, summarized as data stores and their access (e.g., DB Update in Jitterbit, DBStorage in SAP Cloud Platform Integration), as well as memoization and caching during one instance of a scenario or between them (e.g., Add to Cache in Dell Boomi, Shared Variables in Tibco, Global Variables in Jitterbit). For secure messaging, security related storage like the *Key Store* (e.g., in Apache Flume, Camel and SAP Cloud Platform Integration), for storing certificates and secure key material, and the *Secure Store* (e.g., in SAP Cloud Platform Integration), for storing secure tokens, users, and passwords, can be found.

Composition In Zimmermann et al. [ZPHW16] the composition of EIPs is mentioned as one of the missing pieces. Many system vendors, e.g., Dell Boomi, IBM, BizTalk, SAP Cloud Platform Integration, allow subprocess modeling as well as delegation from the main integration process. One important solution are integration process templates, which are configurable re-use processes. Many of the vendors support them, however, under different names, e.g., Template integration process in IBM, Snapshot of Jitterpack in Jitterbit, Blueprint in Cloudpipes.

Miscellaneous The most notable, specific features are explicit or implicit loops, keyword search and replace as well as content sort and format handling. The implicit loop configurations include While Loop activity, e.g., in IBM, Looping in BizTalk, and the Loop Collection connector in Tray.io. Explicit loops are possible in most of the systems by back-references in the process. Dedicated search and replace functionality is provided, e.g., in Dell Boomi, Jitterbit, and Apache Nifi. While most type converters are implicit in most systems, marshalling support is made explicit, e.g., in Jitterbit, SAP Cloud Platform Integration, and Apache Nifi. More “exotic” functions are text sentiment analysis in Cloudpipes, an Archiving activity in IBM, and a Yandex language translator in Apache Nifi.

2.3 Pattern Authoring and a Pattern Catalog

This section considers the information collected in the pattern identification phase, defines an extended pattern format based on the one used for the EIPs and accordingly structures the information as pattern descriptions. Since the new patterns reference existing ones and build on each other, they extend the existing EIP pattern catalog. During the execution of the process we loosely followed the best practices on writing pattern documents by Wellhausen and Fießer [WF12] and Meszaros and Doble [MD97].

2.3.1 Design of a Pattern Catalog

This section summarizes the findings of the literature and system reviews in the form of a pattern catalog, capturing and describing the found ad-hoc solutions and functionalities as new patterns. These patterns can be seen as the starting point of a continuation of the EIPs, but also recent trends to express domain knowledge as patterns [FLR⁺14]. In doing so, **hypothesis H3** “Some trends are handled in an (yet) immature and ad-hoc fashion” is targeted. The design goals for the pattern catalog are:

1. Comprehensiveness, i.e., coverage of system implementations that are not in the literature
2. Novelty, i.e., literature coverage of the missing or only partial pattern definitions for NFAs

The proposed pattern catalog is summarized in [Tables 2.5 and 2.6](#), categorizing the patterns by NFAs as *Category*. The patterns in column *Pattern Name* are further grouped by sub-categories as *Scope*. The descriptions of all patterns contained in the catalog are provided in [\[RR15\]](#) and two of them are introduced in detail in [Section 2.3.2](#).

Goal 1 (Comprehensiveness): System Implementation Coverage

In detail, comprehensiveness is evaluated by comparing the coverage of patterns with the NFAs that are not or only partly covered by patterns in the literature, represented by the *Scope* in relation to a given *Category*. The coverage of system implementations reflected by column *System Examples* was chosen in order to provide pattern definitions referring to examples (but not all) of the corresponding system implementations (if at least one vendor supported them). Subsequently, we refer only to the categories that have relevant for the comprehensiveness of our analysis.

Security Take, for example, NFA *Security* in relation to scope *Confidentiality* (see [Table 2.5](#)), for which no comprehensive pattern is provided in the literature on the one side, but system implementations by, for example, Dell Boomi [\[DEL17\]](#), Informatica [\[Inf17\]](#), or Apache Nifi [\[Apa17b\]](#) exist. Addressing design goals 1) and 2) led to the set of suggested patterns *Message Encryptor*, *Message Decryptor* and *Encrypting Endpoint*. *Message Encryptor*, for example, covers the system implementations *PGP Encrypt / Decrypt*, *AES_ENCRYPT*, and *Encrypt / Decrypt*.

Media Besides formatting patterns for structured message content, the media specific patterns for unstructured content are under-represented in current system implementations, since there is only one pattern with direct relation to multimedia processing (e.g., *Image Resizer* [\[Apa17b\]](#)). Although there are functionalities for the work on the structured multimedia metadata (e.g., *Metadata Extractor*), further research should target the unstructured multimedia data and processing (e.g., in the context of synchronous, streaming protocols).

Summary – Comprehensiveness With the pattern catalog we address 94.74% of the NFA scopes or subcategories (i.e., all but 1 out of 19) derived from system implementations. A synchronous / streaming pattern elicitation — as also mentioned by [\[ZPHW16\]](#) — was not conducted in the context of this work, since the system review did not lead to pattern changes or new patterns, but only adds an additional processing style. However, we consider this an interesting topic especially in the context of the Media and Big Data trends, and propose a separate study for this current gap in the future.

Goal 2 (Novelty): Literature Coverage

Now, we set the pattern findings from the literature review for the NFAs — summarized in [Table 2.1](#) — into context with the new pattern proposals derived from the system analysis. We exclude the solutions from the original EIPs [\[HW04\]](#), and Media, Synchronous / Streaming and Composition (not in NFAs, however, came up during system analysis and mentioned in [\[ZPHW16\]](#)), for which no solutions were found in the literature. In addition, we excluded Error Handling, since it is comprehensively covered from a pattern perspective and compared to system implementations in prior work [\[RS14, RS16\]](#) (not found in the literature review).

Security Although some security patterns were proposed in the SOA domain [\[QYZL05, SP08\]](#), they only provide partial solutions with respect to the NFAs and no solution in the context of the system review.

Table 2.5: New integration patterns for NFAs in the context of system implementations from *security* to *processing* without pattern solutions already covered by literature

Category	Scope	Pattern Name	System Examples
Security	Confidentiality, Privacy	Message Encryptor, Message Decryptor, Encrypted Message	PGP Encrypt / Decrypt [DEL17], AES_ENCRYPT [Inf17], Encrypt / Decrypt [Apa17b]
		Encrypting Endpoint / Adapter	FileProcessorConnector [Inf17], FileChannel [Apa17a], WSSProvider [Tib17]
	Authenticity, Identity	Message Signer, Signature Verifier, Signed / Verified	Digest/Hash [IBM17], Signer, Verifier [SAP19a]
	Storage	Message Encrypted / Encrypting Store	DBStorage, Persist [SAP19a]
		Safe Store	Most of the vendors
Media	Format	Redundant Store	MorphlineSolrSinks [Apa17a], implicit [IBM17, SAP19a]
		Encrypted Channel	Transmission Protection [Jit17]
		Safe, Authenticated Channel	Password, certificate, token-based (Most of the vendors)
		Audit Log	Audit Trails [Jit17, Ora17], Service Auditing [Sof17]
		Type Converter	Type Converter [IA10, Tra17, Zap17]
	Unstructured	Encoder, Decoder	Base64 Encode / Decode [DEL17, IBM17], Encoder / Decoder [SAP19a]
		Marshaller, Unmarshaller	“Data Format” [IA10], “ConvertJSON-ToSQL” [Apa17b], “JsonXMLConverter” [SAP19a]
		Compress Content, Decompress Content	implicit [DEL17, Jit17], Compress Content [Apa17b]
		Custom Script	Script, Processor [IA10], Expression [Inf17]
		Metadata Extractor	Read MIME activity [IBM17], ExtractImageMetadata, ExtractMediaMetadata [Apa17b]
Processing	Synchronous / Streaming	Image Resizer	Image Resizer [Apa17b]
		-	Streaming transformations [Jit17], partially [SAP19a], streaming [IA10]
	Parallel	Parallel Multicast, Sequential Multicast	[Mic17, Sof17, SAP19a]
		Join Router	implicit [IA10], join [SAP19a]
		Delegate	Process Call [SAP19a], Direct-VM [IA10]
Other	Other	Loop	Loop Activity [IBM17], Looping [Mic17]
		Find and Replace	Search/Replace [DEL17], Control Character Replacer [Jit17], Scan Content [Apa17b]
		Content Sort	Sort [IA10]

Table 2.6: New integration patterns for NFAs in the context of system implementations from *conversations* to *composition* without pattern solutions already covered by literature

Category	Scope	Pattern Name	System Examples
Conversations	Endpoint	Commutative Receiver	-
		Timed Redelivery until Acknowledge	-
		Timeout synchronous request	-
	Fault tolerant	Failover Request Handler	Failover Client [Apa17a]
	Resources	Request Collapsing	-
		Request Partitioning	-
Monitoring	Processing	Message Cancellation	[Mic17, Apa17b]
		Usage Statistics	[SAP19a, Ora17]
	Immediate Insights	Raise Indicator	[DEL17, Jit17, Mic17, SAP19a]
		Detect	[Mic17, Apa17b]
	Monitors	Message Interceptor	[Apa17a, IA10], implicit [SAP19a]
		Component Monitor	[SAP19a, Clo17]
		Channel Monitor	[SAP19a, Ora17, Zap17, Clo17, Tib17]
		Message Monitor	[SAP19a, Ora17, Tra17]
		Resource Monitor	[SAP19a, Tib17]
		Circuit Breaker	[IA10]
		Hybrid Monitor	[SAP19a]
Storage	Data, Variable	Data Store	[IBM17, Jit17, SAP19a]
		Store Accessor	DB Update [Jit17], DBStorage [SAP19a]
		Transient Store	Add to Cache [DEL17], Shared Variables [Tib17], Global Variables [Jit17]
	Security	Key Store, Trust Store, Secure Store	[Apa17a, IA10, SAP19a]
Composition		Integration Subprocess	[DEL17, IBM17, Mic17, SAP19a]
		Integration	Template Integration Process [IBM17],
		Subprocess	Snapshot [Jit17], Blueprint [Clo17]
		Template	

Conversations, Processing In terms of conversation patterns, the system implementations only showed basic support (see Table 2.6), however, some more can be found in the literature, showing that this is an area for integration systems to add more features. Although, Barros et al. [BDTH05] mostly reiterate the original EIPs, there are few patterns that are new in the context of the system implementations. In the category of multi-lateral communication, the One-from-many pattern [BDTH05] is a special case of our more general Join Router that we found in the system implementations (e.g., Apache Camel, SAP Cloud Platform Integration). The One-to-many send pattern [BDTH05] is similar to the (parallel) Multicast, found in the systems (e.g., Apache Camel, SAP Cloud Platform Integration), however, some systems have variants that we captured as Sequential Multicast, which routes messages of the same type to multiple receivers in sequence to guarantee the successful delivery to all recipients.

Summary – Novelty From the functionality required by system implementations, 59 distinct, new patterns are derived that were not found in the analyzed literature. However, for 5 out of 7 NFAs (compare to Table 2.1), the literature indicates missing patterns as a research challenge (see Section 2.2.2), thus supporting the extension of the integration pattern catalog for security ([ASPT15, SP08, MSHP15, HOS⁺15, RCVB10, HDX14]), multimedia ([Gar17]), synchronous / streaming ([ZPHW16]), conversations ([ZPHW16], [Gar17]), monitoring ([MSHP15, MLZN09, MN10]), and pattern composition (from system review Section 2.2.3, [ZPHW16]). In addition, the system review raises a demand for storage patterns that was not mentioned in the literature.

Solutions for Future Challenges

We propose several new conversation patterns, of which none was found in the system implementations. The proposed endpoint-specific patterns Commutative Receiver and Timed Redelivery Until Acknowledgment (similar to the Contingent Requests pattern in [BDTH05, Hoh06]) — that together denote a solution for a critical trade-off for scalability inspired by [FLR⁺14] — are discussed in detail in Section 2.3.2. The other patterns are further discussed in [RR15], and target the additional conversation scopes: Fault tolerance and Resources. The multi-tenant processing, conversation patterns (e.g., Cross Scenario and Cross Tenant), patterns that are mostly required in hybrid and cloud computing setups, are already covered by prior work [RH15], thus not shown.

Toward a more stable system, the Timeout Synchronous Request and Failover Request Handler patterns are improving the fault-tolerance of the messaging. Especially in the Big Data context, the resources of an integration scenario or platform become crucial for their stability. Therefore, the Request Collapsing pattern reduces the number of requests within a conversation. In addition, the Request Caching reduces the amount of duplicate requests to the same endpoint, while Request Partitioning optimizes requests to endpoints and confines errors to one request aspect. Together with other patterns from the literature review and the proposed patterns in this work, we see clear evidence for required further research. Since none of the patterns was found during the system review this indicates potential for current application integration system and application endpoint implementations.

The monitoring of integration scenarios reaches from real-time, scenario-specific processing to near-real time monitors. One further challenge — also identified by [MSHP15] and partially covered by the systems with mixed on-premise and cloud integration — is the monitoring across different platforms (e.g., cross-cloud, across on-premise and cloud).

2.3.2 Example Integration Pattern Authoring

As an example from the catalog, we selected patterns related to the non-trivial trade-off between stateful and stateless integration processes (inspired by cloud computing challenges [FLR⁺14]). Especially the system review shows that all vendors provide extensive storage capabilities beyond the EIPs, leading to stateful processes. However, the under-represented conversation patterns could offer an alternative, thus allowing for stateless processes. While stateless processes have scalability benefits, they come with some drawbacks that have to be considered. We selected this trade-off due to its relevance in the context of Big Data, its relevance for Cloud Computing, and because it addresses one well-represented (i.e., storage) and the currently under-represented, but important area of (stateful) conversations. Subsequently, we describe the trade-off as problem description, discuss a suitable pattern format and conclude with two pattern descriptions and their realization.

Problem Description: Stateful vs. Stateless Integration Scenarios

Operating an integration system requires persistent stores and queues, e.g., monitoring, key or secure store to achieve security, or auditing for legal reasons. In addition, transactional message processing (e.g., Aggregator pattern) as well as message delivery semantics (e.g., reliable messaging like *exactly-once-in-order*) [RH15] require some persistent state. While the system operability avoids influencing the message processing by not using shared states between integration scenario instances, the transactional processing and message delivery semantics of the stateful message processors (i.e., patterns) usually require shared state. For example, when a stateful Aggregator—as part of a scenario instance—processes a sequence of messages, a second scenario instance could be used to distribute the load. However, in the absence of “process stickiness” (i.e., messages of one sequence are only sent to one instance), the stateful Aggregator in the second instance has to be able to complete a sequence the other instance started, thus sharing state. Hence, the shared state imply complex state handling across integration processes in compute clusters or cloud environments, and this may have a negative impact on their scalability. Alternatively—following the ideas on (stateful) conversation patterns from Hohpe [Hoh06]—some of the discussed messaging related storage and message delivery semantics could be moved to “smart” message endpoints (i.e., applications), which already have a persistent state, thus making the integration processes stateless.

For example, Figure 2.9 illustrates the trade-off between Exactly-once In Order (EOIO) delivery semantics within the integration scenario (i.e., requires a stateful Message Store, Resequencer and Idempotent Receiver [HW04], and transactional Message Redelivery on Exception [RS16]) in Figure 2.9(a) and as a (stateful) conversational approach in Figure 2.9(b). The integration processes are represented in BPMN 2.0 similar to the notation used in [Rit14a]. An EOIO delivery requires a transactional redelivery in case of an exception, a message ordering step according to a sequence of messages in the form of a Resequencer and an Idempotent Receiver, which is able to deduplicate the messages. Figure 2.9(a) depicts the instance spanning state for the retry and the resequencing. To avoid a stateful integration process, both capabilities can be moved to the endpoints (see Figure 2.9(b)). While this will not work for legacy, packaged applications, it results into an improved scalability within the integration process and moves the resequencing decision to the receiver. To eventually stop sending, the sender—redelivering the message periodically—requires a stop event (i.e., an acknowledgment) from the receiver.

The solution’s trade-off are the several messages that are sent by the receiver until an acknowledgment is received, while being able to process all messages in parallel using stateless integration process instances. That means, the performance improvements gained through better scalability and lower latency of the conversational approach—by not waiting for the failure of a

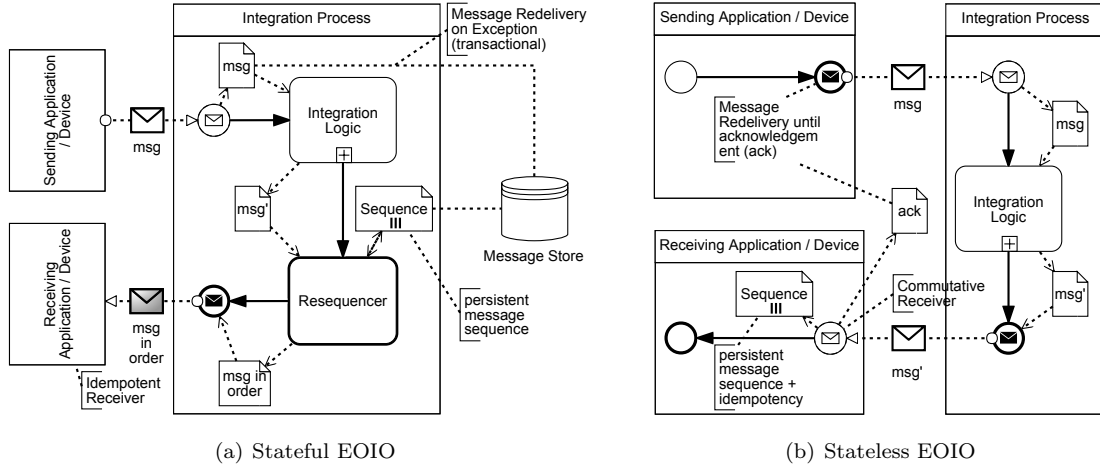


Figure 2.9: Conversational approach for Exactly-Once-in-Order (EOIO) messaging

sent message — is contrasted by the more resources overall required in case of many failures.

Patterns and Pattern Formats

To formalize the new challenges and the resulting, required capabilities within an integration system, thus coming to less immature and ad-hoc solutions (cf. H3), we propose to express them as patterns. Similarly, expert knowledge and best practices were already collected for software design by Gamma et al. [GHJV95], EIPs by Hohpe et al. [HW04], and recently for cloud computing patterns by Fehling et al. [FLR⁺14]. For a suitable pattern representation, we compare their pattern formats in Table 2.7, and select common categories for our proposal.

Table 2.7: Common pattern formats: Enterprise Integration Patterns (EIPs) [HW04], Cloud Computing Patterns (CCP) [FLR⁺14], Design Patterns (DP) [GHJV95]

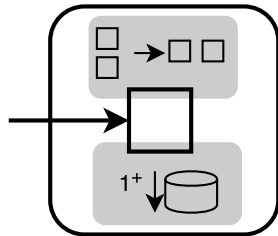
Categories	Description	EIPs	CCP	DP
Name	pattern identifier	✓	✓	✓
Icon	visual representation	✓	✓	-
Problem / Driving Question / Motivation	difficulty as question	✓	✓	✓
Intent	statement about design issue	-	-	✓
Also known as	other pattern names	-	-	✓
Context / Motivation	introduces problem domain	✓	✓	✓
Forces, Applicability	problem constraints	✓	-	✓
Solution	how to solve the problem	✓	✓	-
Sketch, Structure	illustrate solution	✓	-	✓
Participants, Collaborations	participants, responsibilities	-	-	✓
Results / Consequences	how to apply the solution	✓	✓	✓
Next / Related Patterns	related patterns, differences	✓	-	✓
Sidebars / Implementation / Code	pattern variations	✓	-	✓
Examples / Known Uses	real system examples	✓	✓	✓

From the analysis of several known pattern formats in Table 2.7, we selected: name, icon, driving question, context, solution, results, and known uses to round-off the description. We leave out the separate categories of forces (i.e., problem constraints) and implementation (i.e., pattern variants), which we include into the selected context and known uses categories, respectively. The pattern descriptions in [RR15], add a *Data Aspects* category (not further discussed here), which gives even more insight into the configuration of the pattern solutions.

Pattern Examples and Realization

From the problem description we take three capabilities that are required to represent an EOIO, while keeping the integration processes stateless. We summarize the capabilities to the following two patterns, for which we explain the realization: Commutative Receiver and Timed Redelivery until Acknowledge. In addition, we require the Quick Acknowledgment pattern from [Hoh06].

Commutative Receiver The commutative receiver accounts for two tasks: message deduplication and out-of-order handling. Therefore, the application’s state is re-used, hence no additional state in the integration process is required.



How to ensure idempotent, in-order message processing without intermediate state in the form of persistent integration scenarios?

(**Icon:** the icon uses the icon notation from [HW04], combining the in-order sequencing as well as the idempotent storage.)

Context: Out-of-order communication with endpoints / applications.

Solution: Guarantee that endpoint / application handle arriving out-of-order messages will be stored within their sequence and only then processed, if the sequence is (partially) complete and in the correct order.

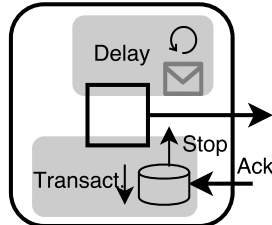
Result: This solution handles out of order messages and applies them in-order within the application endpoint.

Relations to other patterns: This pattern is an extension of the Idempotent Receiver from [HW04] with additional Message Sequence handling.

Known uses: not found in literature or system review, however, Microsoft advises developers to implement commutative endpoints in the context of micro services^a.

^aMicrosoft application architecture guide – designing service applications, visited 5/2019: <https://msdn.microsoft.com/en-us/library/ee658114.aspx>.

Timed Redelivery until Acknowledge The commutative receiver moves the message re-delivery on exception from the integration process to the sender application, while conducting an asynchronous communication. Hence, no exceptions are propagated back to the sender, however, the redeliveries are stopped by asynchronously received Acknowledgments from the receiver (via the integration system). Until then, the messages are resent with increasing delay to reduce the load of duplicate messages.



How to ensure that a message will be received without intermediate storage, e.g., in the form of Redelivery on Exception [RS16]?

(Icon: the icon uses the icon notation from [HW04], combining delayed message send with asynchronous reception of acknowledgments using a transactional store.)

Context: This pattern is used for asynchronous communication with message delivery guarantees.

Solution: Instead of relying on intermediate storage and retry within the integration system, the application sends multiple instances of the same message with configurable timings until the actual receiver endpoint acknowledgments (e.g., Quick Acknowledgment [Hoh06]) reach the sender. Requires an Idempotent [HW04] or Commutative Receiver for certain message delivery semantics [RH15].

Result: Send copies of the same message asynchronously until the receiver's acknowledgment reaches the sender.

Relations to other patterns: This pattern is an extension of the Retry pattern in [Hoh06], and related to the Redelivery on Exception pattern in [RS16].

Known uses: – (not found in literature or system review).

Solution Summary As an extension of the existing pattern catalog a *Timed Redelivery until Acknowledge* pattern would be required that makes multiple attempts to deliver a message (potentially with exponential back-off delay). That might result in duplicate message instances, sent to the receiver. Assuming a stateless integration process, an idempotent receiver [HW04] is required to detect and handle the duplicates. The sketched conversation works fine for exactly-once processing semantics [RH15]. However, for ensuring in-order message processing (e.g., create sales order, before update), it would not be sufficient. A stateless integration process cannot reliably re-order the incoming messages, delegating this task to the receiver application. The receiving application has to handle incoming messages in an associative and commutative way (e.g., handle update, before create). An implementation of this *Commutative Receiver* pattern can be found in the microservice context (e.g., service applications). When the receiver has received all messages belonging to one message sequence (i.e., without duplicates), it sends an acknowledgment that is asynchronously processed by the sender, which stops the sender redelivering corresponding messages immediately.

2.4 Quantitative Analysis

In this section we conduct a quantitative analysis of integration scenarios to study the usage of original EIPs and the new patterns from the pattern catalog in [Section 2.3](#). We selected the SAP Cloud Platform Integration system from the review (see [Section 2.2.3](#)), for which we found “real-world” examples of all scenarios from [Figure 1.3](#) (on [page 6](#)). With over 5,000 customers and several hundred integration scenarios delivered as standard content SAP Cloud Platform Integration represents a cloud integration system for application and data integration. The analysis targets the claim “The original EIPs of the 2004 book are all widely used in praxis” and hypothesis “H4: Solutions not in EIPs can be found in real-world integration scenarios for the trends”. Therefore, we firstly select several scenarios along the identified trends and briefly describe them. Secondly, we evaluate found original EIPs and new solutions.

2.4.1 Integration Scenarios

Originally, EAI focused on the integration of applications within a single organization. However, as hosting (parts of) applications in the cloud becomes increasingly popular, EAI also needs to address scenarios where applications that are hosted in the cloud or on-premise (i.e., within company networks) need to be integrated. We refer to such scenarios as *hybrid applications*, following Forrester [\[For16a\]](#). Especially hybrid applications require a stronger decoupling to integrate on-premise with cloud applications, and consequently, hybrid applications prefer to use (asynchronous) message-based communication patterns, while RPC-style integration is still quite common for EAI in on-premise setups. Most of the current research focuses on RPC-style Service-oriented Architecture (SOA). The new trends — set into context denoted by edges via the integration system node in [Figure 1.3](#) — can be summarized as integration scenario domains:

- On-Premise to Cloud: Most organizations productively run on packaged, on-premise applications. They need to connect these applications with cloud applications for various reasons, e.g., extensions for legal reasons or new functionality, to connect with business partners. This integration domain is called hybrid integration [\[For16a\]](#).
- Cloud to Cloud or Business Network (including social): Native cloud applications or cloud integrated on-premise applications interact with business partners in business networks (e.g., payment, financial, supplier relationships), connect to social networks (e.g., social marketing) or consume cloud services (e.g., machine learning).
- Device to Cloud (including mobile and personal computing): What starts with business applications on “bring your own device” for mobility, extends to intelligence brought to machines (e.g., sensors and actuators in smart logistics or production) and eventually comes down to sensors and devices for personal computing.

We left out the conventional On-Premise to On-Premise application integration and other variations due to our focus on new trends. For the quantitative analysis, we selected one application integration solution for each of the new scenario domains, and we added one for cloud to cloud and business networks due to the slight focus on these domains. The solutions can be visited as SAP Cloud Platform Integration standard content [\[SAP18a\]](#).

SAP Cloud for Customer (C4C) The C4C solutions for the communication with on-premise Enterprise Resource Planning (ERP) and Customer Relationship Management (CRM) applications, abbreviated *c4c-erp* and *c4c-crm*, can be considered a typical hybrid corporate to cloud application integration [\[SAP18a\]](#). The dominant integration styles — according to the

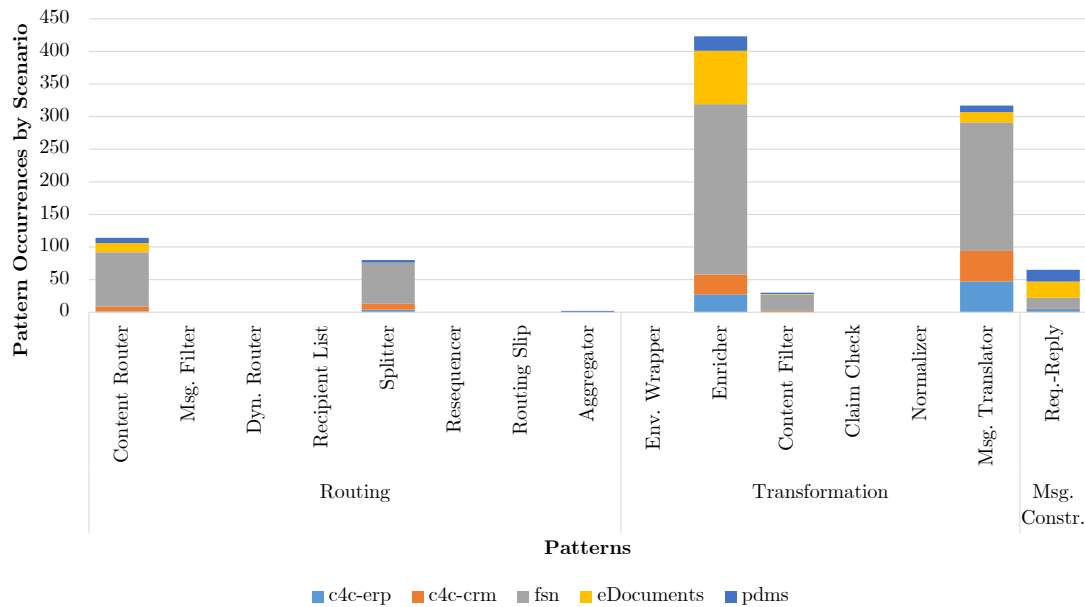


Figure 2.10: Scenarios using original EIPs

classification in [Lin00] — are process invocation and data movement. The state changes (e.g., create, update) of business objects (e.g., business partner, opportunity, activity) as well as master data in the cloud or corporate applications (e.g., ERP, CRM) are exchanged with each other.

SAP Financial Services Network (FSN) In contrast to C4C, FSN [SAP18b], abbreviated *fsn*, is a cloud-based, business network that connects banks and other financial institutes with their corporate customers (e.g., for payments, bank account management). The predominant integration style is process invocation [Lin00], and besides reliability, the major focus lies on secure message exchange.

SAP Cloud Platform Integration eDocument/Electronic Invoicing (eDocument) The SAP Cloud Platform Integration eDocument/Electronic Invoicing is a solution for country-specific electronic document management [SAP18a]. The *edocuments* solution helps cooperations to interact with legal authorities (e.g., implement the new *EU Data Protection Regulation*⁷).

SAP Predictive Maintenance and Service (PdMS) In PdMS [SAP18a], machine data is collected using an Internet of Things (IoT) platform and enriched with business information coming from, e.g., SAP Business Suite. This allows real time monitoring of the machine that triggers alerts resulting in service tickets in SAP CRM or ERP. Based on that any unusual trends or behavior becomes visible and appropriate action, potentially avoiding unnecessary service costs, can be taken.

⁷EU – General Data Protection Regulation, visited 5/2019: <https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%3A52012PC0011>.

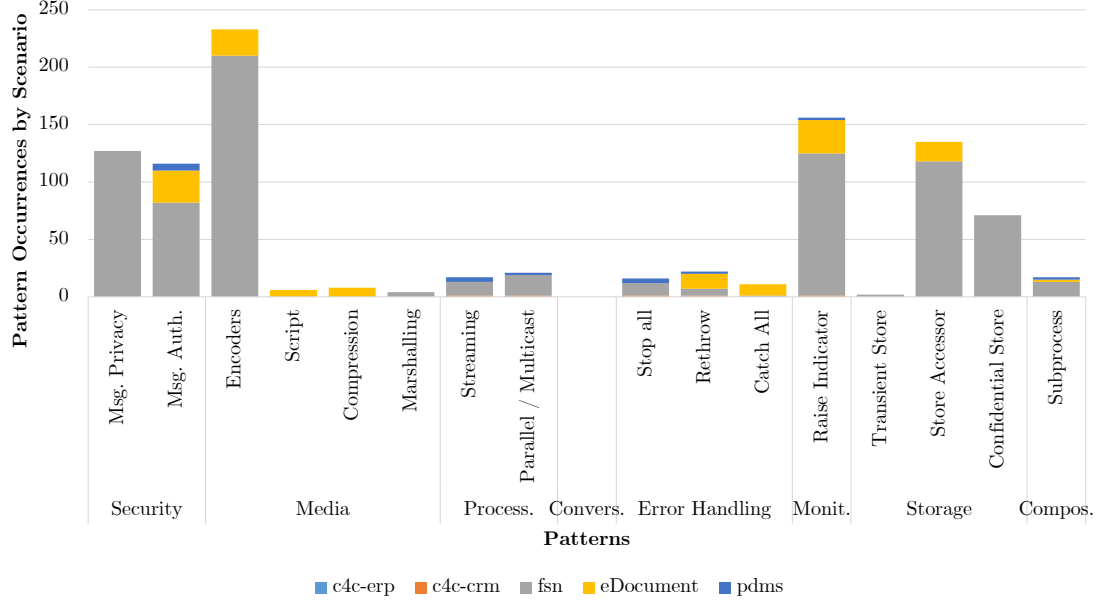


Figure 2.11: New capability categorization

2.4.2 Scenario Analysis

For the analysis of the SAP delivered standard content in this paper, we prototypically implemented data discovery and mining capabilities into the SAP Cloud Platform Integration system, which identified a total of 154 distinct integration scenarios (c4c-erp: 42, c4c-crm: 37, fsn: 56, edocument: 13, pdms: 6).

The total number of patterns for all scenarios is 1501 (without adapters, endpoints). For the more “classical” integration scenarios in c4c-erp and c4c-crm nearly all integration patterns could be covered by original EIPs from [HW04] (apart from second level configuration on monitoring and exception handling). For the cloud integration scenarios fsn and eDocument as well as for the pdms IoT scenario, 466 new requirements (and 597 complementing, second level configurations) were needed in total, out of which 66% are covered by existing EIPs (i.e., 1025 capabilities in total). This means that for these integration scenarios approximately $\frac{1}{3}$ of the required patterns are not covered by the original EIPs.

Pattern Solutions covered by the EIPs The distribution of patterns covered by original EIPs is depicted in Figure 2.10. Not all EIPs were required in the scenarios of the integration solutions, however, nearly all of them facilitate the tasks of (i) Message Construction: solutions *Document Message* and *Request-Reply*; (ii) Messaging Channels: solution *P2P Channel*; (iii) Message Routing: solutions *Content-based Router*, *Splitter*, and *Aggregator*; (iv) Message Transformation: solutions *Content Enricher*, *Content Filter*, and *Message Translator*.

New Pattern Solutions Additional pattern solutions are covered by integration capabilities, depicted in Figure 2.11. We grouped the new solutions according to the pattern catalog from Section 2.3 and added the corresponding pattern proposals for each of them. While approximately half of the new patterns from the catalog are used in the real-world scenarios, the new conversation

patterns, are not yet used in any of the scenarios. For example, the confidentiality requirements covered message confidentiality or privacy patterns (Message Encryptor, Message Decryptor, Encrypted Message, Encrypting Endpoint, Encrypting Adapter) are called *Msg. Privacy*, and the authenticity and integrity requirements (Message Signer, Signature Verifier, Signed / Verified Message) are summarized to message authenticity (*Msg. Auth.*). The message confidentiality is exclusively required for the communication within the FSN business network, while message authenticity is also used for exchanging eDocuments with the legal authorities and for PdMS.

In the category multimedia, currently no real media format handlers (Type Converter, Encoder, Decoder) are used (e.g., image, video). However, we grouped the used functionality into three patterns. The Encoder and Decoder patterns represent the handling of binary message content, exclusively used in FSN and eDocument scenarios. This is due to the various communication partners using different encodings, as well as third party services (e.g., financial document mapping engines), which requires special encodings. The Custom Script allows to add versatile User-defined Functions, which are mostly used as auxiliary in FSN scenarios. The compression algorithms (Compress Content, Decompress Content), which would be immensely relevant in real multimedia scenarios, are used in FSN scenarios due to larger messages sizes (e.g., aggregated FSN payment details). Finally, marshalling (Marshaller, Unmarshaller) support is required in FSN scenarios, since some communication partners require JSON to XML conversion and vice versa.

The processing of messages is mostly done by moving messages through the pipeline. However, especially the FSN and CRM integration require streaming and parallel processing. This is again due to scenarios with larger messages to be processed (e.g., IDoc segments in CRM) and stream-based splitting in PdMS. The Multicast pattern is used as Sequential Multicast in FSN to allow guaranteed rollback for all branches in case of an error and as Parallel Multicast in PdMS for parallel processing purpose.

This behaviour is complemented by a Stop All setting in the FSN, PdMS and partially CRM scenarios, consequently stopping the message processing in all parallel instances of the integration scenario. Another error handling functionality is the Rethrow, which allows to (re-) throw exceptions. The Rethrow is mainly used in eDocument, FSN, PdMS and CRM scenarios to indicate that a situation is still unresolved. Especially in FSN, PdMS and eDocument scenarios, it is important to inform a business expert or administrator about erroneous situations. The Raise Indicator is used for this purpose. To prevent uncontrollable behaviour and to customize the information returned in case of an error in synchronous scenarios, a Catch-all exception process (with several steps) is used in FSN and eDocument scenarios.

To remember parts of a message or additional information generated by adapters or message processors, a Transient Store (cf. [GR11]) is used in FSN. The Store Accessor, used by FSN and eDocument, is mostly used for stateful pattern compositions and for legal reasons (Data Store, Audit Log). Especially in FSN, most of the message stores are encrypting (Encrypting Store), which means that the messages are stored confidentially.

The composition in terms of the Integration Subprocess pattern (excluding the exception sub-processes) indicates complex processing logic, which can mostly be found in FSN, PdMS and eDocument scenarios. Sometimes composition is used for re-usable process parts.

In summary, the analysis shows the need for new patterns and solutions as seen in the system review. While the hybrid integration scenarios simply extend the on-premise integration into the cloud relying on transport level security, all other new integration scenario domains require more on security and control over the message processing in the form of error handling. This becomes more obvious, the more exclusively the integration scenarios are running in the cloud. For example, the small amount of device integration scenarios still relies on integration logic on the devices, thus show only few security, error handling and processing capabilities. The scenarios are less complex compared to the cloud to cloud cases, hence require limited composition options. Moreover, with

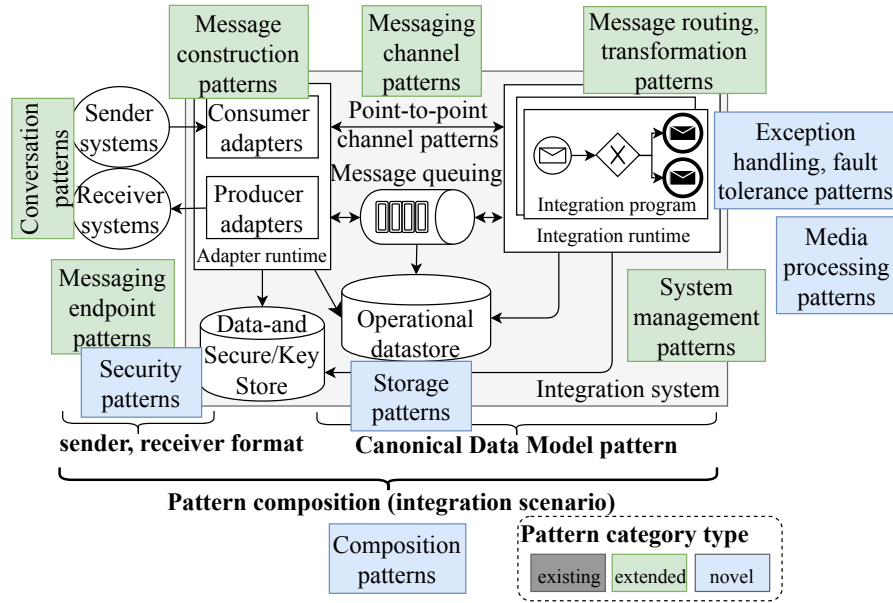


Figure 2.12: Integration system architecture with unaltered (dark grey), extended (green) and novel pattern categories (blue)

increasing cloud focus, the trade-off between more complex, but expressive, stateful and simpler, better scalable, stateless message processing seems to lean towards the interaction with storage. The conversation patterns — including stateful conversations — are still mostly unexplored. The same is true for the increasingly important topic of multimedia processing, which will give a new edge to the variety and interoperability problems in EAI [Lin00].

2.5 Conclusions

The literature and the quantitative analysis of real-world integration scenarios (see Figure 2.10) show that some of the EIPs described by [HW04] in 2004 are still widely used, thus denote best-practices in the area of application integration. This supports the assumption of the EIP authors that the patterns are still practically relevant [ZPHW16]. The literature and system reviews also reveal that since 2005 several Non-Functional Aspects (NFAs) for enterprise application integration have appeared that are not covered by the EIPs from 2004. Literature as well as systems partly offer solutions to these new trends and NFAs where the systems provide a more comprehensive support. Solutions mentioned in the literature comprise patterns, formalization, and modeling, covering the spectrum from a more abstract description as patterns (see Section 2.3.2) to the implementation and execution as well as towards the user’s point of view. For this reason, patterns are regarded as the “glue” for which a comprehensive description is required first. Hence, this analysis (together with the pattern language [RR15]) aimed at filling the gaps in existing patterns for new integration trends and NFAs (see Section 2.3.2): e.g., security, (ideas on) conversations, monitoring, storage. Figure 2.12 summarizes extended and novel pattern categories in the context of the integration system architecture from Figure 2.4 (on page 26). Therefore, the new patterns were assigned to existing pattern categories, where possible, marked as **extended**. The remaining new patterns that do not fit to these categories, were assigned to **novel** pattern categories:

security, storage, exception handling and fault tolerance, media processing, conversations and composition patterns. Some of the existing categories were extended by fitting new sub-categories: message construction and (integration) adapter, and system management and monitoring patterns. Notably, none of the existing categories remain unaltered and the architecture, denoting the EAI domain concepts, is now more comprehensively covered by patterns.

The results in this chapter denote contributions in the form of DSR artifacts:

- A systematic literature review of the trends, e.g., cloud and hybrid application integration approaches (\mapsto H1), and an analysis of the most influential system implementations of this domain (\mapsto H2).

We found for H1 that additional patterns are required to suffice for application scenarios after 2004, and for H2 that they were (mostly) found in current system implementations.

- An extended pattern template plus an example based on descriptions of cross-concern technical qualities (e.g., (stateful) conversation, streaming, security) for a comprehensive coverage of new requirements (\mapsto H3).

For H3, our study showed that some trends are handled in an immature and ad-hoc fashion.

- A comprehensive pattern catalog, documented in [RR15], and the evaluation of the found patterns as part of integration scenarios in a well-established cloud integration system in the form of a quantitative analysis based on new monitoring patterns (\mapsto H4),

H4 was affirmed by the study of current, real-world integration scenarios.

This validates all stated hypotheses, and thus fulfills research challenge C1 “Actuality, Comprehensiveness” and addresses the *pattern research gap* in Section 1.2 by providing a comprehensive foundation for the subsequent formalization of patterns and their composition. Nonetheless, the different reviews and analyzes conducted in this work indicate open research challenges. These are subsequently summarized and discussed.

Limitations Limitations of the approach concern the literature and the system review. Both searches were led by the selection of keywords and criteria due to the vast amount of existing work and in order to not lose focus of this study. Nonetheless, further vertical searches and expert additions that were not found based on the keywords could be included in the analysis. The selection of representative systems is supposed to reflect the current system offering. Different types of systems (open source, commercial, and startup) were considered. In summary, both reviews were envisioned to be comprehensive, but not complete. The quantitative analysis aims at covering a broad range of applications based on five use cases. One might argue that the use cases are all provided by the same organization. However, SAP’s customers are diverse, and thus cover all relevant application integration scenario domains related to the trends. This provides the chance to analyze real-world scenarios instead of toy examples.

Impact The impact of a continuous analysis of integration trends and NFAs on research and practice is enormous. The impact on research is reflected in the open research challenges stated in Section 1.2.3. In order to address these challenges, a plethora of new approaches is necessary. The importance of the topic from a practical perspective is already paramount as the system and scenario analyzes of this chapter show. Facing new trends that often stem from practice will perpetuate the importance of this work in the future. Putting the focus on the human aspect in addition to a more technical treatment of the topic will also lead to a multitude of new research questions and practical implications. While the original EIPs from 2004 are still relevant for many of the new trends in 2016 and beyond, new capabilities are required to address requirements (e.g., non-functional aspects) resulting from these trends (see hypothesis H1).

Chapter 3

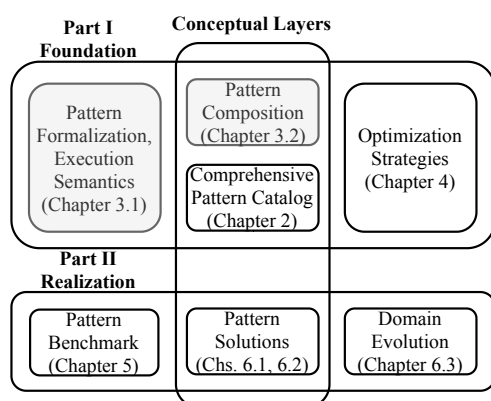
On Formalizing Integration Patterns and their Compositions

Contents

3.1	Formal Pattern Semantics	62
3.2	Composing Patterns	101
3.3	Related Work	129
3.4	Discussion	132

[..] people who write software should have a clear sense of responsibility for its reliable operation and resistance to compromise and error.
About “responsible programming” by Vinton Cerf, 2014 [Sta14]

In this chapter, we build on the extended integration pattern catalog and develop trustworthy integration foundations that allow for a responsible development of integration solutions.



We recall from [Chapter 2](#) that integration patterns are still relevant and their compositions denote integration scenarios that run at the core of many organizations. Due to the identified challenges like the increasing complexity of integration scenarios or the reduced control over integration solutions as part of the trending cloud and mobile / edge computing operation models (cf. [Section 1.2](#)), *trust* into productive integration solutions becomes even more essential. For example, current EAI system vendors use integration pattern compositions as part of their proprietary integration scenario modeling languages (cf. [Chapter 2](#)). Since the integration patterns are only informally described, these languages are not grounded on any formalism and, hence, might produce

formally described, these languages are not grounded on any formalism and, hence, might produce

models that are subject to design flaws (e.g., functional errors). Due to the missing formal definition, currently the detection and analysis of these flaws are by large performed manually. This results in huge effort and potentially in mistakes. Moreover, the integration solutions developed on the foundation of the patterns are not exclusively used (and tested) by technical users anymore. The provided solutions are used in (so far) completely non-technical fields and areas of human life [Sta14, Gol18]. Hence deep technical understanding (e.g., when using it in the context-aware health or entertainment applications on a mobile phone [RH18]) should not be required of users, for whom the desired behavior and functional correctness is essential, users should not be asked to deal with flaws.

This, in turn, requires the means for a *responsible* development of integration solutions (cf. “responsible programming” [Sta14]) in order to avoid design flaws such as functional errors or incomplete functionality, starting with the integration patterns. Therefore, the expected behavior of the single patterns in the form of execution semantics has to be formally defined and their composition to integration scenarios has to fulfill compositional and functional correctness criteria. Such a formalism would allow for an automatic analysis of integration solutions like verification (model-checking), validation and simulation to assess the functional correctness when composing patterns, as well as identification of more efficient pattern compositions (i.e., optimization of integration scenarios).

However, as found in our study in Chapter 2 and also acknowledged by the pattern authors [ZPHW16, p. 16] and recent other studies [FG12, FG13], these formal foundations are missing (cf. *formalization research gap* in Section 1.2). Hence, we address the general sub-research questions of RQ2 “How to formulate integration requirements and scenarios in a usable, expressive and executable integration language?” for single pattern formalization and pattern compositions:

RQ2-1 *What is a suitable formalism for defining execution semantics of existing and new patterns?*

RQ2-2 *What is a sound and comprehensive formal representation of integration patterns that allows for formal validation of integration scenarios and reasoning?*

The resulting formalized pattern compositions (grounded on formally defined patterns) not only provide the first comprehensive formal foundation of application integration, but will allow for the specification of correctness-preserving optimization strategies and the definition of a suitable benchmark and sound practical solutions in the remainder of this work.

Parts of this chapter have previously been published in the proceedings of EDOC 2018 [RRMM⁺18] (pattern formalization), DEBS 2018 [RMFR18] (pattern composition formalization) and a technical report [RRM⁺18] (catalog formalized patterns).

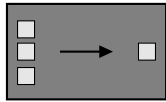
3.1 Formal Pattern Semantics

The extended integration pattern catalog in Chapter 2 describes typical concepts in designing messaging systems as used for EAI (e.g., the communication between applications), and thus an abstraction of the application integration domain. However, due to the informally described EIPs models may be produced that are subject to design flaws such as functional errors, missing or incomplete functionalities. Currently, detection and analysis of these flaws are by large performed manually, as also recently found by Fahland and Gierds [FG13]. Hence, EIPs can rather be considered as a set of informal design solutions than a formal language for modeling and verifying correctness of integration patterns, thus leaving the EAI vendors with their own proprietary semantics, which also hinders compatibility between their implementations. The following example illustrates the current description of the Aggregator pattern from [HW04], which we use as running example during the definition of our formalism. For a better understanding we added

further information, which we extracted from the original description. After these introductory explanations, the pattern is structured in the integration pattern format from [Section 2.3.2](#).

Example 3.1. The subsequent representation of the Aggregator from [\[HW04\]](#) describes the Aggregator as a pattern that combines multiple related incoming messages to one outgoing message. The incoming messages are persistently stored until the sequence of messages is complete. In the implementation details that are given as unstructured text (not shown) [\[HW04, pp. 270–274\]](#), a *timeout* is mentioned, which completes a sequence of related messages and considers only those available within the time limit.

Aggregator (excerpts taken from [\[HW04, pp. 268–282\]](#)) The Aggregator is a special Filter that receives a stream of messages and identifies messages that are correlated. Once a complete set of messages has been received (more on how to decide when a set is “complete” below), the Aggregator collects information from each correlated message and publishes a single, aggregated message to the output channel for further processing.



How do we combine the results of individual, but related messages so that they can be processed as a whole?

(Icon:^a The icon is from the EIP icon notation [\[HW04\]](#).)

Context: A Splitter is useful to break out a single message into a sequence of sub-messages that can be processed individually. [...] In most of these scenarios, the further processing depends on successful processing of the sub-messages. For example, we want to select the best bid from a number of vendor responses or we want to bill the client for an order after all items have been pulled from the warehouse.

Solution: Use a stateful filter, an Aggregator, to collect and store individual messages until a complete set of related messages has been received. Then, the Aggregator publishes a single message distilled from the individual messages.

Result: Several related messages are combined into one new message.

Relations to other patterns: Correlation Identifier, Message Sequence.

^a©The EIP pattern icon, the pattern name, the problem and solution statements and the sketch are available under Creative Commons Attribution license.

While the description in the example might help when building an integration system or using one, a formalism is required that allows for analysis according to the following objectives or design choices for a suitable formalization: (i) realization of EIPs as well as (ii) simulation of the EIP realizations, (iii) the validation and verification of their functional correctness, and (iv) composability to integration scenarios. The objectives target (i) a *responsible* definition of integration patterns, which means that the patterns and their execution semantics can be (ii,iii) formally analyzed for correctness and (iv) allows for composition of formalized patterns. In other words, this gives users creating an integration scenario or solution a “clear sense for its reliable operation and resistance to compromise and error”, according to Cerf [\[Sta14\]](#). We develop and study such a formalism driven by the following detailed sub-questions of RQ2-1:

- (a) *What are relevant EAI requirements for the formal definition of EIPs?*
- (b) *Which formalism allows to specify, simulate and verify EIPs under extracted requirements?*
- (c) *How to realize the EIPs and real-world integration scenarios?*

To address these questions, a *responsible* development of integration solutions, i.e., solutions that can be thoroughly tested for their correctness at design time, requires the formalization of

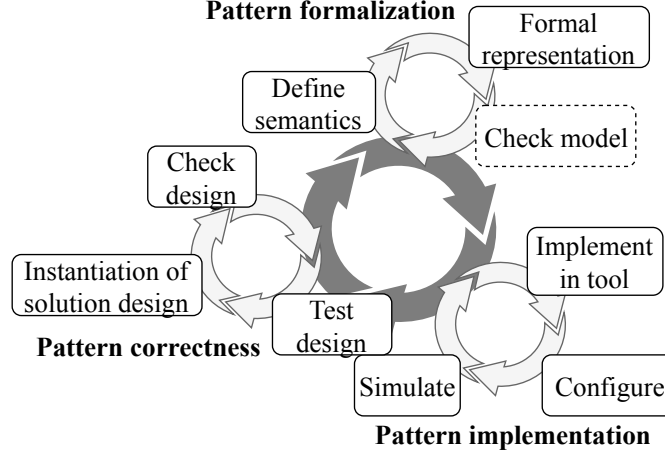


Figure 3.1: *Responsible* pattern formalization process

its pattern foundations. We follow a *responsible pattern formalization process* — similar to the pattern engineering process Figure 2.5 (on page 27; see also [Han12, FBBL14]) — that allows for the objectives. Figure 3.1 shows this process with its three main steps that we subsequently discuss: pattern formalization, pattern implementation, pattern correctness.

Formalization The formalization of a pattern starts with capturing and defining its semantics (cf. question RQ2-1(a)). With a thorough understanding of the pattern and its variations, it can be formally represented (cf. question RQ2-1(b)). The resulting formal pattern model can be analyzed and verified (i.e., model checking). With model checking capabilities, errors in patterns can be found and either their semantics or formal representation is revisited.

Implementations If model checking is not possible or difficult, the formal patterns can be implemented, configured and simulated in a suitable tool (cf. question RQ2-1(c)). The simulation not only bridges the model to implementation gap, but allows for an experimental validation of a pattern.

Correctness The correctness of a pattern can be decided according to its semantics, when put into the context of a dedicated, scenario-specific configuration, a test design, which specifies the desired properties like the expected output of a pattern simulation, for a given input. This test design is instantiated and checked during the simulation of the pattern (cf. question RQ2-1(b)). Any flaws found during this step can results in another round of formal or implementation adjustment.

We argue that existing approaches do not fully support a responsible development and hence the following research questions are formulated to guide the design and development of an EIP formalization living up to objectives (i)–(iv). Our analysis in Chapter 2 identified only one attempt towards a formalization of EIPs using Coloured Petri Nets (CPNs) by Fahland and Gierds [FG13]. Although the CPN colors abstractly stand for data types and CPNs support the control flow through control threads (i.e., tokens) progressing through the net, carrying data conforming to colors, they cannot be used to model, query, update, and reason about requirements inherent to the extended EIPs from Chapter 2, such as persistent data or timings.

Along the responsible pattern formalization process, we systematically harvest pattern requirements from Chapter 2 (i.e., catalogs with 166 integration patterns) in a quantitative analysis

(addressing RQ2-1(a)) in [Section 3.1.1](#). The resulting list of requirements is used to identify a suitable formalism, compliant with the objectives (i)–(iv), which leverages the existing CPN approach [\[FG13\]](#) by adding persistent data and time, consecutively deriving the *timed db-net* formalism, for which we also study decidability of reachability (cf. RQ2-1(b)) in [Section 3.1.2](#). Therefore we briefly introduce background information on the used formalisms, namely db-nets and TAPNs that fulfill the requirements and are compliant with the objectives (i)–(iii) for single patterns close to where they are used. Several of the patterns are then realized using timed db-net in the form of an instructive pattern catalog (i.e., formalizing representatives patterns to instruct the formalization of similar patterns). Then we present means to validate their correctness (all in [Section 3.1.2](#)), before we elaborate on the comprehensiveness of the formalism in a quantitative study. We further show a prototypical db-net realization for testing correctness, and discuss the general applicability of PNs and, in particular, timed db-nets for the composition of integration patterns (cf. objective (iv) and research question RQ2-1(c)) in a real-world example (all in [Section 3.1.3](#)).

3.1.1 Formalization Requirements Analysis and Design Choices

We collect the EAI requirements relevant for the formalization of the EIPs by analyzing the pattern catalogs [\[HW04, RMRM17, RS16\]](#) (cf. RQ2-1(a)), and briefly discuss which of them can be represented by the means of prior work on CPNs, and which require further extensions. Then we discuss design choices based on the requirements.

Pattern Analysis and Categories The EIP formalization requirements are derived by an analysis of the pattern descriptions based on the integration pattern catalogs from 2004 [\[HW04\]](#) (as **original**) and recent extensions in [Chapter 2](#) and [\[RMRM17, RS16, RR15, RH15\]](#) (as **extended**) that consider emerging EAI scenarios (e.g., cloud, mobile and internet of things). Together the catalogs describe 166 integration patterns, of which we consider 139 due to their relevance for this work (e.g., excluding abstract concepts like Canonical Data Model or Messaging System). During the analysis, we manually collected characteristics from the textual pattern descriptions (e.g., data, time) and created new categories, if not existent.

The reoccurring characteristics found in this work allow for a categorization of patterns as summarized in [Figure 3.2](#) to systematically pinpoint relevant EAI requirements into general categories (with more than one pattern). Most of the patterns require (combinations of) *Data* flow, *Control* (Ctrl.) flow, and (*Transacted*) *Resource* ((Tx.) Res.) access. While the control flow denotes the routing of a message from pattern to pattern via channels (i.e., ordered execution), the data flow describes the access of the actual message by patterns (incl. message content, headers, attachments). Notably, most of the patterns can be classified as control (Ctrl.-only; e.g., Wire Tap) and data only (Data-only; e.g., Splitte) or as their combination (Data-Ctrl.; e.g., Message Filter), which stresses on the importance of data-aspects of the routing and transformation patterns. In addition, resources denote data from an external service not in the message (e.g., Data Store from [Chapter 2](#)). The EIP extensions add new categories like combinations of data and {time, resources} (Data-Time like Message Expiration [\[HW04, RMRM17\]](#), Data-Res. like Encryptor from [Chapter 2](#)) and control and time (Ctrl.-Time; e.g., Throttler from [Chapter 2](#)). For instance, our motivating Aggregator pattern example in [Figure 3.3](#) (on [page 68](#)) for the formalization is classified as *Data-Tx.-Res.-Time*. The different categories are disjoint with respect to patterns.

From Categories to Requirements The first requirement is a formal representation and analysis capabilities on the control flow **REQ-0 “Control flow”**, which is inherently covered by any PN approach, and thus in CPN. However, there are two particularities in the routing

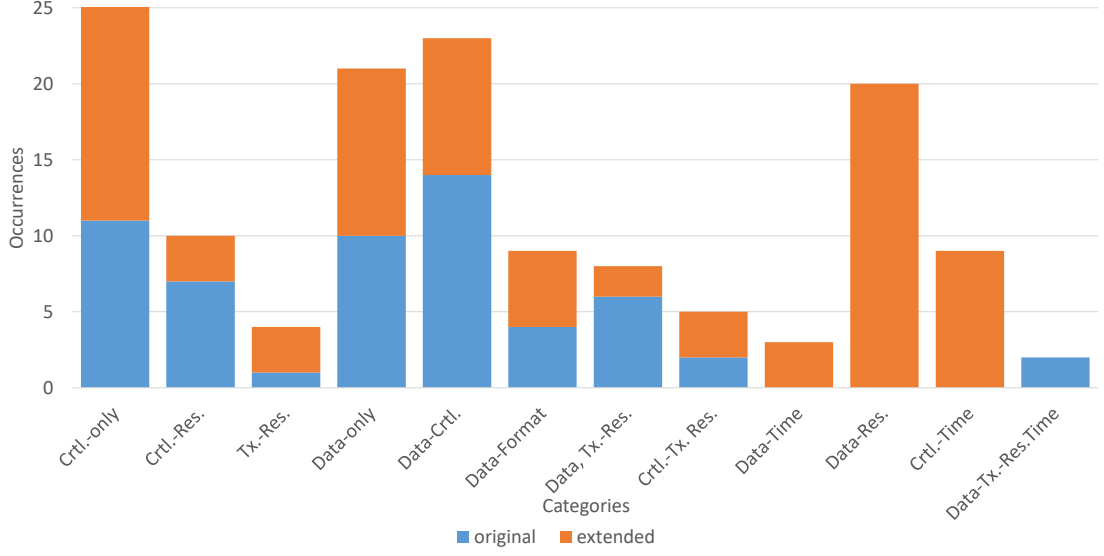


Figure 3.2: EIP requirement categories (with control (ctrl.), resource (res.), transaction (tx.))

patterns that we capture in requirement **REQ-1 “Msg. channel priority, order”**: (a) the ordered evaluation of Msg. channel conditions or prioritized evaluation of guards of sibling PN transitions, required for the Content-based Router pattern, (b) the enablement or firing of a PN transition according to a ratio for the realization of a Load Balancer (cf. Chapter 2). In both cases, neither execution priorities nor ratios are trivially in CPN.

Furthermore, there are 77 patterns in the catalogs with data and 10 with message format aspects, which require an expressive CPN token representation (e.g., for encodings, security, complex message protocols), for which we add as a second requirement **REQ-2 “data, format”** that has to allow for the formal analysis of the data. Although CPNs have to be severely restricted (e.g., finite color domains, pre-defined number of elements) for that, we require a formalism that promises a relational representation that can be formally analyzed.

We capture the 11 patterns with time-related requirements as **REQ-3 “time”**: First, (a) a timeout is required that allows for the numerical representation of fixed, relative time (no global time). An expiry date (b) must be defined, denoting a discrete point in time according to a global time (i.e., based on existing message content). Moreover, the specification of a delay (c) is required that is a numerical, fixed value time to wait or pause until continued (e.g., also often used in a redelivery policy). Finally, patterns like the Throttler, require (d) message/time ratio processing, which specifies the number of messages that are sent during a period of time. Consequently, a quantified, fixed time delay or duration semantics is required.

The 49 patterns with resources **REQ-4 “(external) resources”** require: (a) create, retrieve, update, delete (**CRUD**) access to external services or resources, and (b) **transaction** semantics on a pattern level. Similarly, exception semantics are present in 28 patterns as **REQ-5 “exceptions”**, which require **compensations** and other post-error actions. Consequently, a PN definition that allows for reasoning over these timing and structured (persistent) data access is required.

Requirements Summary and Design Choices Table 3.1 summarizes the formalization requirements, comparing with the coverage of the CPN [FG13]. While CPNs provide a solid

Table 3.1: Formalization requirements

ID	Requirement	Example pattern	CPN
REQ-0	Control flow (pipes and filter)	Multicast	✓
REQ-1	(a) Message channel priority	Content-based	(✓)
	(b) Message channel distribution	Router Load Balancer	-
REQ-2	Data, format incl. message protocol with encoding, security	Encoder	(✓)
REQ-3	(a) Timeout on message, operation	Pause Operation	-
	(b) Expiry date on message	Message Expiry	-
	(c) Delay of message, operation	Delayer	-
	(d) Message/time ratio	Throttler	-
REQ-4	(a) Create, Retrive, Update, Delete (CRUD) operations on (external) resources	Content Enricher	-
	(b) Transaction semantics on (external) resources (incl. roll-back)	Aggregator	-
REQ-5	Exception handling, compensation (similar to roll-back in REQ-4 for transactional resources)	Catch All, Selective	-

covered ✓, partially (✓), not -.

foundation for control (cf. REQ-0) and a simple data flow representation (cf. REQ-2), more complex data structures are not supported, e.g., message protocols in our case (cf. REQ2) and corresponding CRUD operations (cf. REQ-4(a)), transactional semantics (cf. REQ-4(b)), and exception handling (cf. REQ-5), suitable for working with external, transactional resources. In CPNs, message channel distributions cannot be represented and priorities require explicit modeling, leading to complex models. For a comprehensive coverage of all requirements we subsequently discuss alternative formalisms from the PN domain and motivate our choices.

Data, Transacted Resources and Compensation. When facing the problem of formalizing multi-perspective models that suitably account for the dynamics of a system (i.e., the process perspective) and how it interacts with data (i.e., the data perspective), several design choices can be made. In the Petri net tradition, the vast majority of formal models striving for this integration approaches the problem by enriching execution threads (i.e., tokens) with complex data. Notable examples within this tradition are data nets [Las16] and ν -nets [RVdFE11], Petri nets with nested terms [TS16], nested relations [H⁺08], and XML documents [BHM15]. While all of the approaches treat data subsidiary to the control-flow dimension, the EIPs require data elements attached to tokens being connected to each other by explicitly represented global data models. Consequently, they do not allow for reasoning on persistent, relational data such as tree or graph structured message formats [Rit17a]. We select db-nets [MR17] as a foundation of timed db-nets due to their ability to represent relational data (cf. REQ-2), and the built-in support for transactional CRUD operations (cf. REQ-4) as well as exception handling that require compensations (cf. REQ-5), as set into context to the requirements in Table 3.1. Moreover, since db-nets are based on CPNs, it is possible to leverage existing simulation techniques of the latter [MR17].

Time. While the implicit temporal support in PNs (i.e., adding places representing the current time) is rather counterintuitive [vdA93], the temporal semantics of adding timestamps to tokens [vdA93], timed places [Sif80], arcs [JJMS11] and transitions [Zub87] are well studied and naturally capture different facets of time in dynamic systems. The temporal requirements in REQ-3 demand a quantified, fixed or discrete time representation by timed transitions or places, representing the delay induced by a transition enablement or firing. This is currently missing in db-nets. So, in the spectrum of timed extensions to PNs, we select temporal semantics similar to Timed-arc Petri

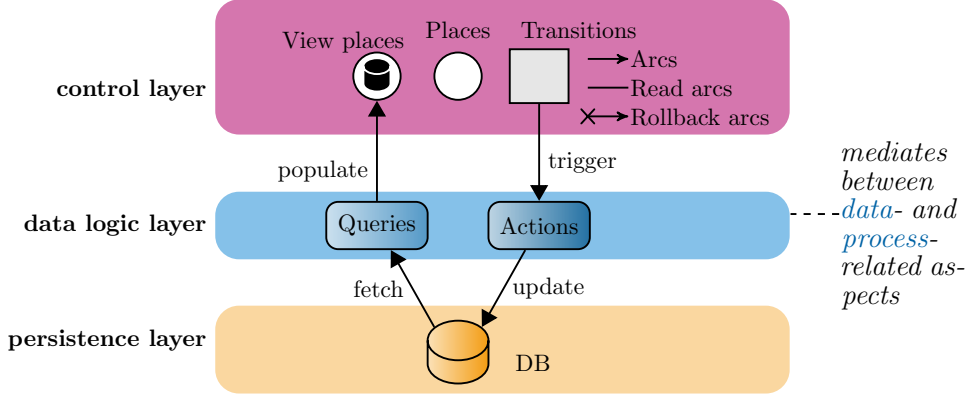


Figure 3.4: Db-net layers (similar to [MR17])

special view place ch_p , and then aggregates them based either on the completion condition (e.g., sequence status is *complete*, modeled via *Aggregate* transition) or on time-out of 30 time units (e.g., sequence status is *expired*, modeled via transition T_3). To collect messages and assign them to correct sequences, the net correlates every incoming *msg* token to those in place ch_p , that, in turn, stores pairs of sequences and lists of messages that have already been collected. If the message is the first in a sequence, new entries, one containing information about the message and another containing data about the referenced sequence, are added to tables called *Messages* and *Sequences*, respectively. This is achieved by firing transition T_1 and executing action *CreateSeq* attached to it. Otherwise, a message is inserted into *Messages* by firing T_2 and executing *UpdateSeq*. However, the update by *UpdateSeq* fails, if a message is already in the database or a referenced sequence has already been aggregated due to a timeout (i.e., status is *expired*). In this case the net switches to an alternative roll-back flow (a directed arc from T_2 to ch_{in}) and puts the message back to the message channel ch_{in} . The sequence completion logic is defined depending on a specific pattern application scenario and must always be realized in transition T_4 that executes an update that changes a given sequence state. ■

Through the selection of db-nets as foundation of our formalism, all data and transacted resource requirements are covered (cf. REQ-1, REQ-2, REQ-4 and REQ-5). We call this formalism a *timed db-net* that leverages db-nets [MR17], essentially a database-centric extension of CPNs (incl. atomic transactions), and Timed-arc Petri Nets (TAPNs) [JJMS11].

Data, Transacted Resources and Compensation

The recent work on db-nets by Montali and Rivkin [MR17] strives to combine business processes and transactional data by connecting CPNs with relational databases, separating the (database) persistence layer \mathcal{P} from the PN control layer \mathcal{N} as illustrated in Figure 3.4. This is realized by an intermediate data logic layer \mathcal{P} that mediates between the two by supporting the control layer with queries and database operations (e.g., **trigger**, **update**, **read**, **bind**). The queries bind to database queries by a new type of place, called view place P_v (denoted by \ominus), and read arc, which continuously executes the defined queries. The CPN transitions are extended to execute actions on the database (e.g., insert, update, delete) and additional roll-back arcs (represented as $\leftarrow*$) represent compensation tasks. The subsequent formal introduction of db-nets is based on [MR17].

Definition 3.3 (Db-Net [MR17]). A *db-net* is a tuple $\langle \mathcal{D}, \mathcal{P}, \mathcal{L}, \mathcal{N} \rangle$, where:

- \mathfrak{D} is a type domain — a finite set of data types $D = \langle \Delta_D, \Gamma_D \rangle$, with value domains Δ_D and a finite set Γ_D of predicate symbols;
- \mathcal{P} is a \mathfrak{D} -typed **persistence layer**, i.e., a pair $\langle \mathcal{R}, E \rangle$, where \mathcal{R} is a \mathfrak{D} -typed database schema, and E is a finite set of first-order $\text{FO}(\mathfrak{D})$ constraints over \mathcal{R} ;
- \mathcal{L} is a \mathfrak{D} -typed **data logic layer** over \mathcal{P} , i.e., a pair $\langle Q, A \rangle$, where Q is a finite set of $\text{FO}(\mathfrak{D})$ queries over \mathcal{P} , and A is a finite set of actions over \mathcal{P} ;
- \mathcal{N} is a \mathfrak{D} -typed **control layer** \mathcal{L} , i.e., a tuple $(P, T, F_{in}, F_{out}, \text{color}, \text{query}, \text{guard}, \text{act})$, where:
 - $P = P_c \uplus P_v$ is a finite set of places partitioned into control P_c and view places P_v ;
 - T is a finite set of transitions;
 - F_{in} is an input flow from P to T ;
 - F_{out} and F_{rb} are respectively an output and roll-back flow from T to P_c ;
 - **color** is a color assignment over P (mapping P to a Cartesian product of data types);
 - **query** is a query assignment from P_v to Q (mapping the results of queries in Q as tokens of places P_v);
 - **guard** is a transition guard assignment over T (mapping each transition to a formula over its input inscriptions); and
 - **act** is an action assignment from T to A (mapping transitions to actions triggering updates over the persistence layer). ■

Input and output / roll-back flows contain inscriptions that match the components of colored tokens present in the input and output / roll-back places of a transition. Such inscriptions consist of tuples of (typed) variables, which then can be mentioned in the transition guard as well as in the action assignment (to bind the updates induced by the action to the values chosen to match the inscriptions), and also, in case of the output flow, the inscriptions may contain rigid predicates. Specifically, given a transition t , we denote by $InVars(t)$ the set of variables mentioned in its input flows, by $OutVars(t)$ the set of variables mentioned in its output flows, and by $Vars(t) = InVars(t) \cup OutVars(t)$ the set of variables occurring in the action assignment of t (if any). Fresh variables $FreshVars(t) = OutVars(t) \setminus InVars(t)$ denote those output variables that do not match any corresponding input variables, and are consequently interpreted as external inputs. While input inscriptions are used to match tokens from the input places to $InVars(t)$, the output expressions that involve rigid predicates operate over $OutVars(t)$. In case of numerical types, these expressions can be used to compare values, or to arithmetically operate over them. We call a db-net *plain*, if it employs matching output inscriptions only (i.e., does not use expressions).

Intuitively, each view place is used to expose a portion of the persistence layer in the control layer, so that each token represents one of the answers produced by the query attached to the place. Such tokens are not directly consumed, but only read by transitions, so as to match the input inscriptions with query answers. A transition in the control layer may bind its input inscriptions to the parameters of data logic action attached to the transition itself, thus providing a mechanism to trigger a database update upon transition firing (and consequently indirectly change also the content of view places). If the induced update commits correctly, the transition emits tokens through its output arcs, whereas if the update rolls back, the transition emits tokens through its rollback arcs.

The terms message and (db-net, CPN) token will be used synonymously hereinafter.

Execution Semantics Briefly, the execution semantics of a db-net in [Definition 3.3](#) accounts for the progression of a database instance compliant with the persistence layer \mathcal{P} and the evolution

of a marking over the control layer \mathcal{N} , mediated by the data logic layer \mathcal{L} . The markings of the control layer \mathcal{N} are defined in [Definition 3.4](#).

Definition 3.4 (Marking [\[MR17\]](#)). A *marking* of a \mathcal{D} -typed control layer $(P, T, F_{in}, F_{out}, \text{color}, \text{query}, \text{guard}, \text{action})$ is a function $m : P \rightarrow \Omega_{\mathcal{D}}^{\oplus}$ mapping each place $p \in P$ to a corresponding multiset of p -compatible tuples using data values from \mathcal{D} . A tuple $\langle o_1, \dots, o_n \rangle$ is p -compatible, if $\text{color}(p)$ is of the form $\langle D_1, \dots, D_n \rangle$, and for every $i \in \{1, \dots, n\}$, we have $o_i \in \Delta_{D_i}$, with the value domain Δ_D . Given a database instance I , we say that m is aligned to I via *query*, if the tuples it assigns to view places exactly correspond to the answers of their corresponding queries over I : for every view place $v \in P$ and every v -compatible tuple \vec{o} , we have that $\vec{o} \in m(v)$, if and only if $\vec{o} \in \text{ans}(\text{query}(v), I)$, with the set of answers $\text{ans}(Q, I)$ to a query Q in I .

A marking over the control layer determines the transitions to be fired, and triggers updates of the database instance. In particular, the distributed tokens have to carry data compatible with the color of the places and the marking of a view place P_v must correspond to the associated queries over the underlying database instance. The markings follow the active domain semantics of database systems (i.e., D-active domain, with $D \in \mathfrak{D}$) [\[MR17\]](#). Furthermore, the db-net persistence and control layers are stateful. During the execution, in each moment (called *snapshot*) the persistence layer is associated to a database instance I , and the control layer is associated to a marking m aligned with I via *query* (for what concerns the content of view places). The corresponding snapshot is then simply the pair $\langle I, m \rangle$. The enablement of a db-net transition is specified in [Definition 3.5](#).

Definition 3.5 (Enablement [\[MR17\]](#)). Let B be a db-net $\langle \mathfrak{D}, \mathcal{P}, \mathcal{L}, \mathcal{N} \rangle$. Furthermore, let σ be a binding for t , i.e., a substitution $\sigma : \text{Vars}(t) \rightarrow \Delta_{\mathfrak{D}}$, where $\text{Vars}(t) = \text{InVars}(t) \cup \text{OutVars}(t)$. A transition $t \in T$ is *enabled* in a B -snapshot $\langle I, m \rangle$ with binding σ , if:

- for every place $p \in \mathcal{P}$, $m(p)$ provides enough tokens matching those required by inscription $w = F_{in}(\langle p, t \rangle)$, once w is grounded by σ , i.e., $\sigma^{\oplus}(w) \subseteq m(p)$;
- The instantiated guard $\text{guard}(t)\sigma$ evaluates to true;
- σ is injective over $\text{FreshVars}(t)$, thus guaranteeing that fresh variables are assigned to pairwise distinct values of σ , and for every fresh variable $v \in \text{FreshVars}(t)$, $\sigma(v) \notin (\text{Adom}_{\text{type}(v)}(I) \cup \text{Adom}_{\text{type}(v)}(m))$.¹ ■

Similar to CPNs, the firing of a transition t in a snapshot is defined in [Definition 3.6](#) by a binding that maps the value domains of the different layers, under certain conditions, e.g., the guard attached to t is satisfied.

Definition 3.6 (Firing [\[MR17\]](#)). Let B be a db-net $\langle \mathfrak{D}, \mathcal{P}, \mathcal{L}, \mathcal{N} \rangle$, and $s_1 = \langle I_1, m_1 \rangle, s_2 = \langle I_2, m_2 \rangle$ be two B -snapshots. Fix a transition t of \mathcal{N} and a binding σ such that t is enabled in s_1 with σ (cf. [Definition 3.5](#)). Let $I_3 = \text{apply}(\text{action}_{\sigma}(t), I_1)$ be the database instance resulting from the application of the action attached to t on database instance I_1 with binding σ for the action parameters. For a control place p , let $w_{in}(p, t) = F_{in}(\langle p, t \rangle)$, and $w_{out}(p, t) = F_{out}(\langle p, t \rangle)$ if I_3 is compliant with \mathcal{P} , or $w_{out}(p, t) = F_{rb}(\langle p, t \rangle)$ otherwise. We say that t *fires* in s_1 with binding σ producing s_2 , written $s_1[t, \sigma]s_2$, if:

- if I_3 is compliant with \mathcal{P} , then $I_2 = I_3$, otherwise $I_2 = I_1$;
- for each control place p , m_2 corresponds to m_1 with the following changes: $\sigma^{\oplus}(w_{in}(p, t))$ tokens are removed from p , and $\sigma^{\oplus}(w_{out}(p, t))$ are added to p . In formulae: $m_2(p_c) = (m_1(p_c) - \sigma^{\oplus}(w_{in}(p, t))) + \sigma^{\oplus}(w_{out}(p, t))$ ■

¹ $\text{Adom}_D(X)$ is the set of values of type D explicitly contained in X .

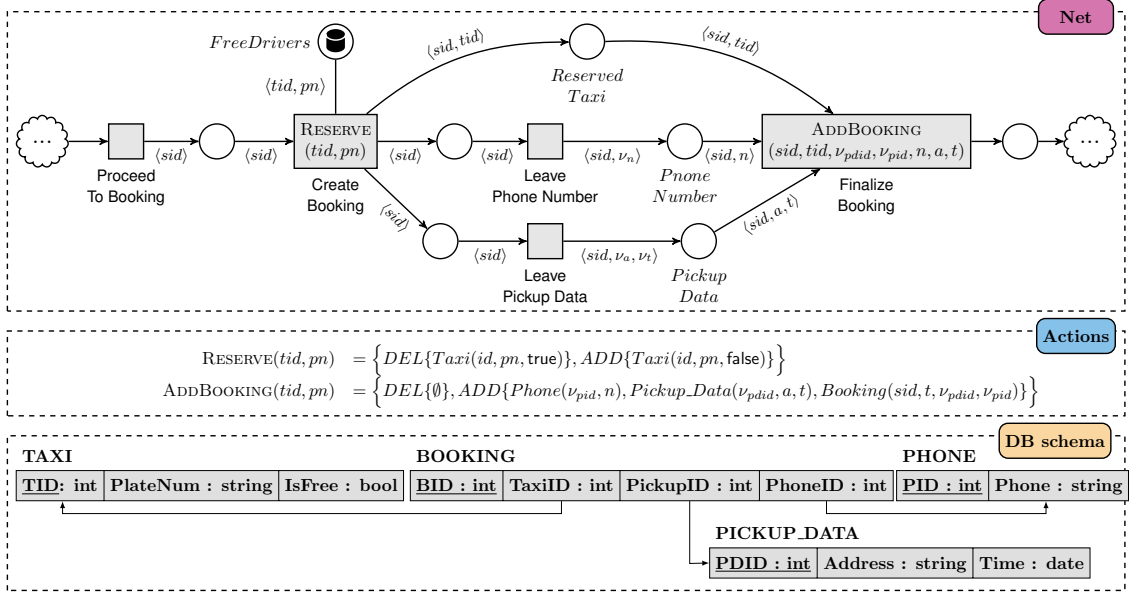


Figure 3.5: Db-net taxi booking process example (similar to [MR17])

This fires the transition, which then has the following effects: all matching tokens in control places P_c are consumed; then the action instance **action** — induced by the firing — is applied on the current database instance in an atomic transaction (and rolled-back, if not successful); and accordingly, tokens on output places F_{out} or roll-back places F_{rb} (i.e., those connected via roll-back flow) are produced. Details are given in [MR17].

Example 3.7. Figure 3.5 denotes a taxi booking process in db-net taken from [MR17] and annotates it with the three db-net layers for a better understanding: the process logic in the control layer, the data logic layer with actions that are executed on the database, and the persistence layer with a full-fledged view on the relational database (incl. constraints). Briefly, the different control places are typed as in a CPN and the only control place *FreeDrivers* accesses the underlying *Taxi* table to check for free taxis (not shown), when the read-arc associated with the *Create Booking* transition is fired. Whenever this transition fires, the *Reserve* action is triggered, which sets the *isFree* property of this taxi to *false* (i.e., the taxi is pre-booked from now on). If no taxi is free, the transition cannot fire. Similarly, when the *Finalize Booking* transition fires, the *AddBooking* action inserts all booking-related data into the corresponding database tables *Phone*, *PickupData*, and *Booking*. Should an action violate database constraints like primary or foreign key relations, the transaction is rolled-back, which can be modeled by adding a roll-back arc (not shown). ■

All in all, the complete execution semantics of a db-net is captured by an infinite-state labeled transition systems (LTS) where each transition represents the firing of a transition in the control layer of the net with a given binding, and each state is a snapshot. Formally, given a db-net $B = \langle \mathcal{D}, \mathcal{P}, \mathcal{L}, \mathcal{N} \rangle$ with $\mathcal{N} = (P, T, F_{in}, F_{out}, \text{color}, \text{query}, \text{guard}, \text{action})$, and an initial snapshot s_0 over B , the LTS $\Gamma_{s_0}^B = \langle S, s_0, \rightarrow \rangle$ starting from s_0 is given by:

- S is a possibly infinite set of B -snapshots;
- $\rightarrow \subseteq S \times T \times S$ is a transition relation over states, labeled by transitions T ;

- S and \rightarrow are defined by simultaneous induction as the smallest sets s.t.
 - $s_0 \in S$;
 - given a B -snapshot $s \in S$, for every transition $t \in T$, binding σ , and B -snapshot s' , if $s[t, \sigma]s'$ then $s' \in S$ and $s \xrightarrow{t} s'$.

Notably, due to the presence of unbounded colors and of the underlying persistence layer, the transition system may contain infinitely many different states even if the control layer is bounded in the classical Petri net sense. However, this infinity can be tamed using faithful abstraction techniques [MR17].

While the following example gives a first intuition on our solution, the different aspects of timed db-net will be explained subsequently by the Aggregator as running example.

Example 3.8. The Aggregator in Figure 3.3 (on page 68) is naturally implemented using a view place ch_p (denoted by \ominus) for storing and updating the message sequences as well as roll-back arc $(T2, ch_{in})$ to manage compensation tasks (represented as $\leftarrow*$). The graphical notation is in line with [MR17]. ■

Extending Db-Nets with Time

We now extend the db-net model so as to account for an explicit notion of time. In the spectrum of timed extensions to PNs, we subsequently extend the db-net control layer \mathcal{N} with a temporal semantics similar to TAPNs that achieves a suitable trade-off: it is expressive enough to capture the requirements in REQ-3, and at the same time it allows us to transfer the existing technical results on the verification of db-nets to the timed extension.

The Timed-arc Petri Nets (TAPNs) by Bolognesi et al. and Hanish [Bol90, Han93] denote an extension of classical P/T nets with continuous time. Other temporal models like timed transitions add time durations for delayed firing (e.g., Ramchandani [Ram73] and Zuberek et al. [Zub87]) or Time petri nets (TPNs) that associate transitions with time intervals denoting earliest and latest firing time of transition after enablement (e.g., Merlin et al. [Mer74, MF76]). In contrast, in TAPNs not the transitions, but tokens carry temporal information in the form of an *age* value. While in a place, the tokens age and the arcs between places and transitions are labeled with time intervals that enable transitions, when the age is within the interval. The subsequent, formal introduction of TAPN syntax and execution semantics is mostly based on the work of Jacobsen et al. [JJMS11].

Let I be a set of well-formed intervals

$$I \subseteq \{[a, b] \mid a, b \in \mathbb{R}, b \geq a\} \cup \{[a, b) \mid a, b \in \mathbb{R}, b > a\} \cup \{(a, b] \mid a, b \in \mathbb{R}, b > a\} \\ \cup \{[a, \infty) \mid a \in \mathbb{R}\} \cup \{(a, \infty) \mid a \in \mathbb{R}\}.$$

A predicate $r \in Int$ is defined for $r \in \mathbb{R}_{\geq 0}$ and $Int \in I$.

Definition 3.9 (TAPN [JJMS11]). A TAPN is a 5-tuple $\langle P, T, IA, OA, Inv \rangle$, where:

- P is a finite set of places;
- T is a finite set of transitions s.t. $P \cap T = \emptyset$;
- $IA \subseteq T \times \mathcal{I} \times P$ is a finite set of input arcs s.t.

$$((p, Int, t) \in IA \wedge (p, Int', t) \in IA) \implies Int = Int';$$

- $OA \subseteq T \times P$ is a finite set of output arcs;

- $Inv : P \rightarrow T^{inv}$ is a function assigning age invariants to places. ■

Note that multiple parallel arcs are not allowed according to [JJMS11]. The preset of a transition $t \in T$ is specified as $\bullet t = \{p \in P \mid (p, Int, t) \in IA\}$, and the postset is $t^\bullet = \{p \in P \mid (t, p) \in OA\}$.

Execution Semantics The TAPN semantics for enabledness and firing require the definition of a marking M (cf. Definition 3.10), which is a function assigning a finite multiset of nonnegative real numbers to each place (all such finite multisets are denoted by $\mathbb{R}_{\geq 0}$). The real numbers represent the age of tokens at a given place. Moreover, the age of every token must respect the age invariant of the place where the token is located.

Definition 3.10 (Marking [JJMS11]). Let $N = (P, T, IA, OA, Inv)$ be a TAPN. A marking M on N is a function $M : P \rightarrow \mathbb{R}_{\geq 0}$, where for every place $p \in P$ and every token $x \in M(p)$ we have $x \in Inv(p)$. The set of all markings over N is denoted by $M(N)$. ■

For simplicity, we use the notation (p, x) to refer to a token in the place p of age $x \in \mathbb{R}_{\geq 0}$. The multiset of a marking with n tokens located in places p_i and age x_i , for $1 \leq i \leq n$ is denoted by $M = \{(p_1, x_1), (p_2, x_2), \dots, (p_n, x_n)\}$. A marked TAPN is a pair (N, M_0) where N is a TAPN and M_0 is an initial marking on N where all tokens have age 0. With that, a TAPN transition is enabled as specified by Definition 3.11, when for all input arcs there is a token in the input place with an age satisfying the age guard of the respective arc. The age of the output token is 0 for all output arcs.

Definition 3.11 (Enabledness [JJMS11]). Let $N = (P, T, IA, OA, Inv)$ be a TAPN. A transition $t \in T$ is enabled in a marking M by tokens $In = \{(p, x_p) \mid p \in \bullet t\} \subseteq M$ and $Out = \{(p', x_{p'}) \mid p' \in t^\bullet\}$ if there is $(p, Int, t) \in I$ with $x_p \in Int$. ■

On enablement, a TAPN transition fires according to Definition 3.12, which essentially denotes an immediate firing.

Definition 3.12 (Firing [JJMS11]). Let $N = (P, T, IA, OA, Inv)$ be a TAPN, M a marking on N and $t \in T$ a transition. If t is enabled in the marking M by tokens In and Out , then it can fire and produce a marking M' defined as

$$M' = (M \setminus In) \cup Out,$$

where \setminus and \cup are operations on multisets. ■

The actual time delay is introduced by Definition 3.13.

Definition 3.13 (Time delay [JJMS11]). Let $N = (P, T, IA, OA, Inv)$ be a TAPN, M a marking on N . A time delay $d \in \mathbb{R}_{\geq 0}$ is allowed in M , if $(x + d) \in Inv(p)$ for all $p \in P$ and all $x \in M(p)$, i.e., by delaying d time units no token violates any of the age invariants. When delaying d time units in M , we reach a marking M' defined as

$$M'(p) = \{x + d \mid x \in M(p)\}$$

for all $p \in P$. ■

Example 3.14. Figure 3.6(a) illustrates the syntax and execution semantics of a basic TAPN. The tokens (*age*) in the places p_1 – p_3 carry an *age* information that is set to 0.0, when entering a place. Further the corresponding arcs are annotated by time intervals *Int* restricting the age of

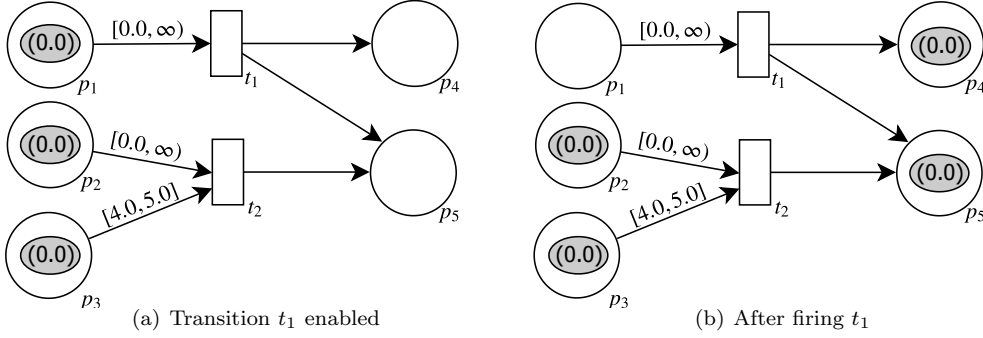


Figure 3.6: Timed-arc Petri net example

tokens available for transition firing. Initially, only transition t_1 is enabled due to the implicit default time interval $[0.0, \infty)$. That allows t_1 to fire and move the token from p_1 to the places p_4 and p_5 by resetting their age to 0.0, shown in Figure 3.6(b). The transition t_2 remains disabled for at least four time units of 1.0 due to time interval $[4.0, 5.0]$ on its second input arc, until it becomes enabled and fires. ■

A general *timed transition system* (TTS), according to [JJMS11], is a pair $T = (S, \longrightarrow)$, where S is a set of states (or processes) and $\longrightarrow \subseteq (S \times S) \cup (S \times \mathbb{R}_{\geq 0} \times S)$ is a transition relation. For discrete transitions $(s, s') \in \longrightarrow$ we write $s \longrightarrow s'$ and $s \xrightarrow{d} s'$ for delay transitions (s, d, s') , for which we write $s \xRightarrow{d} s'$. A TTS should satisfy all of the standard axioms (e.g., time additivity) for delay transitions from [BPV06]. Now, a TAPN N defines a timed transition system $(\mathcal{M}(N), \longrightarrow)$, where states are markings of N , and for two markings M and M' we have $M \xrightarrow{d} M'$, if by delaying d time units in M we reach the marking M' . Thereby \mathcal{N} , and $\mathbb{R}_{\geq 0}$ denote the sets of non-negative integers and non-negative real numbers, respectively. The marking M' is reachable from marking M , if $M \xRightarrow{*} M'$, with $\xRightarrow{*}$ denoting the reflexive and transitive closure of \xRightarrow{d} .

Based on the introduced db-nets and TAPNs, we start by explaining the intuition behind the approach, and then provide the corresponding formalization. We assume that there is a global, continuous notion of time. The firing of a transition is instantaneous, but can only occur in certain moments of time, while it is inhibited in others, even in presence of the required input tokens. Every *control token*, that is, token assigned to a control place, carries a (local) *age*, indicating how much time the token is spending in that control place. This means that when a token enters into a place, it is assigned an age of 0. The age then increments as the time flows and the token stays in the same place. View places continuously access the underlying persistence layer, and consequently their (virtual) tokens do not age. Each transition is assigned to a pair of non-negative (possibly identical) rational numbers, respectively describing the minimum and maximum age that input tokens should have when they are selected for firing the transition. Thus, such numbers identify a relative time window that expresses a delay and a deadline on the possibility of firing.

Definition 3.15. A *timed db-net* is a tuple $\langle \mathfrak{D}, \mathcal{P}, \mathcal{L}, \mathcal{N}, \tau \rangle$ where $\langle \mathfrak{D}, \mathcal{P}, \mathcal{L}, \mathcal{N} \rangle$ is a db-net with transitions T , and $\tau : T \rightarrow \mathbb{R}_{\geq 0} \times (\mathbb{R}_{\geq 0} \cup \{\infty\})$ is a *timed transition guard* mapping each transition $t \in T$ to a pair of values $\tau(t) = \langle v_1, v_2 \rangle$, such that:

- v_1 is a non-negative rational number;

- v_2 is either a non-negative rational number equal or greater than v_1 , or the special constant ∞ .

The default choice for τ is to map transitions to the pair $\langle 0, \infty \rangle$, which corresponds to a standard db-net transition.

Given a transition t , we adopt the following graphical conventions:

- if $\tau(t) = \langle 0, \infty \rangle$, then no temporal label is shown for t ;
- if $\tau(t)$ is of the form $\langle v, v \rangle$, we attach label “@ $\langle v \rangle$ ” to t ;
- if $\tau(t)$ is of the form $\langle v_1, v_2 \rangle$ with $v_1 \neq v_2$, we attach label “@ $\langle v_1, v_2 \rangle$ ” to t .

Example 3.16. The Aggregator in Figure 3.3 (on page 68) defines a timed transition T_3 , that can be fired precisely after 30 time units (here seconds) from the moment when a new sequence *seq* has been created. Upon firing, T_3 enables the *Aggregate* transition, by updating the sequence’s status on the database to *expired* using the *TimeoutSeq* action. ■

Execution semantics The execution semantics of timed db-net builds on the one for standard db-nets, extended with additional conditions on the flow of time and the temporal enablement of transitions in Definition 3.17. The management of bindings, guards, and database updates via actions, is kept unaltered. What changes is that, in a snapshot, each token now comes with a corresponding age, represented as a number in $\mathbb{R}_{\geq 0}$. More formally, a marking is now a function $m : P \rightarrow \Omega_D^\oplus$, where Ω_D^\oplus consists of multisets of tuples $\langle y, o \rangle$ with $y \in \mathbb{R}_{\geq 0}$ and $o \in \Delta_D$.

Definition 3.17 (Transition Enablement). Let B be a timed db-net $\langle \mathcal{D}, \mathcal{P}, \mathcal{L}, \mathcal{N}, \tau \rangle$, and t a transition in \mathcal{N} with $\tau(t) = \langle v_1, v_2 \rangle$. Let σ be a binding for t , i.e., a substitution $\sigma : \text{Vars}(t) \rightarrow \Delta_D$. A transition $t \in T$ is *enabled* in a B -snapshot $\langle I, m \rangle$ with binding σ , if:

- For every place $p \in \mathcal{P}$, $m(p)$ provides enough tokens matching those required by inscription $w = F_{in}(\langle p, t \rangle)$, once w is grounded by σ , i.e., $\sigma \subseteq m(p)$;
- the guard $\text{guard}(t)\sigma$ evaluates to true;
- for every $x_a \in \text{InVars}(t)$, $v_1 \leq \sigma(x_a) \leq v_2$, where $\tau(t) = \langle v_1, v_2 \rangle$;
- σ is injective over $\text{FreshVars}(t)$, thus guaranteeing that fresh variables are assigned to pairwise distinct values of σ , and for every fresh variable $v \in \text{FreshVars}(t)$, $\sigma(v) \notin (\text{Adom}_{\text{type}(v)}(I) \cup \text{Adom}_{\text{type}(v)}(m))$.²
- For each age variable $y_a \in (\text{OutVars}(t) \setminus \text{InVars}(t))$, we have that $\sigma(y_a) = 0$ (i.e., newly produced tokens get an age of 0). ■

As customary in several temporal extensions of Petri nets, we consider two types of evolution steps. The first type deals with *time lapses*: it indicates that a certain amount of time has elapsed with the net being quiescent, i.e., not firing any transition. This results in incrementing the age of all tokens according to the specified amount of time.

The second type deals with transition firing, which refines that of db-nets by checking that the chosen binding selects tokens whose corresponding ages are within the delay window attached to the transition (cf. Definition 3.17). Specifically, let B be a timed db-net $\langle \mathcal{D}, \mathcal{P}, \mathcal{L}, \mathcal{N}, \tau \rangle$, t a transition in \mathcal{N} with $\tau(t) = \langle v_1, v_2 \rangle$, and σ a binding for t . We say that t is enabled in a given B snapshot with binding σ if it is so according to Definition 3.5 (on page 71) and, in addition, all the tokens selected by σ have an age that is between v_1 and v_2 . Firing an enabled transition is identical to the case of standard db-nets (cf. Definition 3.6), with the only addition that for each produced token, its age is set to 0 (properly reconstructing the fact that it is entering into the corresponding place).

² $\text{Adom}_D(X)$ is the set of values of type D explicitly contained in X ;

The execution semantics of a timed db-net then follows the standard construction (using the refined notions of enablement and firing), with the addition that each snapshot may be subject to an arbitrary time lapse. This is done by imposing that every B -snapshot $\langle I, m \rangle$ is connected to every B -snapshot of the form $\langle I', m' \rangle$ where:

- $I' = I$ (i.e., the database instances are identical);
- m' is identical to m except for the ages of tokens, which all get incremented by the same, fixed amount $x \in \mathbb{Q}$ of time.

Given two B -snapshots s and s' , we say that s *directly leads* to s' , written $s \rightarrow s'$, if there exists a direct transition from s to s' in the transition system that captures the execution semantics of B . This means that s' results from s because of a transition firing or a certain time lapse. We extend this notion to finite execution traces $s_0 \rightarrow \dots \rightarrow s_n$. We also write $s \xrightarrow{*} s'$ if s directly or indirectly leads to s' . If this is the case, we say that s' *reachable* from s .

Example 3.18. To complete the Aggregator, when the persisted sequence in the Aggregator is complete or the sequence times out, the enabled *Aggregate* transition fires by reading the sequence number seq and snapshot of the sequence messages, and moving an aggregate msg' to ch_{out} . Notably, the *Aggregate* transition is invariant to which of the two causes led to the completion of the sequence. ■

Checking Reachability over Timed Db-nets

Checking fundamental correctness properties such as *safety* / *reachability* is of particular importance for timed db-nets, in the light of the subsequent correctness testing discussions in Sections 3.1.2 and 3.1.3 on reachable goal states. We consider here, in particular, the following relevant *reach-template* problem:

Input: (i) a timed db-net B with set P_c of control places, (ii) an initial B -snapshot s_0 , (iii) a set $P_{empty} \subseteq P_c$ of *empty control places*, (iv) a set $P_{filled} \subseteq P_c$ of *nonempty control places* such that $P_{empty} \cap P_{filled} = \emptyset$.

Output: *yes* if and only if there exists a finite sequence of B -snapshots of the form $s_0 \rightarrow \dots \rightarrow s_n = \langle I_n, m_n \rangle$ such that for every place $p_e \in P_{empty}$, we have $|m_n(p_e)| = 0$, and for every place $p_f \in P_{filled}$, we have $|m_n(p_f)| > 0$.

Checking the emptiness of places in the target snapshot is especially relevant in the presence of timed transitions, so as to predicate over runs of the systems where tokens are consumed within the corresponding temporal guards. For example, by considering transition $T3$ in Figure 3.3 (on page 68), asking for the ch_{timer} place to be empty guarantees that $T3$ is indeed triggered whenever enabled.

Since timed db-nets build on db-nets, reachability is highly undecidable, even for nets that do not employ timed transitions, have empty data logic and persistence layers, and only employ simple string colors. As pointed out in [MR17], this setting is in fact already expressive enough to capture ν -nets [Las16, RVdFE11], for which reachability is undecidable. Similar undecidability results can be obtained by restricting the control layer even more, but allowing for the insertion and deletion of arbitrarily many tuples in the underlying persistence layer.

However, when controlling the size of information maintained by the control and persistence layers in each single snapshot, reachability and also more sophisticated forms of temporal model checking become decidable for db-nets using string and real data types (without arithmetics) [MR17].

In particular, decidability has been shown for *bounded*, *plain* db-nets. Technically, a db-net B with initial snapshot s_0 is:

- **width-bounded** if there is $b \in \mathbb{N}$ s.t., for every B -snapshot $\langle I, m \rangle$, if $s_0 \xrightarrow{*} \langle I, m \rangle$, then the number of distinct data values assigned by m to the tokens residing in the places of B is bounded by b ;
- **depth-bounded** if there is $b \in \mathbb{N}$ s.t., for every B -snapshot $\langle I, m \rangle$, if $s_0 \xrightarrow{*} \langle I, m \rangle$, then the number of appearances of each distinct token assigned by m to the places of B is bounded by b ;
- **state-bounded** if there is $b \in \mathbb{N}$ s.t., for every B -snapshot $\langle I, m \rangle$, if $s_0 \xrightarrow{*} \langle I, m \rangle$, we have $|\cup_{D \in \mathfrak{D}} \text{Adom}_D(I)| \leq b$.

We say that a db-net is *bounded*, if it is at once width-, depth-, and state-bounded. Intuitively, a db-net is bounded if it does not accumulate unboundedly many tokens in a place, and guarantees that the number of data objects used in each database instance does not exceed a pre-defined bound. Additionally, for bounded timed db-nets, we assume *time-boundedness* over the fixed timed transitions (i.e., fixed time intervals), and thus restricting the ages in the tokens.

The decidability of reachability for bounded db-nets does not imply decidability of reachability for bounded timed db-nets. In fact, ages in timed db-nets are subject to comparison and (global) increment operations that are not expressible in db-nets. However, we can prove decidability by resorting to a *separation* argument: the two dimensions of infinity respectively related to the infinity of the data domains and of the flow of time can in fact be tamed orthogonally to each other. In particular, we get the following.

Theorem 3.19. The *reach-template* problem is decidable for bounded and plain timed db-nets with initial snapshot.

Proof sketch. Consider a bounded timed db-net B with initial snapshot s_0 , empty control places P_{empty} , and filled control places P_{filled} . Using the faithful data abstraction techniques presented in [MR17, Thm 2], one obtains a corresponding timed db-net B' enjoying two key properties.

First, B' is bisimilar to B , with a data-aware notion of bisimulation that takes into account both the dynamics induced by the timed db-net, as well as the correspondence between data elements. Such a notion of bisimulation captures reachability as defined above, and consequently *reach-template*($B, s_0, P_{\text{empty}}, P_{\text{filled}}$) returns *yes* if and only if *reach-template*($B', s_0, P_{\text{empty}}, P_{\text{filled}}$) does so.

Second, the only source of infinity, when characterizing the execution semantics of B' , comes from the temporal aspects, and in particular the unboundedness of token ages. This means that B' can be considered as a “standard” temporal variant of a CPN with bounded colors that, in turn, boils down to a temporal variant of an (uncolored) P/T net. In particular, one can easily see that B' corresponds to a specific type of bounded TAPN [JJMS11], where:

- whenever B' contains a transition t with $\tau(t) = \langle v_1, v_2 \rangle$, its corresponding TAPN labels each arc entering in t with the same interval $[v_1, v_2]$;
- each transition-place arc is labeled with the interval $[0, 0]$.

Consequently, the infinity of B' can be tamed using standard techniques known for *bounded* TAPNs, which indeed enjoy decidability of reachability for the queries tackled by *reach-template* [BD91, AGH16]. In particular, notice that *reach-template* does not explicitly express constraints on the expected token ages when reaching the final state. \square

It is interesting to notice that TAPNs have a more expressive mechanism to specify temporal guards in the net (cf. [JJMS11]). In fact, TAPNs attach temporal guards to arcs, not transitions, and can therefore express different age requirements for different places, as well as produce

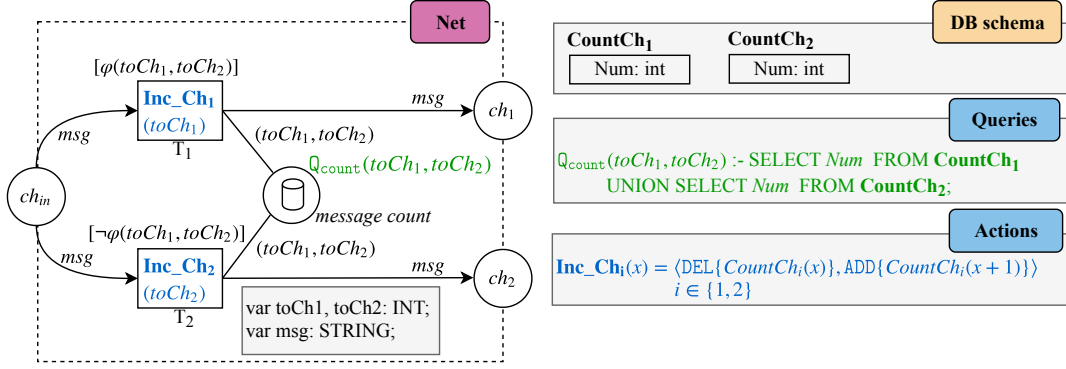


Figure 3.7: Load balancer realization as a timed db-net

tokens with an age nondeterministically picked from a specified interval. Hence, this more refined temporal semantics can be seamlessly introduced in our timed db-net model without compromising Theorem 3.19.

Pattern Realizations

A pattern realization denotes a representation of a pattern as a timed db-net (e.g., the Aggregator in Figure 3.3, on page 68). In this section we discuss (formal) pattern realizations using timed db-nets. Due to the high number of patterns, the formalization and corresponding in-detail description of all of them seems impractical. However, thanks to the fact that patterns can be classified into disjoint categories (see the requirement categories in Section 3.1.1), it suffices to discuss the most representative ones from each of such categories that serve as examples on how to construct the others. We call this an *instructive pattern formalization*, which strives to formalize the patterns and, at the same time, offers modeling guidelines for other patterns of the respective categories using the provided examples.

Control Flow: Load Balancer To demonstrate the *control flow only* pattern (cf. REQ-0 in Table 3.1, on page 67), we have chosen the Load Balancer pattern (cf. Chapter 2). Interestingly, this pattern also covers the message channel distribution requirement (cf. REQ-1(b) in Table 3.1) and thus can be considered as a relevant candidate of this category as well.

In a nutshell, the balancer distributes the incoming messages to a number of receivers based on a criterion that uses some probability distribution or ratio defined on the sent messages. To realize the former one could resort to stochastic PNs [Zen85, Bal01] or extend the db-net transition guards definition with an ability to sample probability values from a probability distribution (e.g., [HSS07]). While the latter would extend the db-net persistence layer, it is unclear whether the decidability results discussed in the previous section will still hold. Hence, we opted for the ratio criterion that, as shown in Figure 3.7, is realized using a persistence storage and transition guards with a simple balancing scheme. Specifically, a message msg in channel ch_{in} leads to a lookup of the current ratio by accessing the current message counts per output channel in the database and evaluating guards assigned to one of the two transitions based on the extracted values. The ratio criterion is set up with two (generic) guards $\varphi(toCh_1, toCh_2)$ and $\neg\varphi(toCh_1, toCh_2)$ respectively assigned to T_1 and T_2 . If one of the guards holds, the corresponding transition fires by moving the message to its output place as well as updating the table by incrementing the corresponding channel count. The latter is done by consecutively performing $Inc_Ch_i \cdot del = \{CountCh_i(x)\}$

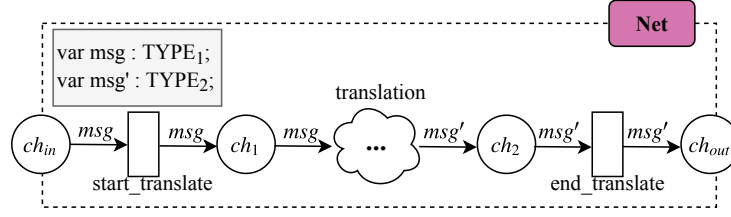


Figure 3.8: Message translator realization as a timed db-net

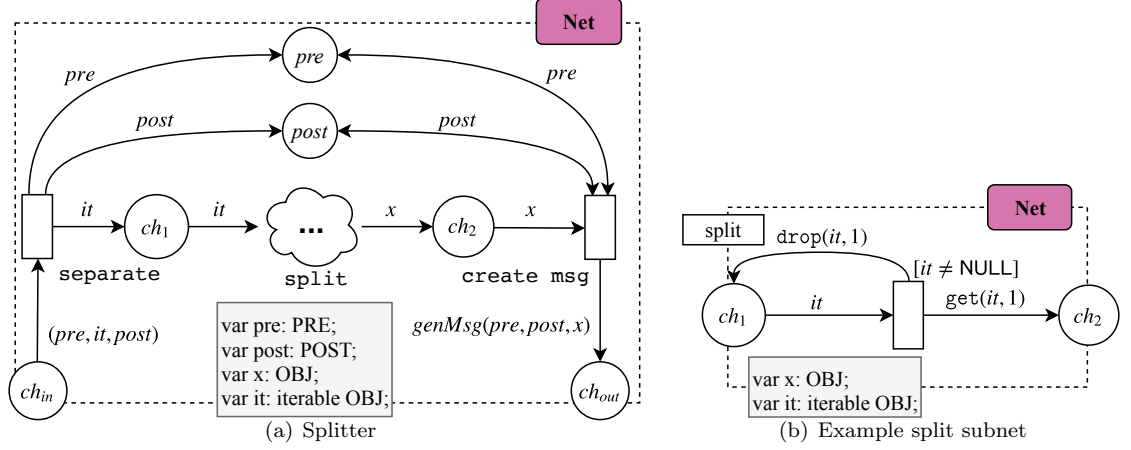


Figure 3.9: Splitter realization and sample *split* subnet realization as a timed db-net

and $Inc_Ch_i \cdot \text{add} = \{NumberCh_i(x + 1)\}$ (for $i \in \{1, 2\}$).

Data Flow: Message Translator, Splitter The stateless Message Translator, shown in Figure 3.8, is the canonical example of a *data flow only* pattern (cf. [HW04]). The translator works on the data representation level, by transforming an incoming message of type $TYPE_1$ from ch_{in} using a subnet that starts by firing *start_translate* and finishes by firing *end_translate*, and that produces a new message of type $TYPE_2$ into the receiver place ch_{out} . Note that for the representation of a subnet we use a cloud symbol, which denotes a configurable model part in the form of a subnet.

The (iterative) Splitter is an example of a non-message transformation *data flow only* pattern, which is also required for a case study scenario in Section 3.1.3. The pattern itself represents a complex routing mechanism that, given an iterable collection of input messages *it* together with two objects *pre* and *post*, is able to construct for each of its elements a new message of the form $[\langle pre \rangle][it : msg_i][\langle post \rangle]$ with optional *pre* and *post* parts. As shown in Figure 3.9(a), the splitter can be fully realized in CPN under certain restrictions assumed for the type of the iterable collection at hand. The entering message payloads in ch_0 are separated into its parts: *pre*, *post* and *it*. While the first two are remembered during the processing, the iterable *it* is iteratively split into parts according to some criterion realized in the *split* subnet, which represents a custom split logic and thus is intentionally left unspecified (indicated by a cloud symbol in Figure 3.9(a)).

Example 3.20. The *split* subnet can be adapted to the message format and the requirements

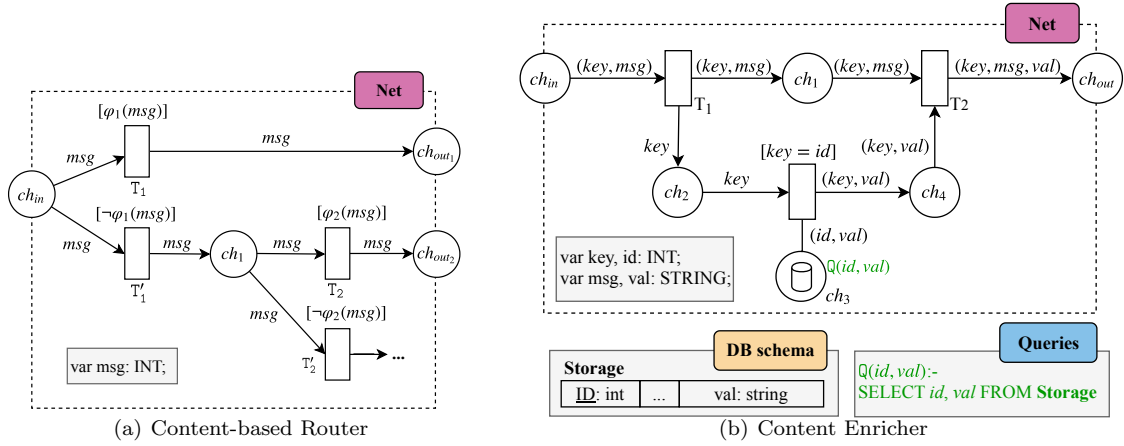


Figure 3.10: Content-based router and content enricher realization as a timed db-net

of a specific scenario. Figure 3.9(b) demonstrates a possible implementation of the subnet. Here, functions *get* and *drop* are used to read and remove the n -th element of an iterable object. In our case, we alternate their applications to the *iterable* object *it* from place ch_1 in order to extract and delete its first element that is then placed into ch_2 . Such a procedure is repeated until *it* is empty (i.e., it is NULL). ■

Each of extracted elements from *it*, together with the information about *pre* and *post*, is then used to create a new message (by calling function *genMsg*) that is passed to the output channel ch_{out} (Figure 3.9(b)).

Data and Control Flow: Content-based Router The Content-based Router pattern is the canonical candidate for *data and control flow* patterns. The realization of the router with conditions φ_1, φ_2 that have to be evaluated strictly in-order (cf. REQ-1(a)) is shown in Figure 3.10(a). Although the router could be realized more elegantly by using priority functions similarly to [Zen85], we explicitly realized them with pair-wise negated timed db-net transition guards. A message from incoming channel ch_{in} is first evaluated against condition φ_1 . Based on the evaluation result, the message is moved either to ch_1 or ch_2 . In case it has been moved to ch_2 , the net proceeds with the subsequent evaluation of other conditions using the same pattern. When none of the guards can be evaluated, a non-guarded, low-priority default transition fires (not shown). This explicit realization covers the router’s semantics, however, requires $(k \times 2) + 1$ transitions (i.e., condition, negation, and only one default), with the number of conditions k .

Data Flow with Transacted Resources: Content Enricher, Resequencer The first pattern chosen for this category such that it includes a data flow with *transacted resources* is the Content Enricher. It requires accessing external resources (e.g., relational database) based on *data* from an incoming message. Such data are then used to enrich the content of the message. As one can see in Figure 3.10(b), the pattern uses request-reply transitions T_1 and T_2 to direct the net flow towards extracting message-relevant data from an external resource. The extraction is performed by matching a message identifier *key* with the one in the storage. While the stateless enriching part is essentially a coloured Petri net, in order to access a stateful resource in ch_3

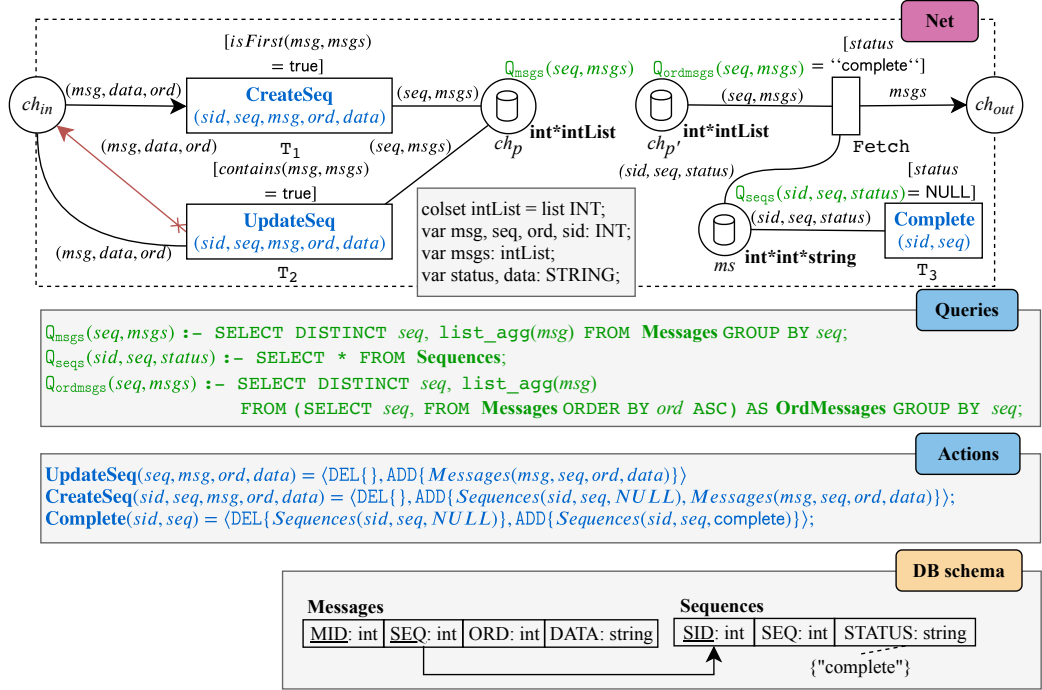


Figure 3.11: Resequencer realization as a timed db-net

one requires to use db-nets so as to specify and perform queries on the external storage (cf. REQ-4(a,b)).

The stateful Resequencer is a pattern that ensures a certain order imposed on messages in (asynchronous) communication [HW04]. Figure 3.11 shows how the resequencer can be represented in db-nets. The incoming message msg contains information about its sequence seq , some order information ord and is persisted in the database. The information about stored messages can be accessed through the view place ch_p . For the first incoming message in a sequence, a corresponding sequence entry with a unique identifier value bound to sid ³ will be created in the persistent storage (sequences can be accessed in the view place ms), whereas for all subsequent messages of the same sequence, the messages are simply stored. As soon as the sequence is complete, i.e., all messages of that sequence have arrived, the messages of this sequence are queried from the database in ascending order of their ord component (see the view place $ch_{p'}$ and its corresponding query). The query result is represented as a list that is forwarded to ch_{out} . Note that, similarly to the Aggregator in Figure 3.3, the completion condition can be extended by a custom logic in T_3 (e.g., a condition on the number of message to be aggregated).

Control Flow with Transacted Resource and Time: Circuit Breaker To demonstrate a family of patterns that are based on a control flow with transacted resources and time, we selected as its representative the Circuit Breaker pattern (see Chapter 2). It addresses failing or hang up remote communication, which impacts the control flow of the Request-Reply pattern by using transacted access to external resources. Figure 3.12 shows a representation of the request-reply

³Note that since sid is not bound to variables in the input flow of T_i ($i \in \{1, 2, 3\}$), it can be treated as a fresh variable that, whenever the transition is executed, gets a unique value of a corresponding type assigned to it.

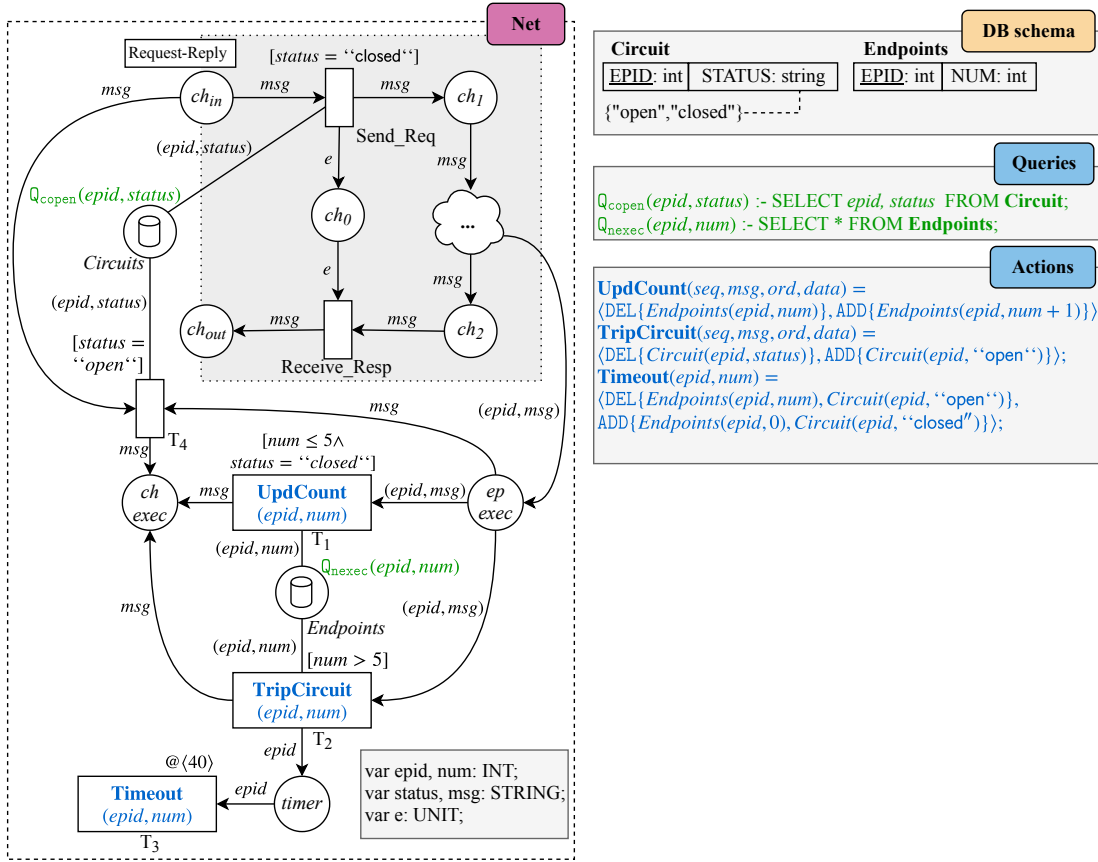


Figure 3.12: Circuit breaker realization as a timed db-net

pattern in timed db-nets, extended by a circuit breaker “wrapper” that protects the remote call. At the beginning, every (endpoint-dedicated) circuit⁴ in the circuit breaker is closed (that is, its status in table *Circuit* is initially set to *closed*), thus allowing for the communication via the Request-Reply pattern part. If the request-reply pattern executes normally, the resulting message is placed in *ch_{out}*. Otherwise, in the case when an exception has been raised, the information about the failed endpoint is stored both in the *Endpoints* table of the persistent storage and a special place *ch_{exec}*. Such a table contains all endpoints together with the number of failures that happened at them. If the number of failures reaches a certain limit (e.g., $num > 5$), the circuit trips and updates its status in the corresponding entry of the *Circuit* relation to *open*. This in turn immediately blocks the communication process that, however, can be resumed (i.e., the circuit is again set to *open* and the failure count is set to 0) after 40 time units have been passed. Note that whenever at least one circuit remains closed, the messages from *ch_{in}* will be immediately redirected to *ch_{exec}*.

Control Flow with Time: Throttler, Delayer The representative patterns of this group mostly require control flow and time aspects, and thus can be represented using timed CPNs. The first pattern is the Throttler. It helps to ensure that a specific receiver does not get overloaded by

⁴For simplicity, every endpoint is identified with a unique number EPID.

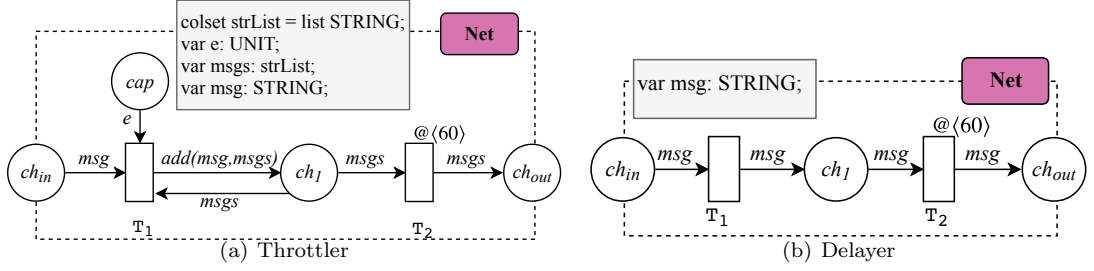


Figure 3.13: Throttler and delayer realizations as a timed db-net

regulating the number of transferred messages. Figure 3.13(a) shows the realization of a throttler that emits at most n messages (here n is the number of “simple”, black tokens assigned with an initial marking to place cap) per 60 time units to the receiving place ch_{out} .

A slightly different pattern of this category is the Delayer that uses a timer to reduce the frequency of messages sent to the receiving place ch_{out} . As shown in Figure 3.13(b), the sent message is delayed by 60 time units.

Data Flow with transacted Resources Time: Aggregator The combination of data, transacted resources and time aspects in patterns makes them the semantically most complex ones. For example, Figure 3.3 (on page 68) specifies the semantics of a commonly used stateful Aggregator pattern. The Aggregator persistently collects messages, that can be seen in a dedicated view place ch_p , and aggregates them using the *Aggregate* transition based on a completion condition (i.e., a sequence that the message is related to is complete) or on a sequence completion timeout. For this an incoming message msg is correlated to an existing sequence based on a sequence label seq attached to it. If the message is first in a sequence, a new sequence is created in the *Sequences* table and the message together with a reference to this sequence is recorded in the persistent storage using the *Messages* table. If a message correlates to an existing sequence seq , which has been aggregated due to a timeout, the update fails. This results in the roll-back behavior: the database instance is restored to the previous, while the net uses the roll-back arc to put the message back to the message channel ch_{in} . This message can be then added as the first one to another newly created sequence seq .

Discussion The db-net foundation implicitly covers REQs-2,4 in the form of a relational formalization with database transactions. Together with the realizations of the Content-based Router, Load Balancer (cf. REQ-1(a), (b)) and Aggregator in Figure 3.3 (cf. REQs-3(a), REQ-4 and REQ-5) we showed realizations for all of the requirements from Section 3.1.1. The expiry of tokens, depending on time information within the message, can be represented using CPNs and db-nets by modeling it as part of the token’s color set and transition guards (similar to [vdA93]). Nevertheless, to model the transition timeouts (cf. REQ-3(a)) and delays (cf. REQ-3(c)) one needs to resort to more refined functionality realized in timed transitions provided by the timed extension of db-nets. Similarly, the message / time ratio (cf. REQ-3(d)) can be represented (see Throttler pattern in Figure 3.13(a)).

The categorization of patterns according to their characteristics allows for an *instructive* formalization based on candidates of these categories and shows that even complex patterns can be defined in timed db-nets. This, in turn, allowed us not to discuss candidates of all the

categories from Figure 3.2 (on page 66), since they can be seamlessly derived by the introduced patterns from other categories. For example, *control* and *data* with resource patterns do not require transacted resources, and can thus be realized similar to their transacted resource cases by substituting view places with normal ones. The building blocks for the realization of *transacted resource* as well as *data flow with time* patterns can be derived from, e.g., the Resequencer or Aggregator patterns. Finally, the *data flow with format* patterns can be represented using CPNs, and thus are not further discussed here.

Thanks to our model checking result presented in the previous section and those derived from db-net (e.g., liveness property from [MR17][Theorem 2] based on the argument on μ -calculus for data-centric dynamic systems [MR16] that enjoy a liveness property due to proof in [BHCDG⁺13]), the correctness of the realization of each pattern can be formally verified. However, due to the absence of a model checker for (timed) db-nets, the formal analysis (cf. [MR17]) of such cannot currently be automatically performed. Nevertheless, as an alternative to the model checking approach, it is possible to perform the correctness testing using the experimental validation via (repeated) simulation of db-net models. We discuss this approach in the following section.

Correctness Testing

The correctness of an integration pattern realization represented in timed db-nets can be validated by evaluating the execution traces of such models (e.g., similar to the *state-oriented* testing scheme by Zu and He [ZH02]), where at each step, an execution trace contains a *B*-snapshot representing a current state of the persistence layer together with a control layer marking. According to the timed db-net execution semantics (see Section 3.1.2), a consecutive, finite enactment of a pattern model starting from an initial *B*-snapshot $s_1 = \langle I_1, m_1 \rangle$ produces several *B*-snapshots $s = \langle I, m \rangle$ that, depending on the number of enactment steps, generates a finite execution trace $s_1 \rightarrow \dots \rightarrow s_{n+1}$ for some $n \in \mathbb{N}$.

Example 3.21. Consider a Content-based Router model *B* from Figure 3.10(a) (on page 81) with an initial *B*-snapshot s_1 containing markings for messages of two employees with their age (Jane, 23), and (Paul, 65). Since the router has no persistence, the database snapshot is empty. Figure 3.14(a) shows a possible, finite execution of *B* starting from s_1 . In order to reach s_2 from s_1 , transition T_1 with routing condition $[age \leq 25]$ has to be fired. The resulting marking only contains the message of Jane that has satisfied the condition, while the message of Paul has not been forwarded. ■

In the router example, given the initial marking $\{(Jane, 23), (Paul, 65)\}$ analyzed against the guard of T_1 , the marking $s_{expected}$ (cf. (Jane, 23) in Figure 3.14(a)) denotes the only allowed final state s_2 for a properly working router. More generally this is defined in Definition 3.22, which allows for a configurable, correctness criterion definition over finite traces induced by integration pattern models.

Definition 3.22 (Correctness Criterion). Let $s_1 \rightarrow \dots \rightarrow s_{n+1}$ be a finite execution trace of some pattern model *M* and $C = \{c_1, \dots, c_{n+1}\}$ be a set of reference *B*-snapshots that define a set of correct, desired states. We say that a pattern execution is *correct*, if exists $c_i \in C$ such that $c_i \sim s_i$ holds, for $i \in \{1, \dots, n+1\}$. The operator \sim typically denotes equality, but can also correspond to more sophisticated comparison operators (e.g., distributions, time spans) for relating reached and desired snapshots. ■

Note that the definition still captures the situation where target snapshots are enumerated explicitly. Other forms of validation (e.g., based on statistical goals formulated over the exhibited behaviors of the system) would require a more fine-grained approach able to aggregate snapshots and traces. This is matter of future work.

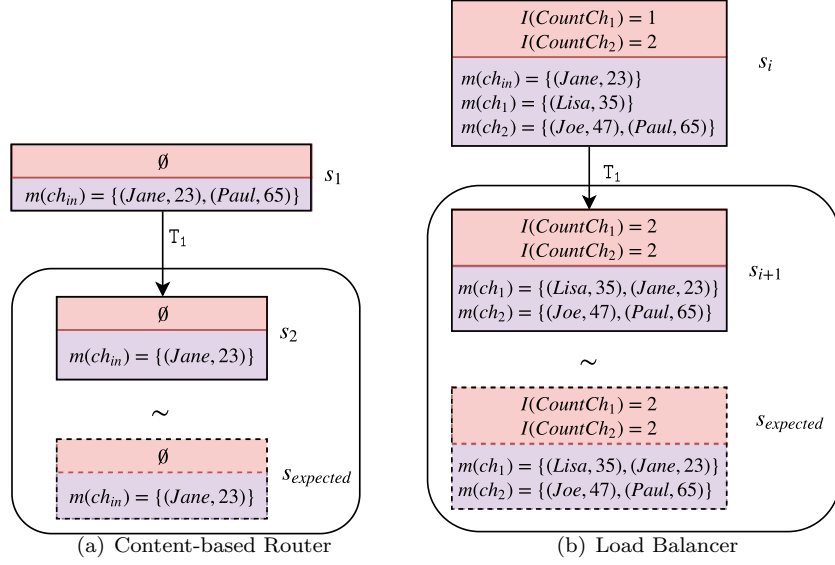


Figure 3.14: A finite db-net execution of a content-based router (*empl* short for employee) and a finite execution trace of a load balancer in timed db-net. Here we use *empl* to define an input place with employees and *count* as a function that counts a number of tokens in marking m .

Next we discuss the application of this correctness criterion for three different requirement categories from our analysis: control flow, data flow together with format and (transacted) resource, and timed patterns.

Control Flow Patterns To test control flow patterns for correctness, the operator \sim can be defined so as to compare the number of tokens in the correct, final snapshot. Nevertheless, there are control flow patterns whose correctness testing puts additional requirements on \sim . For example, the load balancer pattern (cf. REQ-1) denotes a special case, since it requires a sequence of input tokens, which then have to produce data entries in the output instances that fit the probability values and distribution of the balancer (e.g., Kolmogorov-Smirnov test [Kol33]). Therefore, the \sim operator has to check whether the number of tokens in the desirable states follows a probability distribution. Note that the subsequent example shows two snapshots for simplicity, however, actually a larger sample size of snapshots is required.

Example 3.23. Consider a load balancer in timed db-net B from Figure 3.7 (on page 79) with an initial B -snapshot s_1 containing a marking m with several messages composed of employee names and ages $m = \{(Jane, 23), (Lisa, 35), (Joe, 47), (Paul, 65)\}$. In the example shown in Figure 3.14(b), three of the tokens have already been distributed. This means that the current, observable snapshot s_i has a database instance with two tuples $CountCh1(1)$ and $CountCh2(2)$, and a marking with one token in the input state $m(ch_{in}) = \{(Jane, 23)\}$, one token in the first channel $m(ch_1) = \{(Lisa, 35)\}$ and two tokens in the second channel $m(ch_2) = \{(Joe, 47), (Paul, 65)\}$. Assuming that in the current state φ_1 holds, we can fire transition T_1 . This, in turn, generates a new state s_{i+1} . In order to check whether s_{i+1} is an expected state, we run a correctness test that is performed on the number of messages sent to the channels. Such a test allows us to see whether a final, desired message ratio is produced by the model. For example, knowing that the

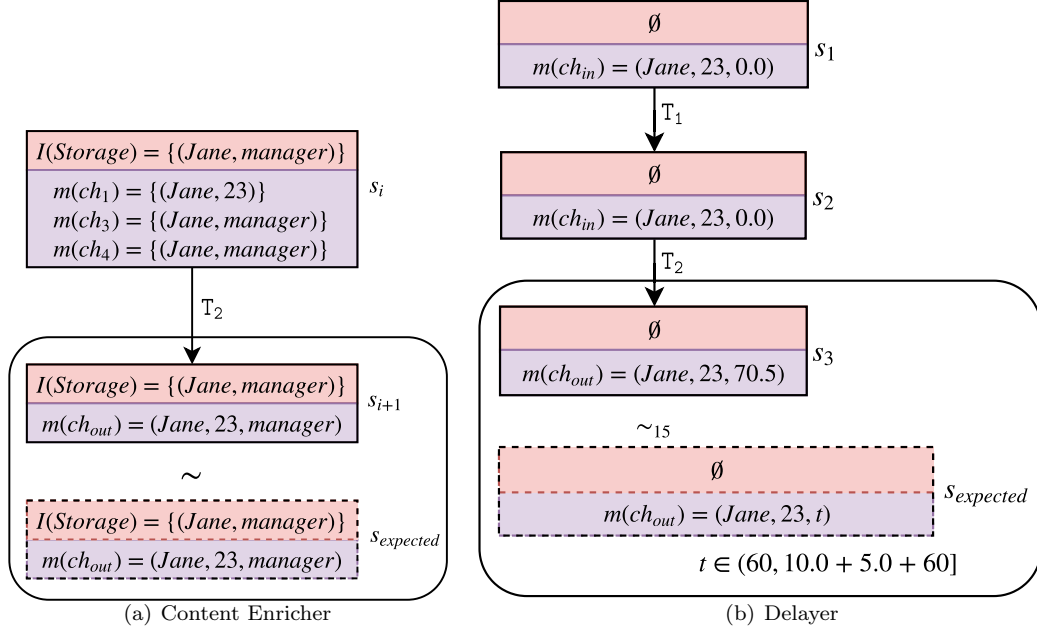


Figure 3.15: A partial execution of a content enricher timed db-net. For simplicity we omit the view place marking in s_{i+1} and $s_{expected}$ and a partial execution of a delayer timed db-net

bandwidth of the second channel is considerably greater than the one of the first channel, we may expect that the final ratio of $\frac{ch_1}{ch_2}$ is not greater than 0.7. Our \sim operator can be accordingly adopted to perform such a test. ■

The example shows that, even though the correctness testing of control flow patterns is feasible, there are cases in which such tasks may require extra workload, mainly on the configuration of the testing setup.

Data Flow and (transacted) Resource Patterns In order to test the correctness of patterns that meet requirements REQ-2 and REQ-4 (cf. Table 3.1, on page 67), one needs to consider testing not only the marking, as it is done in the case of control flow patterns, but also to compare states of the persistent storage. Specifically, for a given initial snapshot s_1 with an instance I_1 , either an expected state s_n with an instance I_n or an expected error state s_j must be produced by the pattern. Otherwise the pattern is considered incorrect.

Example 3.24. Let us consider a timed db-net B for the content enricher in Figure 3.10(b) (on page 81) with an initial B -snapshot s_1 that contains a marking m with a message composed of an employee name and age such that $m(ch_{in}) = (Jane, 23)$. In Figure 3.15(a) the database of the content enricher model stores information about employees and their positions in a relation called *Empl*. To reach the final state s_{i+1} from s_i , we need to fire T_2 . The fact that we have s_i with a marking containing two tokens (one with the employee's name and age, and another one with the same name and position) shows that, a few steps before, the employee token $(Jane, 23)$ was matched to the corresponding entry in the persistent storage and extra information about her position was extracted. If it was not the case, the execution trace, which is partially represented

in Figure 3.15(a), would not contain such two tokens that, in turn, would mean that the transition used for accessing the view-place could not fire since no matches were found. ■

Note that, however, in this example the internal database state does not play the main role when testing the correctness. The correctness checking is done on the markings which are populated from the database based on the matching condition assigned to the transition inspecting the view place.

Timed Patterns Finally, a timed pattern can be validated by extending database schemas with extra attributes for storing timestamps (as “on-insert timestamps” in actual databases) or by adding such timestamps to tokens, indicating the token creation time. This allows for checking delays, e.g., by comparing the insert timestamps $time(I_1)$, $time(I_n)$ of data to instance I_1 and those of the final instance I_n , or the timestamps in the tokens, respectively. With this, a numeric delay interval $d = (d_1, d_2]$ can be checked, with $d_1 = \tau$ being the delay configured in the pattern and $d_2 = \tau + avg(t_p) + var(t_p)$, the average time t_p and the variance the pattern requires for the internal transition firings without the configured delay. Since the delay τ is an interval itself, its upper value is taken for the application of the correctness criterion. More formally, we define the correctness criterion \sim from Definition 3.22 (on page 85) as relation \sim_k , with $(x, t) \sim_k (x', t') \iff x = x', t \in (t', t' + k]$.

Example 3.25. Consider a timed db-net B for the delayer in Figure 3.13(b) (on page 84) with an initial B -snapshot s_1 that has a marking m with a message composed of an employee’s name and age $m(ch_{in}) = (Jane, 23)$. In Figure 3.15(b) transition T_2 fires with a time delay of 60 time units. Since the delayer does not require a database state, the correctness of the timestamps is checked on the markings. In this example, we assume the average time $avg(t_{delayer})$ is 10.0 and the variance $var(t_{delayer})$ is 5.0 without the delay. This results in a desirable marking $(Jane, 23, t)$ in $s_{expected}$ with $t \in (60, 75]$ that, in turn, can be checked against the one in s_3 using \sim_k with $k = 15$ that, on top of comparing the states by equality, also compares whether the time stamp belongs to a desired time interval. ■

Erroneous Patterns The main sources of error during the responsible pattern formalization process in Figure 3.1 (on page 64) are the conceptual work on defining the formal representation of a pattern, as well as the model to implementation step, in which the formal model is implemented and configured. Subsequently, we briefly describe these types of errors by example.

Pattern Description to Model Errors. The formal representation of a pattern depends on different challenging factors concerning the quality and comprehensiveness of the pattern description as well as the clarity of its variations, and the complexity of the formalism. Consequently, the process of formalizing a pattern can introduce flaws prone due to understanding of the complex task at hand.

Example 3.26 (Content-based Router). While the Content-based Router in Figure 3.10(a) (on page 81) represents the pattern correctly, one could go wrong with the ordered execution (cf. REQ-1(a)), e.g., through transition guards at T_1 and T'_1 . If these transition guards were set with overlapping conditions, then several tokens would be produced in different output places, i.e., $m(ch_{out_1}) = \{(Paul, 65)\}$, $m(ch_{out_2}) = \{(Paul, 65)\}$, which does not match the desired state in the example in Figure 3.14(a). ■

Pattern Model to Implementation Errors. The model to implementation gap specifies the difficulties that can arise during the implementation of a formalized pattern. With the model on one side and the tool-specificities on the other, errors can occur during the translation and configuration.

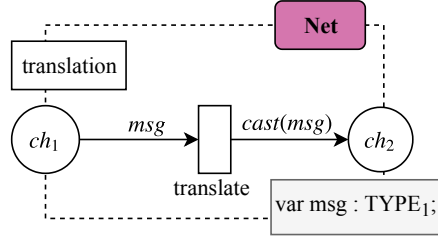


Figure 3.16: Sample *translation* subnet realization

While translation-related errors target particularities of the chosen tool or language, configuration errors can occur in user-defined subnets.

Example 3.27 (Message Translator). The Message Translator in Figure 3.8 (on page 80) allows for the configuration of a user-defined subnet that translates an input message msg of type $TYPE_1$ an output message of $TYPE_2$ using a special casting function $cast$. In case of an expected final state $m(ch_{out}) = \{(Paul, 65, London)\}$ for an input state $m(ch_{out}) = \{(Paul, 65, EC4M)\}$, a configuration of the subnet as shown in Figure 3.16 might not suffice, due to the resulting final state $m(ch_{out}) = \{(Paul, EC4M)\}$, which indicates structural or configuration issues (e.g., in the cast function). ■

3.1.3 Evaluation: Comprehensiveness, Correctness, and Case Studies

We quantitatively evaluate the comprehensiveness of the timed db-net formalism against the real-world integration scenarios (including pattern composition cases), show the correctness of the formal pattern realizations for the requirements discussed in Section 3.1.1 via the simulation, and discuss the application of our formalization approach to one hybrid integration (i.e., “on-premise to cloud” (OP2C)) and one internet of things integration scenario (i.e., “device to cloud” (D2C)) (cf. research question RQ2-1(c)).

Comprehensiveness of Timed Db-nets

The comprehensiveness of timed db-nets is evaluated with respect to the coverage of the patterns in the catalogs depicted in Figure 3.17(a). Here we compare the applicability of the existing CPN-based formalization [FG13] (*Current-CPN*), coloured Petri nets in general (*CPN (general)*) and timed db-nets (*timed db-net*). While the formalization proposed in [FG13] covers only some of the EIP from [HW04], many more EIPs as well as the recently extended patterns can be represented by coloured Petri nets. Now, as we have indicated in the previous sections, one can formalize all but one of the EIPs using timed db-nets. The only exception is one pattern, namely Dynamic Router, whose requirements cannot be represented using Petri net classes discussed in this work. In fact, in order to represent such a pattern one would need to employ a formalism that, on the one hand, subsumes db-nets and thus covers all the requirements discussed in Table 3.1 (on page 67) and, on the other hand, supports extra requirements (i.e., dynamically added or removed channels during runtime [FG13]) that, in turn, extend the expressiveness of the formalism with the ability to generate arbitrary topologies. To allow for such a functionality one might opt for an approach similar to the one in [FL01], where the authors enrich classical Petri nets with tokens carrying Linear Logic formulas that allows for dynamic re-configurations of a net based on those formulas. This, however, would require further investigations.

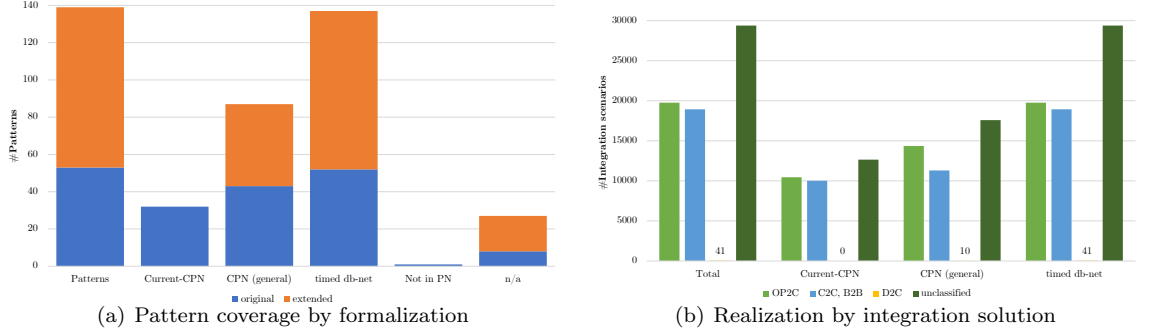


Figure 3.17: Timed db-net comprehensiveness

After having analyzed the pattern coverage per formalism, we now consider the relevance of such formalisms against real-world integration scenarios. For this we implemented a Content Monitor pattern (see Chapter 2), which allows for the analysis of the actually deployed integration scenarios that are, for example, running on SAP Cloud Platform Integration (SAP CPI) [SAP19a]. Figure 3.17(b) shows the coverage of the formalisms grouped by the following integration scenario domains, taken from Chapter 2: On-Premise to Cloud (short OP2C, also known as hybrid integration), Cloud to Cloud or Business Network (native cloud applications C2C, B2B), and Device to Cloud (D2C, including Mobile, IoT and Personal Computing) integration. The results show that the current approach by Fahland and Gierds [FG13] is only partially sufficient to cover the OP2C, C2C and B2B scenarios. With CPNs in general, more than 70% of more conventional OP2C communication patterns can be covered. The more recent and complex cloud, business network and device integration requires timed db-nets to a larger extend, which covers all analyzed scenarios. Note that the Dynamic Router with arbitrary topologies was not practically required for these scenarios, and thus seems to be rather of theoretical relevance.

Conclusions (1) timed db-nets are sufficient to represent most of the EIPs; (2) EIPs that are generating arbitrary topologies are not covered by considered PN classes; (3) hybrid integration requires less complex semantics and thus is largely in CPN; (4) timed db-nets cover all of the current integration scenarios in SAP CPI.

Simulation: Pattern Correctness Testing

We prototypically implemented the db-net formalism so as to experimentally test the correctness of the pattern realizations via simulation, following the idea described in Section 3.1.2. In order to test the correctness, we simply generate a finite execution trace, starting in an initial B -snapshot s_1 and finishing in s_n , using the prototype and inspect the generated marking together with the database instance. If s_n corresponds to an expected state according to Definition 3.22 (on page 85), then the test is considered to be successful. Since the inner workings of a pattern can differ between various pattern implementations (e.g., the implementation generates some intermediate states, which are not related to the actual pattern model, but are used, for example, for collecting statistics), the correctness can be also checked at any step of such pattern’s finite execution trace.

Prototype In this work we have chosen CPN Tools v4.0 [JKW07] (CPN Tools, visited 5/2019: cpntools.org) for the modeling and simulation. As compared to other PN tools like Renew v2.5 (Renew, visited 5/2019: <http://www.renew.de>), CPN tools supports third-party extensions

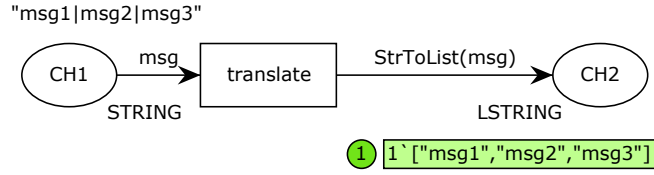


Figure 3.18: Message translator as a timed db-net (in CPN Tools)

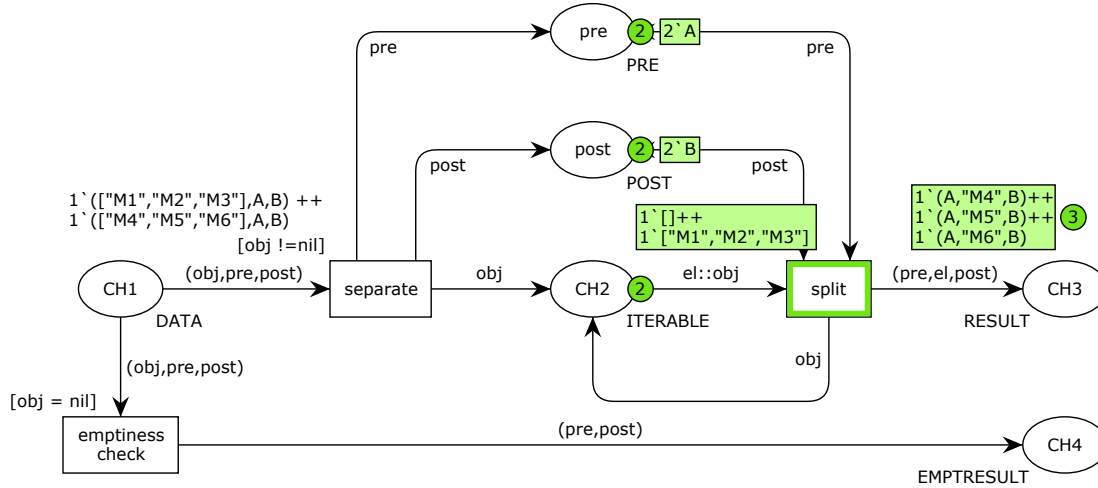


Figure 3.19: Splitter as a timed db-net (CPN Tools)

that can address the persistence and data logic layers of db-nets. Moreover, CPN Tools handles sophisticated simulation tasks over models that use the deployed extensions. To support db-nets, our extension⁵ adds support for defining view places together with corresponding SQL queries and actions, and realizes the full execution semantics of db-nets using a PostgreSQL database.

Simulation We illustrate the correctness for the majority of the formalized patterns from [Section 3.1.2](#) using the simulation in our CPN Tool extension. We focus on the following case studies: Message Translator, Splitter, Content Enricher and Aggregator. In addition, we discuss the case of a flawed example of the Content-based Router pattern from [Section 3.1.2](#). Together, these patterns denote the most frequently used patterns in practice according to [Chapter 5](#) and cover patterns from five out of seven categories discussed in [Section 3.1.2](#) (excluding “control flow only” and “control flow with transacted resources”).

Message Translator, Splitter. The realization of a variant of the message translator from Figure 3.8 (on page 80) is shown in Figure 3.18. Here, as input, the pattern receives a delimiter-separated string and translates it into a list of strings using a special function *StrToList* defined in CPN Tools. The final marking of the net shows the expected state $s_{expected} = \langle \mathcal{I}_{exp}, m_{exp} \rangle$ in which the database instance is empty (thus not shown) and the net is having only *CH2* marked such that $m_{exp}(CH2) = \{(\text{“msg1”}), (\text{“msg2”}), (\text{“msg3”})\}$.

The splitter from Figure 3.9(a) (on page 80) is implemented as shown in Figure 3.19. In this

⁵CPN Tools extension for timed db-net and pattern models available for download, visited 5/2019: <https://github.com/dritter-hd/db-net-eip-patterns>.

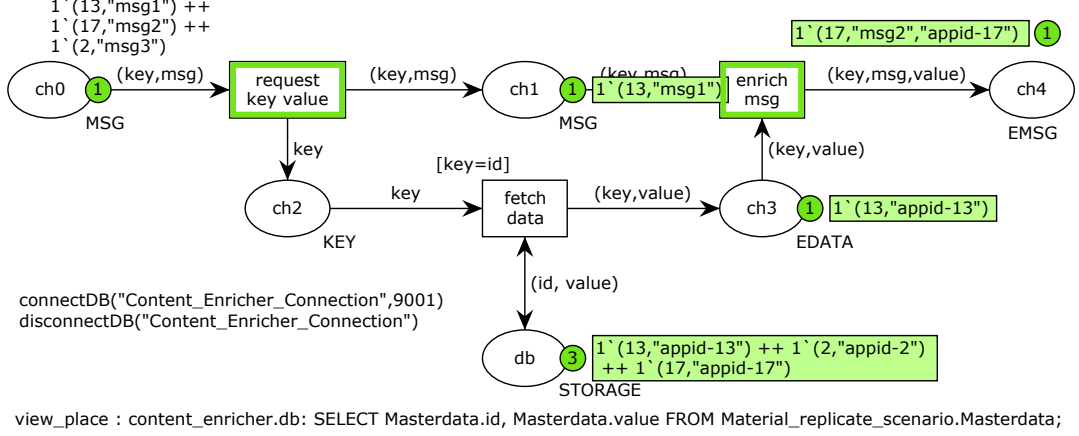


Figure 3.20: Content enricher as a timed db-net (in CPN Tools)

model, we have two input messages ($["M1", "M2", "M3"], A, B$) and ($["M4", "M5", "M6"], A, B$) consisting of iterable objects of size three each as well as pre and post data values A and B . These two messages are then split into six single objects of a shape $(A, "Mi", B)$, for $i \in 1, \dots, 6$. The partial execution of the splitter in Figure 3.19 demonstrates the second message to be already split (see the three output messages in place $CH3$), whereas the first message is ready to be split (i.e., the *split* transition is enabled⁶). Note that the current marking of the net can be already intermediately tested against the expected state $s_{expected} = \langle \mathcal{I}_{exp}, m_{exp} \rangle$ in which the database instance is empty and the marking is having only $CH3$ marked such that $m_{exp}(CH3) =$

$$\{(A, "M1", B), (A, "M2", B), (A, "M3", B), \\ (A, "M4", B), (A, "M5", B), (A, "M6", B)\}.$$

Indeed, it is easy to see that $m(CH3) \sim m_{exp} \setminus \{(A, "M1", B), (A, "M2", B), (A, "M3", B)\}$, indicating that elements of the second message have been correctly processed, by duly adding pre and post data values. The correctness of the splitter implementation, as it is defined in Definition 3.22, naturally follows.

Content Enricher. The content enricher from Figure 3.10(b) (on page 81) can be realized as shown in Figure 3.20. The demonstrated net has three messages (namely, $(13, "msg1")$, $(17, "msg2")$ and $(2, "msg3")$) in its initial marking and in its current state has already enriched message $msg2$ by adding to a corresponding token an extra data value "appid-17" from the storage (see place $ch4$), that is accessed through the view place called db . The data in db is stored in a shape of key-value pairs which are then matched with messages by their keys (that is, first components of the pairs). One can see that the net is ready to enrich $msg1$: the *enrich msg* transition is already enabled and the data from the storage that match the key of the token carrying $msg1$ had been fetched from db and placed in $ch3$. While the type of data used in different applications might require to reconfigure the query on the storage layer as well as to use a different enrichment function, the topology of the net representing the enricher remains the same. To test the correctness, we assume an expected state with a partial marking only. Specifically, we are interested in $m_{exp}(ch4) =$

$$\{(13, "msg1", "appid-13"), (17, "msg2", "appid-17"), (2, "msg3", "appid-2")\}.$$

⁶Graphically, enabled transitions are highlighted by a green frame, indicating that they are ready to fire.

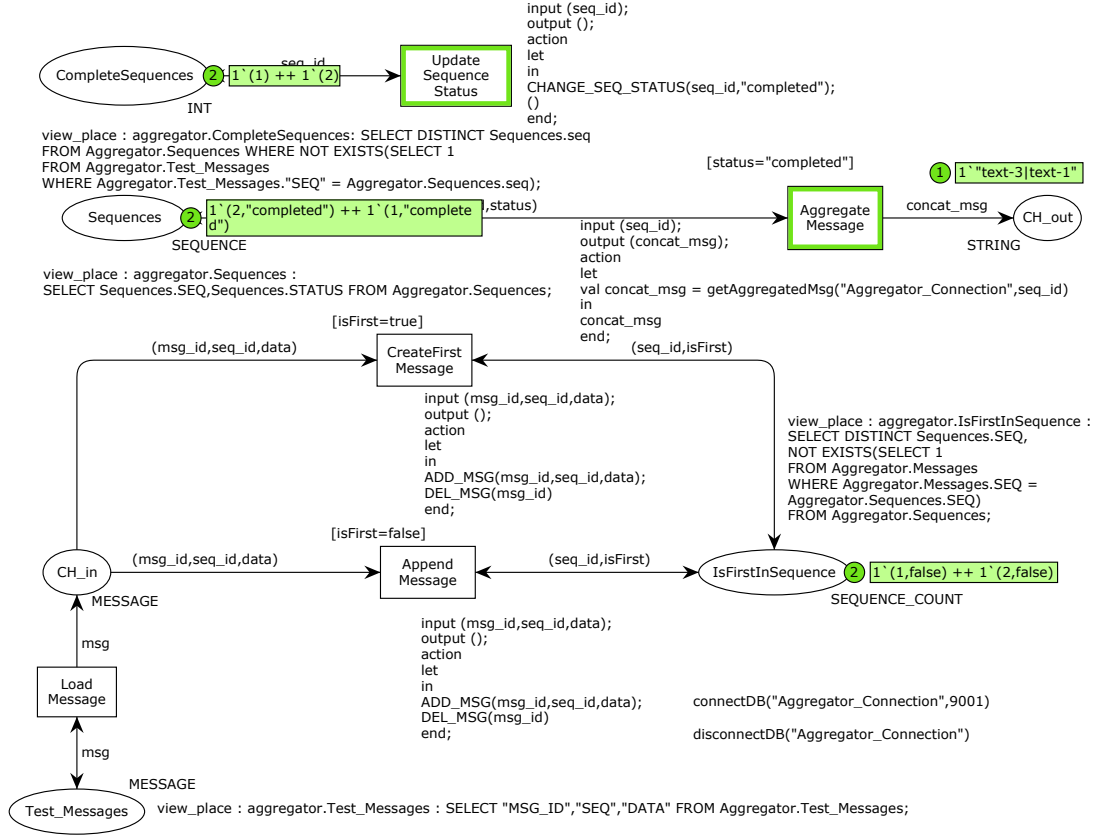


Figure 3.21: Aggregator as a timed db-net (in CPN Tools)

Given that the current net demonstrates the enricher being in its intermediate state and having processed only message one out of three, with its current marking in $ch4$ we have that $m(ch4) \sim m_{exp} \setminus \{(13, \text{"msg1"}, \text{"appid-13"}), (2, \text{"msg3"}, \text{"appid-2"})\}$, and thus can conjecture that the given pattern realization works as expected.

Aggregator. The Aggregator pattern in Figure 3.3 (on page 68) can be realized using our CPN Tool extension as it is shown in Figure 3.21. Here we neglect the timed completion condition due to the differing temporal semantics in the tool. For the ease of simulation, we added a table *Test_Messages* containing four test messages (1,1, "text-1"), (2,2, "text-2"), (3,1, "text-3"), (4,2, "text-4"), with ids from $\{1, \dots, 4\}$, two sequences $\{1,2\}$ and a textual payload. The completion condition is configured to aggregate after two messages of the same sequence and the aggregation function concatenates the message payloads separated by '|'. The expected result in the output place *CH_out* for the first sequence is one message with both payloads aggregated (1, "text-3—text-1").

Now, when establishing a connection to the database and to the CPN Tools extension server, the data from the connected database tables are queried and the net is initialized with the data from the database in place *CH_in*. We simulated the Aggregator realization in Figure 3.21 for the two test sequences, until one sequence was complete. The intermediate marking in

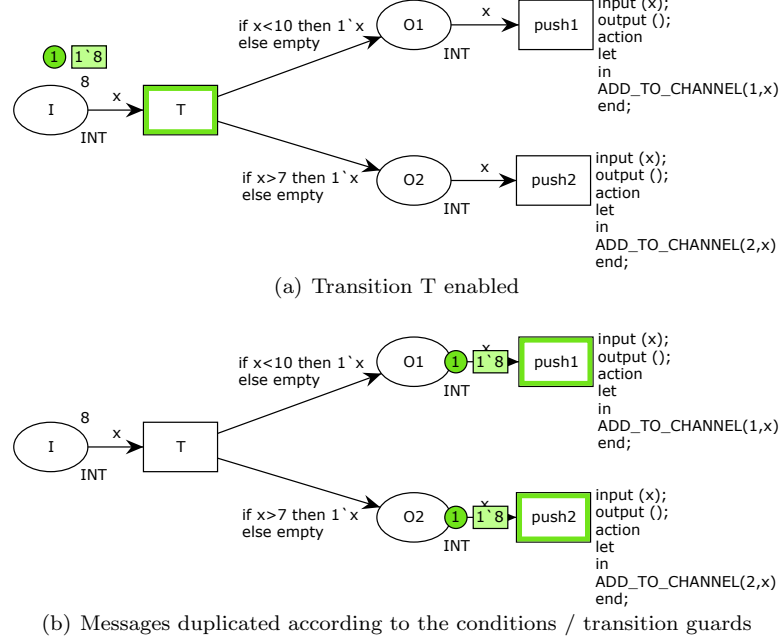


Figure 3.22: Flawed content-based router (in CPN Tools)

$m(CH_out) \sim m_{exp} \setminus \{("text-4" | "text-2")\}$, for $m_{exp}(CH_out) =$

$$\{("text-3" | "text-1"), ("text-4" | "text-2")\},$$

will eventually result to the expected outcome in CH_out and the database.

Flawed Content-based Router. While the previously discussed pattern implementations are correct, we added a *flawed* implementation of a content-based router, which is not required for the subsequent case studies, so as to demonstrate how the simulation could be used to detect an erroneous design. A Content-based Router, is a pattern that takes one input message and passes it to exactly one receiver without changing its content. This is done by evaluating a condition per recipient on the content of the message. Figure 3.22(a) shows one out of many router implementations, which may look correct, but, however, its process layer violates the correct design. For the evaluation we use the aforementioned method for “data and (transacted) resource-bound patterns”, which is based on the reachability of a correct database state. Such a correct state would be a database instance with one entry in table $Channel_1$ and an empty table $Channel_2$. This should happen due to the fact that the logical expressions on the arcs outgoing from T are expected to be disjoint. Now, let us explore the inner workings of the flawed pattern realization. In Figure 3.22(a), transition T reads the token in place I and then conditionally inserts it to the two subsequent places. Since the value of the token matches all conditions, both output places O_1 and O_2 receive a copy of the token as it is shown in Figure 3.22(b). In terms of application integration, this could mean that two companies receive a payment request or a sales order that was actually meant for only one of them. In the net, the two subsequent transitions $push_1$ and $push_2$ are enabled and fire by executing the database inserts defined in the $ADD_TO_CHANNEL(i, x)$ function, where i is being an index of one of the $Channel$ tables and

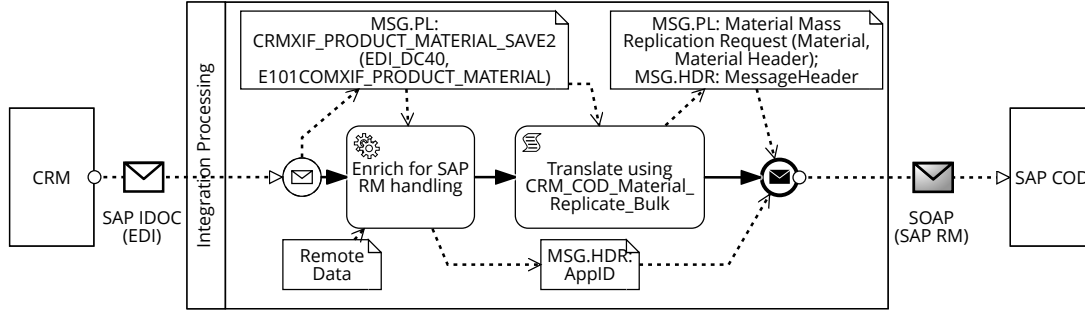


Figure 3.23: Replicate material from SAP Business Suite (a *hybrid integration* scenario in BPMN)

x is a data value to be inserted. From the net alone (i.e., in the initial state in Figure 3.22(a)), the pattern realization seems to be correct. However, after its execution, we can see that no correct state has been reached. Indeed, after the tokens have been processed on the timed db-net control layer, the database instance contains two entries (not shown), one in each table, that, in turn, would mean that the logical expressions that are meant to guard two different outputs are not disjoint, and by executing T we populated both $O1$ and $O2$ (instead of generating a token in only one of them).

Note that, when assuming one input token in I and a precedence of *push1* over *push2*, and considering that $I_{exp} = \{Channel_1(8)\}$, the final database instance $I(Channel_1)$ comes out to be as expected (that is, $I(Channel_1) \sim I(Channel_1)_{exp}$), whereas $I(Channel_2) \not\sim I_{exp}(Channel_2)$. It is easy to see that knowing the control-flow and data aspects a given timed db-net allows for detecting flaws in a pattern realizations as well as provide richer information for fixing them.

Conclusions (5) The CPN Tools extension allows for EIP simulation and correctness testing; (6) model checking implementations beyond correctness testing are desirable.

Applicability: Case Studies

The single patterns can be composed to represent integration scenarios, for which we study the formalism with respect to its applicability to two scenarios from the analysis: one hybrid OP2C and one D2C scenario.

Hybrid Integration: Replicate Material Many organizations have started to connect their on-premise applications such as Customer Relationship Management (CRM) systems with cloud applications such as SAP Cloud for Customer (COD) using integration processes similar to the one shown in Figure 3.23. A *CRM Material* is sent from the CRM system via EDI (more precisely SAP IDOC transport protocol) to an integration process running on SAP Cloud Platform Integration (SAP CPI) [SAP18a]. The integration process enriches the message header (MSG.HDR) with additional information based on a document number for reliable messaging (i.e., AppID), which allows redelivery of the message in an exactly-once service quality [RS16]. The IDOC structure is then mapped to the COD service description and sent to the COD receiver.

Formalization For this study, we manually encoded the BPMN scenario into a timed db-net as shown in Figure 3.24. While the message translator is close to the current CPN solution in [FG13], the Content Enricher (incl. the query on the machine's state) should be represented as timed db-nets. Consequently, the enricher is a pattern not covered before, for which soundness could not be checked. Hybrid integration usually denotes data movement between on-premise

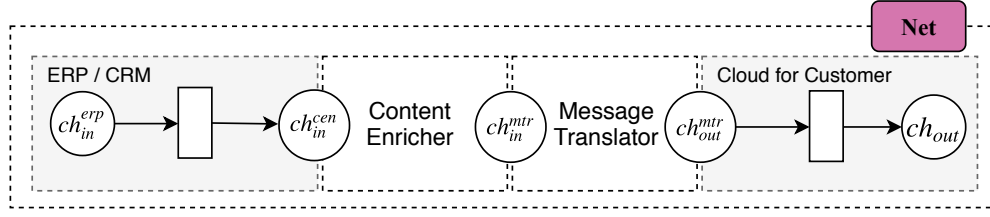


Figure 3.24: Replicate material scenario translated into its timed db-net representation (schematic)

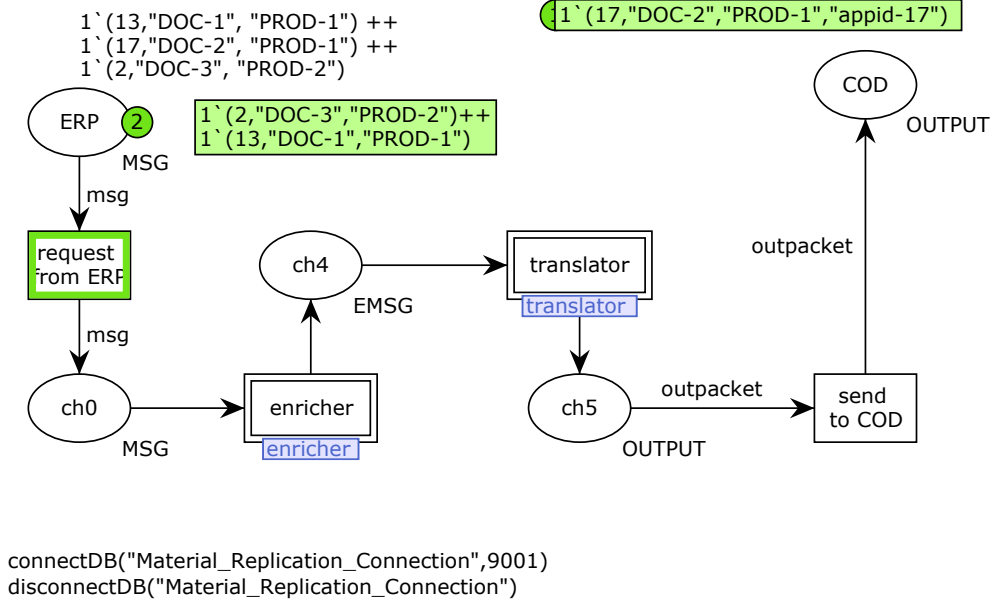


Figure 3.25: Pattern composition as hierarchical timed db-net (in CPN Tools)

and cloud applications, which do not require complex integration logic (cf. [Chapter 2](#). For these less complex hybrid integration scenarios the timed db-net representation gives richer insight into the data stored in the database as well as its manipulation (as opposed to, for example, BPMN), while the models remain still intuitively understandable.

Simulation The replicate material scenario in a timed db-net (cf. [Figure 3.24](#)) is implemented as a hierarchical net with our CPN Tools extension in [Figure 3.25](#), which references the pattern implementations of the enricher from [Figure 3.20](#) and translator from [Figure 3.18](#), annotated with *enricher* and *translator*, respectively. In the hierarchical model representing this scenario, the *MSG* message from the ERP system is enriched with master data. The derived enriched message of type *EMSG* is then sent to the translator that maps the intermediate message format to the one understood by the COD system, thus generating a new message of type *OUTPUT*. Note that in this case the arc inscriptions abstractly account for messages without revealing their concrete structure.

In order to check the correctness of the given scenario, one has to keep in mind that, in general, the composition of the single patterns as timed db-nets requires a careful, manual alignment of the “shared” control places (e.g., *ch0*, *ch4* and *ch5*) with respect to the exchanged data and

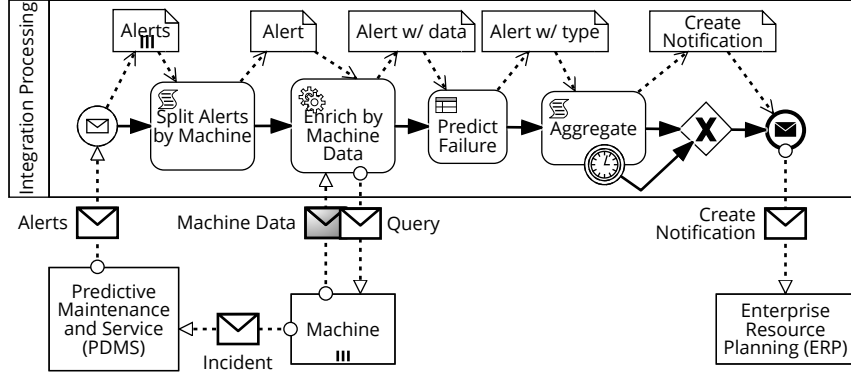


Figure 3.26: Predictive maintenance — create notification scenario as modeled by a user (an “internet of things” scenario in BPMN)

the characteristics of the neighboring patterns. Thus it is required to carefully consider various pattern characteristics together with input and output message types to ensure its correctness. Assume that the expected marking in our case is $m_{exp}(COD) =$

$$\{(13, \text{"DOC-1"}, \text{"PROD-1"}, \text{"appid-13"}), \\ (17, \text{"DOC-2"}, \text{"PROD-1"}, \text{"appid-17"}), \\ (2, \text{"DOC-3"}, \text{"PROD-2"}, \text{"appid-2"})\}.$$

Then, given the intermediate marking in COD , we can see that $m(COD) \sim m_{exp} \setminus \{(13, \text{"DOC-1"}, \text{"PROD-1"}, \text{"appid-13"})\}$ and thus conjecture that the scenario is correct. Note that, while the composition in Figure 3.25 denotes a correct implementation of the replicate material scenario, the general question of composition correctness remains open.

Conclusions (7) timed db-net representations allow for an understandable, sound and comprehensive representation of single patterns and their compositions; (8) the correctness of the compositions requires further considerations.

Internet of Things: Predictive Maintenance and Service (PDMS) In the context of digital transformation, an automated maintenance of industrial machinery is imperative and requires the communication between the machines, the machine controller and ERP systems that orchestrate maintenance and service tasks. Integrated maintenance is realized by one of the analyzed D2C scenarios in Section 3.1.3, which helps to avoid production outages and to track the maintenance progress. Thereby, notifications are usually issued in a PDMS solution as shown in Figure 3.26 from SAP CPI, represented in BPMN according to [RS16].

Although we simplified the scenario, the relevant aspects are preserved. Industrial manufacturing machines, denoted by *Machine*, measure their own states and observe their environment with sensors in a high frequency. When they detect an unexpected situation (e.g., parameter crosses threshold), they send an incident to a local endpoint (e.g., IoT edge system), the *PDMS*, indicating that a follow-on action is required. The *PDMS* system creates alerts for the different machines and forwards them to a mediator, connecting the *PDMS* to the *ERP* system. To throttle the possibly high frequent alerts, several incidents are collected (not shown) and sent as list of alerts. Before the *ERP* notification can be created, additional data from the machines are queried based on the split and single alerts, and then enriched with information that adds the feature

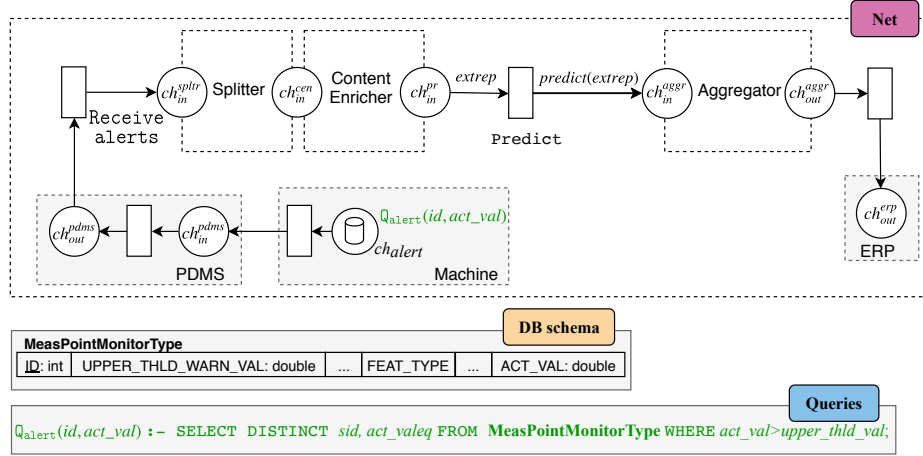


Figure 3.27: Create notification scenario translated into timed db-nets (schematic)

type. The information of the single alerts is used to predict the impact by value and machine type, and then gets aggregated to be sent to ERP. In case the notification has been created successfully in ERP, the PDMS gets notified including the service task identifier and thus stops sending the alert (not shown).

Formalization Again, we manually translated the BPMN scenario into a timed db-net as shown in Figure 3.27. While the splitter from Figure 3.9(a) (on page 80) is an extension of the current CPN solution in [FG13] and configured by the split condition from Figure 3.9(b), the content enricher (incl. the query on the machine’s state) and Aggregator require the functionality of timed db-nets. Note that alerts in the PDMS system are created based on query Q_{alert} that returns the device id and the critical value act_val . The additional feature type $feat_type$ is provided by query Q_{get} , using the content enricher from Figure 3.10(b). Finally, the Aggregator from Figure 3.3 (on page 68) is configured to concatenate the machine names.

Simulation The predictive maintenance scenario in timed db-nets (cf. Figure 3.26) is implemented as hierarchical net with our CPN Tools extension in Figure 3.29, which references the pattern implementations of the enricher from Figure 3.20 and translator from Figure 3.21, annotated with *enricher*, *Aggregator*, respectively. In the original scenario, the PDMS sends lists of incidents to the integration system to reduce the number of requests as shown in Figure 3.28. The incidents have an incident ID, a machine ID, and the actual critical incident value (e.g., (101,1,76)). Unfortunately, due to the fact that CPN Tools does not support third party extensions with complex data types like lists, it was decided to make the *PDMS* component emit single messages. Consequently, the splitter is not required for separating the single incidents, but the incident messages of type *REPORT* are immediately enriched by the *enricher*. After master data has been added to the message, a new one of type *E-REPORT* has been produced. The net then immediately proceeds with predicting the impact using transition *predict*, which usually assesses the probability of a timely machine error based on previous experiences with the particular machine type. Here, for simplicity, the prediction is always set to *true* and the results are placed into *prediction result*. Tokens in this place are then used to aggregate several incidents by machine, where, for simplicity, we use machine identifiers to identify Aggregator’s sequences. The aggregated incident messages are then sent to the *ERP* system. With the final marking in $m(ERP)$ and $m_{exp}(ERP) = \{\text{“Assembly Robot”}, \text{“Engine Robot”} | \text{“Engine Robot”} | \text{“Engine Robot”}\}$, for the three incidents from machine *Engine Robot* and one from *Assembly Robot*, we can see that

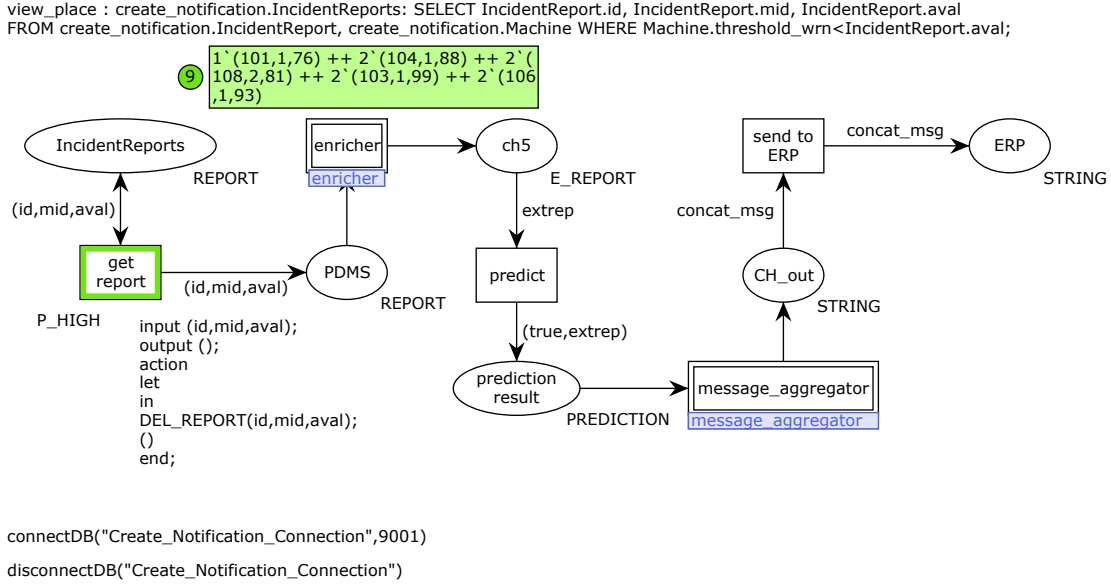


Figure 3.28: Create notification pattern composition as hierarchical timed db-net before simulation (in CPN Tools)

$m(ERP) \sim m_{exp}$ and thus conjecture that the scenario is correct.

Although the resulting timed db-net provides so far unmatched insights into the different aspects of integration scenarios, the complexity of the composed patterns increased even, when using hierarchical nets.

Conclusions (9) timed db-net representations allow for an explicit modeling of all data aspects in complex data-aware scenarios (e.g., roll-back, queries); (10) the formalism's technical complexity might prevent non-technical users from using it on a regular basis.

Discussion

With the timed db-net formalization, it is possible to model and reason about EAI requirements like data, transacted resources and time (cf. conclusions (1), (5)), going beyond the simple hybrid integration scenarios (cf. conclusion (3)). Thereby the pattern realizations are self-contained, can be composed into complex integration scenarios (e.g., Figure 3.27; cf. conclusion (7)) and analyzed (cf. conclusions (4)), while leaving the extension of our tool prototype to model checking as well as a formal treatment of pattern compositions as future work (cf. conclusions (6), (8), respectively). The composition is facilitated manually through carefully defining "sharing" control places, preventing unwanted side-effects between patterns.

However, there are some limitations that we briefly discuss next. PN classes considered in this work fall short when it comes to generation of places, transitions or arcs (cf. conclusion (2)). For example, Dynamic Router requires a proper representation of dynamically added or removed channels. Further, the deep insights into data-aware patterns and scenarios lead to the trade-off between sufficient information and model complexity (cf. conclusion (9)). The complexity of PN models compared to their BPMN counterparts in Figure 3.26 might not allow for modeling by non-technical users (cf. conclusion (10)). Hence, we propose modeling in a less technical modeling notation, which can be then encoded into PN models, e.g., for verification. Further, while the PN

Table 3.2: Formalization requirements (summary)

ID	Requirement	CPN	db-net	timed db-net
REQ-0	Control flow (pipes and filter)	✓	✓	✓
REQ-1	(a) Message channel priority	(✓)	(✓)	(✓)
	(b) Message channel distribution	-	(✓)	(✓)
REQ-2	Data, format incl. message protocol with encoding, security	(✓)	✓	✓
REQ-3	(a) Timeout on message, operation	-	-	✓
	(b) Expiry date on message	-	-	✓
	(c) Delay of message, operation	-	-	✓
	(d) Message/time ratio	-	-	✓
REQ-4	(a) Create, Retrive, Update, Delete (CRUD) operations on (external) resources	-	✓	✓
	(b) Transaction semantics on (external) resources (incl. roll-back)	-	✓	✓
REQ-5	Exception handling, compensation (similar to roll-back in REQ-4 for transactional resources)	-	✓	✓

covered ✓, partially (✓), not -.

for the composition of the single, timed db-net patterns (cf. objective (iv)), and thus more abstract representations (e.g., for the application of optimizations). This denotes the basis for structurally and semantically correct pattern compositions as well as subsequently introduced correctness-preserving optimizations.

3.2 Composing Patterns

With the formalization of the integration patterns, the fundamental EAI building blocks can be experimentally validated through simulation and formally analyzed, and thus, e.g., verified for functional correctness. However, the formal treatment of integration solutions and their improvements requires compositions of patterns. A *pattern composition* is a combination of several patterns (i.e., filters) that are connected by message channels (i.e., pipes) in a pipes-and-filter style (cf. Section 2.1.2). Already the EIPs describe a few compositions as composite patterns like Composite Message Processor in Figure 2.2(b) (on page 24; essentially a combination of Splitter, Content-based Router, and Aggregator patterns), which indicates that the integration patterns are actually meant to be composed to represent integration logic that connects several endpoints in an integration scenario.

While the composite patterns denote best-practices of pattern compositions known in 2004, the example scenarios in Section 3.1.3 represent more elaborate, real-world integration scenarios. The evaluation of these pattern compositions formalized as a timed db-nets essentially resulted in the following observations, which essentially denote a trade-off between (i) a comprehensible and simple modeling of integration scenarios, and (ii) an expressive and comprehensive coverage of the execution semantics of integration patterns and compositions that allows for the analysis of their correctness. First, (i) while the PN grounding of timed db-nets allows for a composition of the single pattern formalizations into a graph of timed db-net patterns as hierarchical PNs, the complexity of the solution through its fine granularity (i.e., PN places and transitions) results in a decline in its comprehensibility and accessibility even for integration experts, and blurs the *boundaries* of the single, self-contained patterns. Since the patterns' internal logic is not expected to change often, after a *correct* pattern is developed, the question is raised, whether the

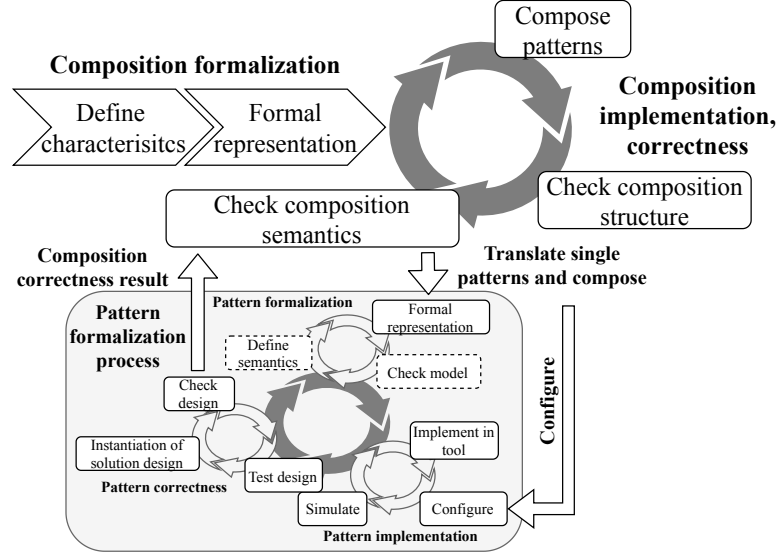


Figure 3.30: Responsible pattern composition process (process-perspective)

user needs to deal with the pattern logic explicitly. While being more explicit, the pattern logic complexity limits the creation of newly formalized pattern compositions, the comprehension of existing solutions and their adaptation or extensibility (e.g., [FG13]). This can be easily seen in the example scenarios in Section 3.1.3, in which the single, self-contained patterns are graphically identifiable only through their dashed *boundaries*, while the token exchange is facilitated through places that are visible for the neighboring patterns. We argue that a suitable formalism should be directly represented by a higher-level graph structure that does not require the user to understand the construction of a pattern on a lower level timed db-net. To formally ground such a graph representation (e.g., with respect to execution semantics), a translation from the graph to its executable timed db-net representation is mandatory. Second, (ii) the correctness of these compositions is not guaranteed. The resulting concerns about the correctness of a composition as well the identification and application of improvements such as optimizations, require a further sophistication of timed db-nets towards a formal composition model, which we study driven by the following sub-questions of RQ2-2 “What is a sound and comprehensive formal representation of integration patterns that allows for formal validation of integration scenarios and reasoning?”:

- (a) *How can pattern compositions be suitably formalized for compositional correctness?*
- (b) *How to assess and guarantee the correctness of composed, formalized patterns?*
- (c) *How to realize formalized pattern compositions in real-world integration scenarios?*

To address these questions, we follow a *responsible pattern composition process* in Figure 3.30, similar to the process for the pattern formalization (cf. Figure 2.5 (on page 27)). In the *composition formalization* and *formal representation* steps the *composition formalization* is addressed (cf. research question RQ2-2(a)). Intuitively, the single patterns and their boundaries become more prominent, when acknowledging the timed db-net control layer as a graph (i.e., without persistence and data logic layers), applying a pattern-oriented condensation mechanism [CLRS09] (i.e., patterns denote strongly connected components in that sense), and visually highlighting the patterns (containing PN places and transitions) as boxes and overlaying the “shared” PN

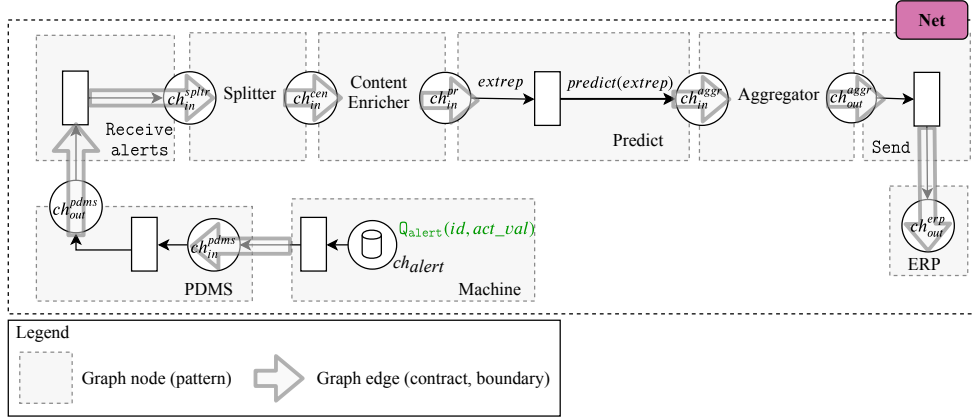


Figure 3.31: Condensation based on pattern boundaries of example from Figure 3.26

places by arrows, giving the direction of the data exchange. The resulting condensed overlay graph for the predictive maintenance composition from Figure 3.26 is shown in Figure 3.31. Such a graph-based representation of pattern compositions makes them more comprehensible and accessible. Hence, for a simpler definition of scenarios in step *compose patterns*, we assume the patterns are composed in a graph-like manner to additionally make an abstract definition of improvements (e.g., composition-level optimizations) more tractable (i.e., define identification of optimizations and rewritings on graphs and not PNs). This essentially targets to solve the identified complexity problem in Section 3.1.3 (cf. observation (i)).

For the considerations on the yet unsolved correctness of pattern boundaries (cf. observation (ii)) and correctness guarantees (cf. research question RQ2-2(b)), we differentiate between structural and semantic composition criteria, which we address in separate steps, “check composition structure” and “check composition semantics”, respectively. Ideally, the structural correctness criteria is built-in the pattern composition formalism (i.e., the higher-level graph representation), and can thus be immediately decided on the graph level in the form of *composition contracts* that do not allow for structurally incorrect compositions. The structural correctness could thus be checked without translation to an executable representation, such as timed db-net. The semantic correctness, however, requires more elaborate validation or verification on the timed db-nets pattern level, for which we embed the responsible pattern formalization process into the overall composition process. Note that the *define semantics* step is not required, since we assume that the patterns are already defined. However, similar to the contracts on the graph level, the timed db-net patterns require constructs — called *boundaries* — that allow for the formal analysis when they are composed. Hence, the contract graphs are directly translated into timed db-nets with such boundaries, verified or configured and then validated through simulation. If the composition is not structurally or semantically correct, it can be adjusted in the *compose patterns* step. We argue that this way correct real-world integration scenarios (cf. research question RQ2-2(c)) can be developed. The separation of higher-level graphs that can be translated to executable lower-level timed db-nets addresses observations (i) and (ii): the graph structure abstracts from the patterns inner workings as well as their boundaries in the technical PN representation, and thus is more comprehensible for a user and allows for faster modeling of integration scenarios and subsequently for a less complex definition of optimization strategies. The translation to the lower-level timed db-nets with boundaries allows for semantic correctness checks of the execution semantics, while a translation mechanism abstracts from the technical peculiarities of

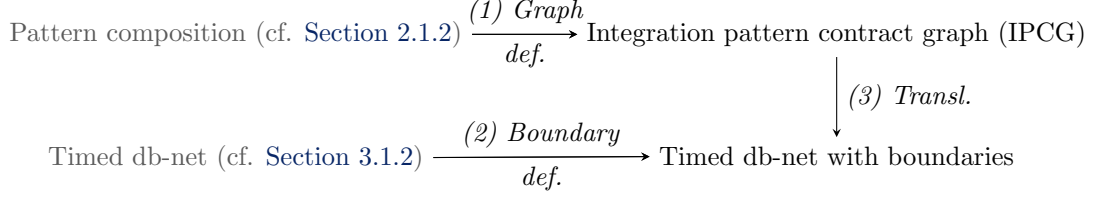


Figure 3.32: From pattern contract graphs to timed db-nets with boundaries (methodological)

the underlying PN. A translation from the lower-level PN to the graphs is not required and might only be useful, when new patterns are developed, which we leave as future work.

Methodologically, this is summarized in Figure 3.32, illustrating the outline of this section. First, we analyze the structural properties of current integration patterns in Section 3.2.1, which provides constraints on the structure of pattern composition graphs. Based on these structural properties and pattern characteristics — representing semantic pattern aspects — pattern compositions are formally defined using control flow graphs and extending them with data flow within and between integration patterns in Section 3.2.2 (cf. (1) *Graph def.*), which we call *Integration Pattern Contract Graph* (IPCG). The structural correctness of these compositions is defined through *contracts* between patterns, and thus the name integration pattern contract graphs. For the semantic correctness, timed db-nets are extended with boundaries (i.e., matching the contracts on the graph level) in Section 3.2.3. For such compositional contracts in the PN domain, *Open Nets* [BBGM15, BM18] are usually considered. In our case, open nets allow for the definition of pattern boundaries (cf. (2) *Boundary def.*) in the form of PN constructs (e.g., place, transition) that are necessary to send the data to a subsequent pattern or read from the predecessor (i.e., arrows in Figure 3.31), which is in line with the pipes-and-filter composition (cf. Section 2.1.2). In particular, such a boundary constitutes the contract between two or more patterns (e.g., PN token colors correspond to those of the PN places), and thus allows for an assessment of the correctness of a pattern composition. Then structural pattern contract graphs are translated to boundary-aware timed db-nets in Section 3.2.4 (cf. (3) *Transl.*). The replicate material scenario in Figure 3.23 is used as running example throughout the formal sections. The applicability and soundness of our approach are evaluated by revisiting the replicate material and predictive maintenance scenarios in case studies in Section 3.2.5, before we conclude in Section 3.2.6.

3.2.1 Structural Pattern Analysis

Besides the pattern characteristics, collected in Section 3.1.1 (e.g., control and data flow, time), the structural property of channel cardinality is relevant for the definition of a pattern composition. The channel cardinality denotes the number of input and output message channels of a pattern including (transactional) access to external resources (e.g., databases). A message channel signifies a message-based transport of a message as for pipes in pipes-and-filters. For example, the Aggregator in Figure 3.3 and the Splitter in Figure 3.9(a) each have one input and one output channel, and thus has a channel cardinality of one-to-one. And, while the Content-based Router in Figure 3.10(a) has one input and several conditional output channels (i.e., conditional one-to-many or fork), the Load Balancer in Figure 3.7 denotes an unconditional fork pattern.

More systematically, from the extended pattern catalog in Chapter 2, we manually identified 102 out of the 166 integration patterns with structural significance (i.e., excluding abstract concepts like Canonical Data Model or Messaging System) and structurally classified them.

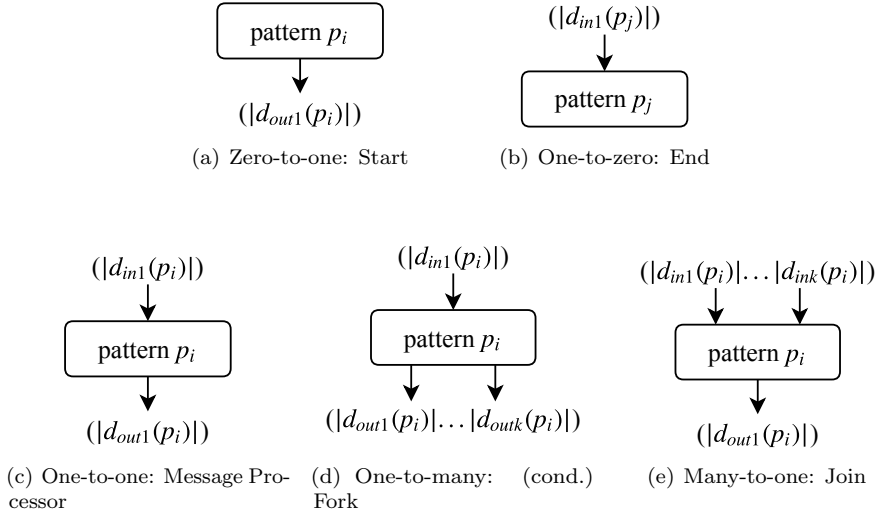


Figure 3.33: Structural integration pattern categories: Start, end, message processor, fork, join

Figure 3.33 show all but one of the resulting pattern categories: Start, End, Message Processor, (conditional) Fork, and Join, which we subsequently discuss.

From the perspective of an integration pattern composition, the message endpoint patterns (e.g., Message Endpoint, Idempotent Receiver, Commutative Receiver) denote patterns with unknown input or output. A message sending endpoint p_i is specified as in Figure 3.33(a) by one output channel with data cardinality $|d_{out1}(p_i)|$ and a message receiving endpoint p_j is shown in Figure 3.33(b) with one input channel and data cardinality $|d_{in1}(p_j)|$. The data cardinalities denote arbitrary data sets within the exchanged messages. The majority of the patterns belongs to the one-to-one category, shown in Figure 3.33(c), which we call Message Processor. Prominent examples from the routing patterns are the Aggregator and Splitter as well as most of the transformation patterns like Message Translator and Content Filter.

The category of (conditional) fork patterns have one incoming, but several outgoing message channels, as depicted in Figure 3.33(d). While the unconditional fork patterns (e.g., Multicast, Content Enricher) use all of their message channels every time they are called, the conditional fork patterns (e.g., Content-based Router, Detour) effectively use only some of the outgoing channels, which are selected based on a (routing) condition.

The structural antipode of a fork is a many-to-one pattern that we call join. The join structurally combines several incoming message channels to one as shown in Figure 3.33(e). The incoming data is not changed, but forwarded to a single channel, assuming that the data format of the incoming message complies with the outgoing channel.

Since only one pattern can be regarded as many-to-many (i.e., Control Bus) and this category can be represented by a subsequent join and fork, it is not shown.

Subsequently, an identified structural integration pattern categories are relevant for the definition of structurally correct pattern compositions and server as templates during the definition and the translation to corresponding timed db-nets with boundaries.

3.2.2 Graph-based Pattern Compositions

With the results from the semantic (cf. Section 3.1.1) and structural pattern analyses, we step-wise define graph-based, structurally correct pattern compositions. As motivated before, such a formalization is needed in order to verify the compositional correctness or talk about composition-level optimizations more rigorously.

First, we subsequently define an *Integration Pattern Typed Graph* (IPTG) based on the structural pattern categories from Section 3.2.1 as an extended, acyclic *Control Flow Graph* (cf. Prosser [Pro59] and Allen [All70]), as also used for optimizing compilers (e.g., Offner [Off13]). Informally speaking, a control flow graph is a directed graph of a program, where the nodes denote a set of program instructions that are executed in an ordered way. Similar to the program instructions, the integration patterns have to be executed in order, and thus denote nodes in our graph. More formally, control flow graphs denote a special case of labeled graphs, where the definition of a labeled graph requires a label alphabet as in Definition 3.28 according to Taentzer et al. [EPT06], Ehrig et al. [ERK99].

Definition 3.28 (Labeled graph). A *label alphabet* $C = (C_V)$ comprises a set C_V of node labels.

A *graph* over a label alphabet C is a system $G = (V_G, E_G, s_G, t_G, l_G)$ comprising a finite set V_G of nodes, a finite set E_G of edges, source and target functions $s_G, t_G : E_G \rightarrow V_G$, and a partial node labeling function $l_G : V_G \rightarrow C_V$. A graph G is empty, if $V_G = \emptyset$. A graph G is totally labeled, if l_G is a total function. ■

Subsequently, we write $G = (P, E)$ for a labeled graph G , where the set P denote a set of program instruction representing a pattern as nodes, with the set of edges and source and target functions represented by a relation E . We specify the node labeling function l_G separately. Consequently, we think of pattern types as sets of program instructions. For example, pattern types like the conditional fork is “a set of program instructions”, and thus a node referred to as *condition*. In addition to the structurally separated patterns types, the complex external call (actually a fork) and merge (actually a message processor) are added as separate pattern types. Let us first fix some notation: a directed graph is given by a set of nodes P and a set of edges $E \subseteq P \times P$. For a node $p \in P$, we write $\bullet p = \{p' \in P \mid (p', p) \in E\}$ for the set of direct predecessors of p , and $p \bullet = \{p'' \in P \mid (p, p'') \in E\}$ for the set of direct successors of p .

Definition 3.29 (Integration pattern type graph). An *integration pattern typed graph* (IPTG) is a directed graph with set of nodes P and set of edges $E \subseteq P \times P$, together with a function $type : P \rightarrow T$, where $T = \{\text{start, end, message processor, fork, structural join, condition, merge, external call}\}$. An IPTG $(P, E, type)$ is *correct* if

- $\exists p_1, p_2 \in P$ with $type(p_1) = \text{start}$ and $type(p_2) = \text{end}$;
- if $type(p) \in \{\text{fork, condition}\}$ then $|\bullet p| = 1$ and $|p \bullet| = n$, and if $type(p) = \text{join}$ then $|\bullet p| = n$ and $|p \bullet| = 1$;
- if $type(p) \in \{\text{message processor, merge}\}$ then $|\bullet p| = 1$ and $|p \bullet| = 1$;
- if $type(p) \in \{\text{external call}\}$ then $|\bullet p| = 1$ and $|p \bullet| = 2$;
- The graph (P, E) is connected and acyclic. ■

In the definition, we think of P as a set of extended EIPs that are connected by message channels in E , as in a pipes-and-filters architecture. The function $type$ records what type of pattern each node represents. The first correctness condition says that an integration pattern has at least one source and one target, while the next three states the cardinality of the involved patterns coincide with the in- and out-degrees of the nodes in the graph representing them. The

last condition states that the graph represents one integration pattern, not multiple unrelated ones, and that messages do not loop back to previous patterns. Although the structural pattern categories would allow for cycles, neither the current patterns nor the analyzed integration scenarios require them, and thus we leave cyclic compositions for future work.

Second, to represent the data flow, i.e., the basis for the optimizations, the control flow has to be enhanced with (a) the data that is processed by each pattern, and (b) the data exchanged between the patterns in the composition. The data processed by each pattern (a) is described as a set of *pattern characteristics*, formally defined as follows:

Definition 3.30 (Pattern characteristic). A *pattern characteristic* assignment for an IPTG $(P, E, type)$ is a function $char : P \rightarrow 2^{PC}$, assigning to each pattern a subset of the set

$$\begin{aligned} PC = & (\{MC\} \times \mathbb{N} \times \mathbb{N}) \cup \\ & (\{ACC\} \times \{ro, rw\}) \cup \\ & (\{MG\} \times \mathbb{B}) \cup \\ & (\{CND\} \times 2^{BExp}) \cup \\ & (\{PRG\} \times Exp \times (\mathbb{Q}^{\geq 0} \times (\mathbb{Q}^{\geq 0} \cup \{\infty\}))) , \end{aligned}$$

where \mathbb{B} is the set of Booleans, $BExp$ the set of Boolean expressions, Exp the set of program expressions, and MC , CHG , MG , CND , PRG some distinct symbols. ■

The property and value domains in [Definition 3.30](#) are based on the pattern descriptions in [\[HW04, RMRM17\]](#), and could be extended if further analysis required it. We briefly explain the intuition behind the characteristics: the characteristic (MC, n, k) represents a message cardinality of $n:k$, (ACC, x) the message access, depending on if x is read-only (ro) or read-write (rw), and the characteristic (MG, y) represents whether the pattern is message generating depending on the Boolean y . Finally $(CND, \{c_1, \dots, c_n\})$ represents the conditions c_1, \dots, c_n used by the pattern to route messages, $(PRG, (p, (v, v')))$ the program used by the pattern for message translations, together with its timing window. The characteristics determine a node labelling function l_G for the graph.

Example 3.31. The characteristics of a Content-based Router CBR is $char(CBR) = \{(MC, 1:1), (ACC, ro), (MG, false), (CND, \{cnd_1, \dots, cnd_{n-1}\})\}$, because of the workflow of the router: it receives exactly one message, then evaluates up to $n - 1$ routing conditions cnd_1 up to cnd_{n-1} (one for each outgoing channel), until a condition matches. The original message is then rerouted read-only (in other words, the router is not message generating) on the selected output channel, or forwarded to the default channel, if no condition matches. ■

The data exchange between the patterns (b) is based on *input and output contracts* (similar to data parallelization contracts in [\[BEH⁺10\]](#)). These contracts specify how the data is exchanged in terms of required message properties of a pattern during the data exchange, formally defined as follows:

Definition 3.32. A *pattern contract* assignment for an IPTG $(P, E, type)$ is a function $contr : P \rightarrow CPT \times 2^{EL}$, assigning to each pattern a function of type

$$CPT = \{\text{signed, encrypted, encoded}\} \rightarrow \{\text{yes, no, any}\}$$

and a subset of the set

$$EL = MS \times 2^D$$

where $MS = \{HDR, PL, ATTCH\}$, and D is a set of data elements (the concrete elements of D are not important, and will vary with the application domain). We represent the function of type CPT by its graph, leaving out the attributes that are sent to any, when convenient. ■

Each pattern will have an inbound and an outbound pattern contract, describing the format of the data it is able to receive and send respectively — the role of pattern contracts is to make sure that adjacent inbound and outbound contracts match. The set CPT in a contract represents integration concepts, while the set EL represents data elements and the structure of the message: its headers (HDR, H), its payload (PL, Y) and its attachments ($ATTCH, A$). The contracts determine a node labelling function l_G for the graph.

Example 3.33. A content-based router is not able to process encrypted messages. Recall that its pattern characteristics included a collection of routing conditions: these might require read-only access to message elements such as certain headers h_1 or payload elements e_1, e_2 . Hence the input contract for a router mentioning these message elements is

$$inContr(CBR) = (\{(encrypted, no)\}, \{(HDR, \{h_1\}), (PL, \{e_1, e_2\})\}) .$$

Since the Content-based Router forwards the original message, the output contract is the same as the input contract. ■

Definition 3.34. Let $(C, E) \in 2^{CPT} \times 2^{EL}$ be a pattern contract, and $X \subseteq CPT \times 2^{EL}$ a set of pattern contracts. Write $X_{CPT} = \{C' \mid (\exists E') (C', E') \in X\}$ and $X_{EL} = \{E' \mid (\exists C') (C', E') \in X\}$. We say that (C, E) *matches* X , in symbols $match((C, E), X)$, if the following condition holds:

$$\begin{aligned} & (\forall (p, x) \in C) (x = \text{any} \vee (\forall C' \in X_{CPT}) (\exists (p', y) \in C') \\ & \quad (p = p' \wedge (y = \text{any} \vee y = x))) \wedge \\ & (\forall (m, Z) \in E) (Z = \bigcup_{(m, Z') \in \cup X_{EL}} Z') . \end{aligned}$$

■

We are interested in an inbound contract K_{in} matching the outbound contracts K_1, \dots, K_n of its predecessors. In words, this is the case if (i) for all integration concepts that are important to K_{in} , all contracts K_i either agree, or at least one of K_{in} or K_i accepts any value (*concept correctness*); and (ii) together, K_1, \dots, K_n supply all the message elements that K_{in} needs (*data element correctness*).

Since pattern contracts can refer to arbitrary message elements, a formalization of an integration pattern can be quite precise. On the other hand, unless care is taken, the formalization can easily become specific to a particular pattern composition. In practice, it is often possible to restrict attention to a small number of important message elements (see examples below), which makes the formalization manageable.

Putting everything together, we formalize pattern compositions as integration pattern typed graphs with pattern characteristics and inbound and outbound pattern contracts for each pattern:

Definition 3.35. An *integration pattern contract graph* (IPCG) is a tuple

$$(P, E, type, char, inContr, outContr) ,$$

where $(P, E, type)$ is an IPTG, $char : P \rightarrow 2^{PC}$ is a pattern characteristics assignment, and $inContr : \prod_{p \in P} (2^{CPT} \times 2^{EL})^{|p|}$ and $outContr : \prod_{p \in P} (2^{CPT} \times 2^{EL})^{|p|}$ are pattern contract assignments — one for each incoming and outgoing edge of the pattern, respectively — called the

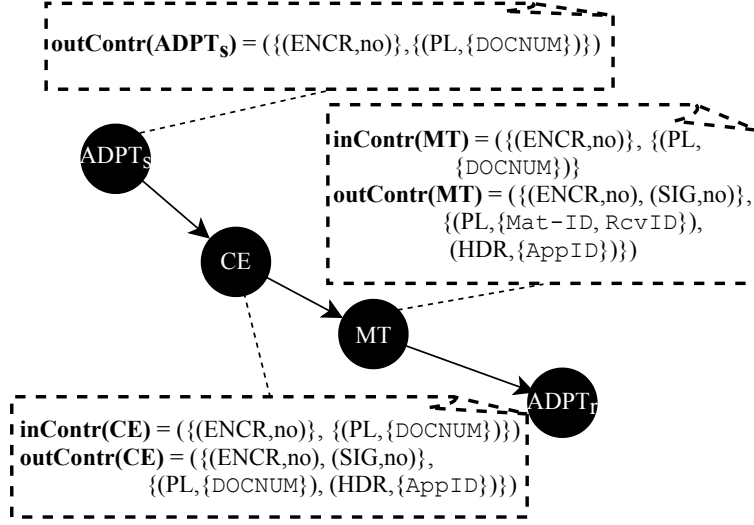


Figure 3.34: IPCG from the material replication scenario

inbound and outbound contract assignment respectively. It is *correct*, if the underlying IPTG $(P, E, type)$ is correct, and inbound contracts match the outbound contracts of the patterns' predecessors, i.e.

$$(\forall p)(p = \text{start} \vee \text{match}(\text{inContr}(p), \{\text{outContr}(p') \mid p' \in \bullet p\})) .$$

Two IPCGs are *isomorphic* if there is a bijective function between their patterns that preserves edges, types, characteristics and contracts. ■

Example 3.36. Figure 3.34 shows the IPCG representing the material replication scenario from Figure 3.23 with a focus on the pattern contracts. The input contract $\text{inContr}(CE)$ of the content enricher pattern CE requires a non-encrypted message and a payload element $DOCNUM$. The content enricher makes a query to get an application ID $AppID$ from the message, and appends it to the message header. Hence the output contract $\text{outContr}(CE)$ contains $(HDR, \{AppID\})$. The content enricher then emits a message that is not encrypted or signed. A subsequent message translator MT requires the same message payload, but does not care about the appended header. It adds another payload $RcvID$ to the message. Comparing inbound and outbound pattern contracts for adjacent patterns, we see that this is a correct IPCG. ■

3.2.3 Timed Db-Nets with Boundaries and Synchronization

The definition of integration pattern contract graphs allows for a structurally correct composition of patterns in the form of abstracting the low-level pattern logic by high-level graphs, in which each pattern denotes a node in the graph. The pattern contracts are implicitly represented by the edges, which enforce the correct composition of adjacent patterns. While the graph approach conveniently abstracts from the internal pattern logic, and thus from unnecessary complexity from a user perspective, the semantic correctness is not yet covered. As motivated before, for the problem of formalizing pattern compositions, we propose a solution grounded on timed db-nets.

Pattern contracts are represented as what we call *boundaries*. Similar to the contracts in IPCGs, these boundaries encode pattern characteristics for correct compositions, however, only this time accounting for the inner pattern semantics.

In the PN domain, *Open Nets* [BBGM15, BM18] are usually considered for compositional contracts, and thus taken up in the workflow (e.g., [vDA02, KMR00]) and service interaction (e.g., [vDALM⁺10]) domains. These approaches focus on the composition of the control flow and correctness, e.g., in the sense of the accordance of private and public views in an interaction [vDA02]. In case of the integration patterns, not only the data flow, but also the identified structural and semantic pattern characteristics have to be taken into account, when formalizing compositions and checking their correctness. For that, we selected the work by Sobociński [Sob10] on nets with boundaries – essentially a sufficiently expressive variant of open nets — for our case. Subsequently, we discuss the background on nets with boundaries for better understandability close to our contribution, timed db-nets with boundaries or open timed db-nets, which follow immediately afterwards.

Nets with Boundaries

The composition of PNs of the same type (e.g., timed db-nets) can be facilitated by “ports” or “open boundaries” for communicating with each other (e.g., [BBGM15, BM18]). Thereby PN tokens can progress from one PN to another through these ports, which can have a structural boundary configuration that allows for their correctness checking, called *synchronization* [Sob10, Fon16]. The subsequently introduced nets are mostly based on [Sob10].

Let $\underline{k}, \underline{l}, \underline{m}, \underline{n}$ range over finite ordinals $n = \{0, 1, \dots, n-1\}$. Intuitively, the notion of a left $\bullet-$ and a right $-^\bullet$ *boundary* is introduced in Definition 3.37 that extends the already known predecessor and successor relations, here made explicit by $^\circ-$, $-^\circ$, respectively. Being explicit about these relations allows for a differentiated visualization of the different conventional control and new boundary places.

Definition 3.37 (Net with boundaries [Sob10]). Let $m, n \in \mathbb{N}$. A (finite) net with boundaries $N : m \rightarrow n$, is a sextuple $(P, T, {}^\circ-, -^\circ, \bullet-, -^\bullet)$ where:

- P is a set of places;
- T is a set of transitions;
- $^\circ-, -^\circ : T \rightarrow 2^P$ are predecessor and successor functions;
- $\bullet-, -^\bullet : T \rightarrow 2^{\underline{m}}, -^\bullet : T \rightarrow 2^{\underline{n}}$ are boundary functions. ■

We say that m and n denote the left and right boundaries of N , respectively. Further, a homomorphism $f : N \rightarrow M$ between two nets with equal boundaries $N, M : m \rightarrow n$ is a pair of functions $f_T : T_N \rightarrow T_M$, $f_P : P_N \rightarrow P_M$, s.t. $^\circ_{-N}; 2^{f_P} = f_T; {}^\circ_{-M}, -^\circ_M; -^\circ_M, \bullet_{-N} = f_T; -^\bullet_M$ and $\bullet_{-N} = f_T; -^\bullet_M$. If its two components are bijections, a homomorphism is an isomorphism. We write $N \cong M$, if there is an isomorphism from N to M .

For contract situations, Sobcinski [Sob10] introduces an independence of transitions, i.e., transitions t, u are independent, if $^\circ t \cap {}^\circ u = \emptyset$. Moreover, for some transition t , a place p can be both in $^\circ t$ and ${}^\circ t$. We extend the notion of independence of transitions to nets with boundaries by: $t, u \in T$ are *independent*, if:

$$^\circ t \cap {}^\circ u = \emptyset, {}^\circ t \cap {}^\circ u = \emptyset, \bullet t \cap \bullet u = \emptyset \text{ and } t^\bullet \cap u^\bullet = \emptyset.$$

To define composition along the boundary of two nets $M : l \rightarrow m$ and $N : m \rightarrow n$, we introduce the concept of a *synchronization*: a pair (U, V) , with $U \subseteq T_M$ and $V \subseteq T_N$ mutually independent sets of transitions $U^\bullet = \{u^\bullet | u \in U\}$ and $^\bullet V = \{^\bullet v | v \in V\}$, s.t.:

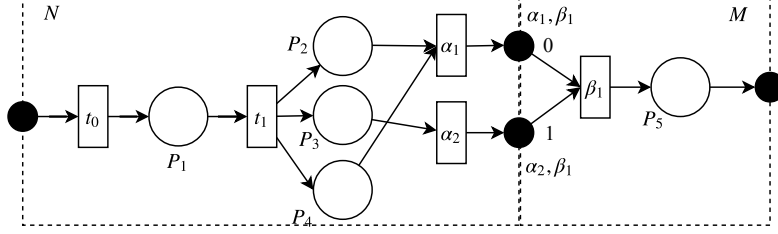


Figure 3.35: Example net with boundaries

- $U \cup V \neq \emptyset$;
- $U^\bullet = \bullet V$.

A set of synchronisations inherits an ordering from the subset relation, i.e., $(U', V') \subseteq (U, V)$, if $U' \subseteq U$ and $V' \subseteq V$. The synchronization is *minimal*, if it is minimal with respect to this order:

$$T_{M;N} \stackrel{\text{def}}{=} \{(U, V) | U \subseteq T_M, V \subseteq T_N, (U, V) \text{ a minimal synchronization}\}.$$

Notably, any transition t in M (or N) that is not connected to a shared boundary m denotes a minimal synchronization $(\{t\}, \emptyset)$.

We define $^\circ -, -^\circ : T_{M;N} \rightarrow 2^{P_M \sqcup P_N}$, with the disjoint union \sqcup , by $^\circ(U, V) = {}^\circ U \cup {}^\circ V$, $(U, V)^\circ = U^\circ \cup V^\circ$, as well as $^\bullet -: T_{M;N} \rightarrow 2^l$ by $^\bullet(U, V) = {}^\bullet U$ and $-^\bullet : T_{M;N} \rightarrow 2^n$ by $(U, V)^\bullet = V^\bullet$. Consequently, the *composition* of M and N is written $M;N : l \rightarrow n$ with:

- set of transitions $T_{M;N}$;
- set of places $P_M \sqcup P_N$;
- $^\circ -, -^\circ : T_{M;N} \rightarrow 2^{P_M \sqcup P_N}$, $^\bullet -: T_{M;N} \rightarrow 2^l$

Proposition 3.38.

- Let $M, M' : k \rightarrow n$ and $N, N' : n \rightarrow m$ be nets with $M \cong M'$ and $N \cong N'$, then $M;N \cong M';N'$;
- Let $L : k \rightarrow l, M : l \rightarrow m, N : m \rightarrow n$ be nets, then $(L;M);N \cong L;(M;N)$.

Further we define the *tensor product* as another binary operation on nets with boundaries. For nets $M : k \rightarrow l$ and $N : m \rightarrow n$ the resulting tensor product is the net that results from their parallel composition. More precisely, $M \otimes N : k \sqcup m \rightarrow l \sqcup n$ in the net with:

- set of transitions $T_M \sqcup T_N$;
- set of places $P_M \sqcup P_N$;
- $^\circ -, -^\circ, {}^\bullet -, -^\bullet$ defined in the obvious way.

Example 3.39. The composition $N;M$ of a nets with boundaries $M : 1 \rightarrow 2$ and $N : 2 \rightarrow 1$ is shown in Figure 3.35. Thereby, $T_N = \{t_0, t_1, \alpha_1, \alpha_2\}$, $T_M = \{\beta_1\}$ and $P_N = \{P_1, \dots, P_4\}$, $P_M = \{P_5\}$. The non-empty values of $^\circ -$ and $-^\circ$ are: $t_0^\circ = \{P_1\}$, ${}^\circ t_1 = \{P_1\}$, $t_1^\circ = \{P_2, P_3, P_4\}$, ${}^\circ \alpha_1 = \{P_2, P_4\}$, ${}^\circ \alpha_2 = \{P_3\}$, and $\beta_1^\circ = \{P_5\}$; and the non-empty values of $^\bullet -, -^\bullet$: $\alpha_1^\bullet, {}^\bullet \beta_1 = \{0\}$, $\alpha_2^\bullet, {}^\bullet \beta_1 = \{1\}$. ■

Timed DB-Nets with Boundaries

We recall timed db-nets from [Section 3.1](#) and describe them as nets that are open, in the sense that they have “ports” or “open boundaries” for communicating with the outside world: tokens can be received and sent on these ports [\[Fon16, Sob10\]](#). Similar to nets with boundaries [\[Sob10\]](#), we define a boundary configuration that records what we expect from the external world for the net to be functioning. This is not the most general notion of open timed db-nets, but it is general enough for our purposes.

The boundary configurations so far only concerned the number of “ports” [\[Fon16, Sob10\]](#) for the synchronization of nets. Hence in [Definition 3.40](#), we define a boundary configuration considering the number of ports, but also the data exchanged between nets.

Definition 3.40 (Boundary configuration). Let \mathfrak{D} be a type domain and $\mathcal{L} = (Q, A)$ a data layer over it. A *boundary configuration* over $(\mathfrak{D}, \mathcal{L})$ is an ordered finite list of colors

$$c \in \{\mathcal{D}_1 \times \dots \times \mathcal{D}_m \mid D_i \in \mathfrak{D}\}$$

We write such a list as $c_1 \otimes \dots \otimes c_n$, and I for the empty list. ■

The length of a boundary configuration list gives the number of “open ports” of the boundary [\[Fon16, Sob10\]](#). Each color c in the list describes the type of the data to be sent/received on the port. An open timed db-net has a left and a right boundary (similar to inbound and outbound contracts of a pattern in IPCG), both described by boundary configurations.

The boundary configurations allow for the extension of the timed db-net definition in [Definition 3.15](#) with boundaries concerning the data exchange to timed db-nets with boundaries given in [Definition 3.41](#).

Definition 3.41 (Timed db-net with boundaries). Let \mathfrak{D} , \mathcal{P} , and \mathcal{L} be a type domain, a persistence layer and a data layer respectively, and let $\otimes_{i < m} c_i$ and $\otimes_{i < n} c'_i$ be boundaries over \mathfrak{D} , \mathcal{L} . A control layer with left boundary $\otimes_{i < m} c_i$ and right boundary $\otimes_{i < n} c'_i$ is a tuple

$$(P, T, F_{in}, F_{out}, F_{rb}, \text{color}, \text{query}, \text{guard}, \text{action})$$

which is a control layer over \mathcal{L} , except that F_{in} is a flow from $P \uplus \{1, \dots, m\}$ to T , and F_{out} and F_{rb} are flows from T to $P_c \uplus \{1, \dots, n\}$, i.e.,

- $P = P_c \uplus P_v$ is a finite set of places partitioned into control places P_c and view places P_v ,
- T is a finite set of transitions,
- F_{in} is an input flow from $P \uplus \{1, \dots, m\}$ to T (where we assume $\text{color}(i) = c_i$),
- F_{out} and F_{rb} are respectively an output and roll-back flow from T to $P \uplus \{1, \dots, n\}$ (where we assume $\text{color}(j) = c'_j$),
- color is a color assignment over P (mapping P to a Cartesian product of data types),
- query is a query assignment from P_v to Q (mapping the results of Q as tokens of P_v),
- guard is a transition guard assignment over T (mapping each transition to a formula over its input inscriptions), and
- action is an action assignment from T to A (mapping transitions to actions triggering updates over the persistence layer).

We write $(\mathfrak{D}, \mathcal{P}, \mathcal{L}, \mathcal{N}, \tau) : \otimes_{i < m} c_i \rightarrow \otimes_{i < n} c'_i$ for timed db-nets with control layers with the given boundaries, and call such a tuple a *timed db-net with boundaries*. ■

Note that the boundaries are carefully incorporated into the timed db-net definition by solely concerning the exchanged data elements, and thus in particular that a timed db-net with empty boundaries is by definition a timed db-net. We can extend the \otimes operation on colors to nets, by defining $N \otimes N' : \vec{c} \otimes \vec{c}' \rightarrow \vec{d} \otimes \vec{d}'$ for $N : \vec{c} \rightarrow \vec{d}$ and $N' : \vec{c}' \rightarrow \vec{d}'$ to be the two nets N and N' next to each other — this gives a tensor product or “parallel” composition of nets. The point of being explicit about the boundaries of nets is to enable also a “sequential” composition of nets, whenever the boundaries are compatible. In [Definition 3.42](#), we use the notation $X \uplus Y$ for the disjoint union of X and Y , with injections $\text{in}_X : X \rightarrow X \uplus Y$ and $\text{in}_Y : Y \rightarrow X \uplus Y$. For $f : X \rightarrow Z$ and $g : Y \rightarrow Z$, we write $[f, g] : X \uplus Y \rightarrow Z$ for the function with $[f, g](\text{in}_X(x)) = f(x)$ and $[f, g](\text{in}_Y(y)) = g(y)$.

Definition 3.42 (Synchronization). Let $(\mathfrak{D}, \mathcal{P}, \mathcal{L}, \mathcal{N}, \tau) : \otimes_{i < m} c_i \rightarrow \otimes_{i < n} c'_i$ and $(\mathfrak{D}', \mathcal{P}', \mathcal{L}', \mathcal{N}', \tau') : \otimes_{i < n} c'_i \rightarrow \otimes_{i < k} c''_i$ be two timed db-nets with boundaries. We define their composition as

$$(\mathfrak{D} \cup \mathfrak{D}', \mathcal{P} \cup \mathcal{P}', \mathcal{L} \cup \mathcal{L}', \mathcal{N}'', \tau'') : \otimes_{i < m} c_i \rightarrow \otimes_{i < k} c''_i$$

(where union of tuples is pointwise) with

$$\mathcal{N}'' = (P'', T'', F''_{in}, F''_{out}, F''_{rb}, \text{color}'', \text{query}'', \text{guard}'', \text{action}'')$$

where

$$\begin{aligned} P'' &= P \uplus P' \uplus \{x_1, \dots, x_n\} \\ T'' &= T \uplus T' \\ F''_{in}(x, y) &= \begin{cases} F_{in}(p, t) & \text{if } (x, y) = (\text{in}_P(p), \text{in}_T(t)) \\ F'_{in}(p', t') & \text{if } (x, y) = (\text{in}_{P'}(p'), \text{in}_{T'}(t')) \\ F'_{in}(j, t') & \text{if } (x, y) = (x_j, \text{in}_{T'}(t')) \\ \emptyset & \text{otherwise} \end{cases} \\ F''_{out}(x, y) &= \begin{cases} F_{out}(p, t) & \text{if } (x, y) = (\text{in}_P(p), \text{in}_T(t)) \\ F'_{out}(p', t') & \text{if } (x, y) = (\text{in}_{P'}(p'), \text{in}_{T'}(t')) \\ F_{out}(j, t) & \text{if } (x, y) = (x_j, \text{in}_T(t)) \\ \emptyset & \text{otherwise} \end{cases} \\ F''_{rb}(x, y) &= \begin{cases} F_{rb}(p, t) & \text{if } (x, y) = (\text{in}_P(p), \text{in}_T(t)) \\ F'_{rb}(p', t') & \text{if } (x, y) = (\text{in}_{P'}(p'), \text{in}_{T'}(t')) \\ F_{rb}(j, t) & \text{if } (x, y) = (x_j, \text{in}_T(t)) \\ \emptyset & \text{otherwise} \end{cases} \\ \text{color}'' &= [\text{color}, \text{color}', x_i \mapsto c_i] \\ \text{query}'' &= [\text{query}, \text{query}'] \\ \text{guard}''(\text{in}_T(t)) &= \text{guard}(t) \\ \text{guard}''(\text{in}_{T'}(t')) &= \text{guard}'(t') \\ \text{action}'' &= [\text{action}, \text{action}'] \\ \tau'' &= [\tau, \tau']. \end{aligned}$$

■

The composed net consists of the two constituent nets, as well as n new control places x_i for communicating between the nets. These places take their color from the shared boundary.

Remark We only use nets with boundaries to plug together open nets. In other words, we only consider the execution semantics of nets with no boundary, and since these are literally ordinary timed db-nets, we inherit their execution semantics from those. Consequently, the resulting nets preserve essential properties like liveness (cf. [MR17][Theorem 2]) or reachability (cf. Section 3.1.2).

Composition of nets behaves as expected: it is associative, and there is an “identity” net which is a unit for composition. All in all, this means that nets with boundaries are the morphisms of a strict monoidal category [Sel11]:

Lemma 3.43. For any timed db-nets N, M, K with compatible boundaries, we have $N \circ (M \circ K) = (N \circ M) \circ K$, and for each boundary configuration $c_1 \otimes \dots \otimes c_n$, there is an identity net $\text{id}_c : c_1 \otimes \dots \otimes c_n \rightarrow c_1 \otimes \dots \otimes c_n$ such that $\text{id}_c \circ N = N$ and $\text{id}_c \circ M = M$ for every M, N with compatible boundaries. Furthermore, for every N, M, K , we have $N \otimes (M \otimes K) = (N \otimes M) \otimes K$.

Proof. Associativity for both \circ and \otimes is obvious. The identity net for $c_1 \otimes \dots \otimes c_n$ is the net with exactly n places x_1, \dots, x_n , with $\text{color}(x_i) = c_i$. \square

In particular, the lemma implies that we can use a graphical language (e.g., “string diagrams” [Sel11]) to define nets and their compositions (cf. [KMS12]). Moreover, the composition of two open nets results in an open net again (e.g., [vDA02]).

3.2.4 Translating Contract Graphs to Timed DB-nets with Boundaries

The integration pattern contract graphs, denoting a more abstract composition of patterns, and timed db-nets with boundaries both allow for ensuring compatible boundaries. We now specify a translation from the pattern graphs to the nets in a stepwise approach. First, we define how to translate single nodes of a pattern graph to patterns in *timed db-net with boundaries* using a template approach based on the pattern categories, before we extend the translation for the edges. Finally, we show the correct synchronization of these compositions and the correctness of the translation.

Translating Single Patterns

We first assign a timed db-net with boundaries $\llbracket p \rrbracket$ for every node p in a integration pattern contract graph. Which timed db-net with boundaries we construct depends on $\text{type}(p)$. If the cardinality of p is $k : m$, then the timed db-net with boundaries will be of the form $\llbracket p \rrbracket : \bigotimes_{i=1}^k \text{color}_{\text{in}}(p)_i \rightarrow \bigotimes_{j=1}^m \text{color}_{\text{out}}(p)_j$ where the colors $\text{color}_{\text{in}}(p)_i$ and $\text{color}_{\text{out}}(p)_j$ are defined depending on the input and output contracts of the pattern respectively:

$$\begin{aligned} \text{color}_{\text{in}}(p)_i &= \prod_{(k,X) \in \text{inContr}_i(p)_{EL}} (k, \prod_{x \in X} x) \\ \text{color}_{\text{out}}(p)_j &= \prod_{(m,X) \in \text{outContr}_j(p)_{EL}} (m, \prod_{x \in X} x) \end{aligned}$$

This incorporates the data elements of the input and output contracts into the boundary of the timed db-net, since these are essential for the dataflow of the net. In Section 3.2.4, we will also incorporate the remaining concepts from the contracts such as signatures, encryption and encodings into the translation as well as the pattern characteristics. More precisely, the concrete characteristics required for the translation of type $\text{type}(p)$ and pattern category p

from Definition 3.29 (on page 106) are

$$type(p) = \begin{cases} start||end & \{\} \\ fork & \{\} \\ condition & \{ (CND, \{ cnd_1, \dots, cnd_n \}) \} \\ join & \{\} \\ merge & \{ (CND, \{ cnd_{cr}, cnd_{cc} \}), (PRG, prg_{agg}, (v_1, v_2)) \} \\ ext.call & \{ (PRG, prg_1, -) \} \\ mp & \{ (CND, \{ cnd_1, \dots, cnd_m \}), (PRG, prg_1, (v_1, v_2)) \} \\ - & \text{otherwise.} \end{cases}$$

Since the pattern categories subsume all relevant patterns into pattern types of similar patterns, we define the translation for each pattern category. We specify timed db-net with boundary templates for each of the pattern categories with their configurable values.

Start and End Pattern Types We translate a start pattern p_{start} into a timed db-net with boundary $\llbracket p_{start} \rrbracket : I \rightarrow color_{out}(p_{start})$ as shown in Figure 3.36(a). Similarly, Figure 3.36(b) shows the translation of an end pattern p_{end} into a timed db-net with boundary $\llbracket p_{end} \rrbracket : color_{in}(p_{end}) \rightarrow I$. The boundary PN places are denoted by dashed lines, and output places annotated with B for the right boundary and input places by A for the left boundary, respectively.

Non-conditional Fork Pattern Types We translate a non-conditional fork pattern p_{fork} with cardinality $1:n$ to the timed db-net with boundaries $\llbracket p_{fork} \rrbracket : color_{in}(p_{fork}) \rightarrow \bigotimes_{j=1}^n color_{out}(p_{fork})_j$ shown in Figure 3.37(a). The output channel cardinality is adapted according to the IPCG node's outgoing edge cardinality.

Non-conditional Join Pattern Types We translate a non-conditional join pattern p_{join} with cardinality $n:1$ to the timed db-net with boundaries $\llbracket p_{join} \rrbracket : \bigotimes_{j=1}^n color_{in}(p_{join})_j \rightarrow color_{out}(p_{join})$ shown in Figure 3.37(b). As for the fork pattern template, the cardinality is adapted according to the IPCG.

Conditional Fork Pattern Types We translate a conditional fork pattern p_{cfork} of cardinality $1:n$ with conditions $cond_1, \dots, cond_{n-1}$ to the timed db-net with boundaries $\llbracket p_{cfork} \rrbracket :$

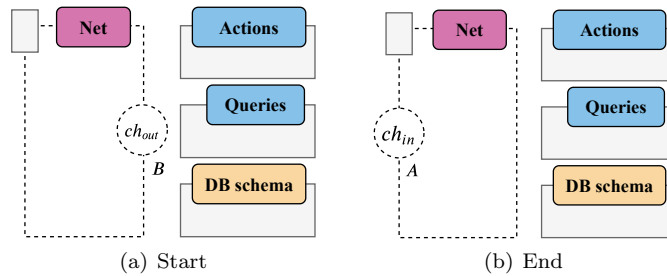


Figure 3.36: Translation templates for the start and end pattern categories

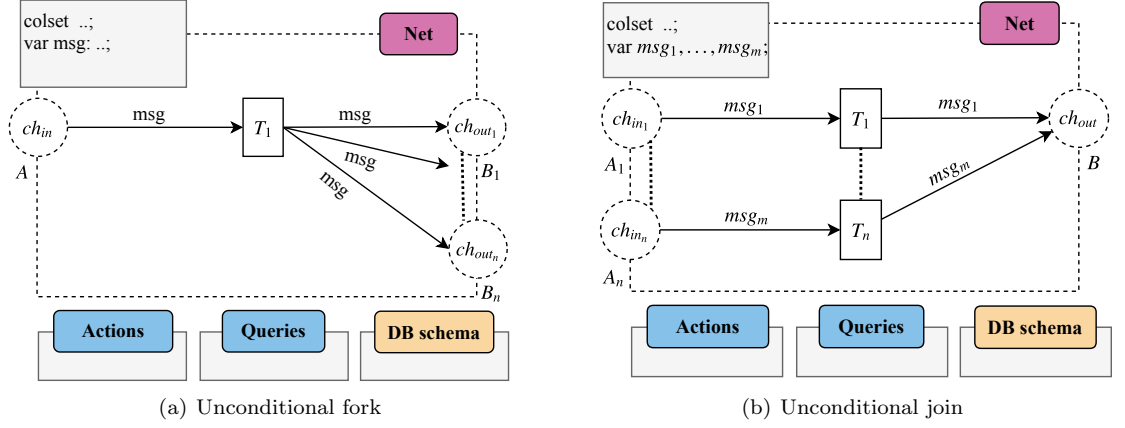


Figure 3.37: Translation templates of unconditional fork and join pattern categories

$\text{color}_{\text{in}}(p_{\text{cfork}}) \rightarrow \bigotimes_{j=1}^n \text{color}_{\text{out}}(p_{\text{cfork}})_j$ shown in Figure 3.38. Note that net is constructed so that the conditions are evaluated in order — the transition corresponding to condition k will only fire if condition k is true, and conditions $1, \dots, k-1$ are false. The last transition will fire if all conditions evaluate to false. The conditions $\{cond_1, \dots, cond_{n-1}\}$ are assigned from the IPCG *CND* characteristic.

Message Processor Pattern Types We translate a message processor pattern p_{mp} with persistent state DB and processor function f to the timed db-net with boundaries $\llbracket p_{\text{mp}} \rrbracket : \text{color}_{\text{in}}(p_{\text{mp}}) \rightarrow \text{color}_{\text{out}}(p_{\text{mp}})$ shown in Figure 3.39. with parameter function $g : \langle EL \rangle \rightarrow \langle TYPE \rangle$, filter condition $h(msg)$, time interval $[\tau_s, \tau_e]$, and with hd the head and tl the tail of a list in CPNs, and INS the insert list function, all parameters in tuple $\langle \cdot \rangle$. We argue that this template covers a wide range of one-to-one patterns like control-only, control-resources, control-time.

Merge Pattern Types We translate a merge pattern p_{merge} with aggregation function f and timeout τ to the timed db-net with boundaries $\llbracket p_{\text{merge}} \rrbracket : \text{color}_{\text{in}}(p_{\text{merge}}) \rightarrow \text{color}_{\text{out}}(p_{\text{merge}})$ shown in Figure 3.40. Briefly, the net works as follows: the first message in a sequence makes transition T_1 fire, which creates a new database record for the sequence, and starts a timer. Each subsequent message from the same sequence gets stored in the database using transition T_2 , until the age of the sequence token is in the time interval $[\tau_1, \tau_2]$, which will fire transition T_3 . Alternatively, the action associated to T_4 will make the condition for the Aggregate transition true, which will retrieve all messages $msgs$ and then put $f(msgs)$ in the output place of the net. Thereby, the characteristics $\{(\{cnd_{cr}, cnd_{cc}\}), (PRG, prg_{agg}, (v_1, v_2))\}$ from the IPCG are assigned to net template: $(v_1, v_2) \rightarrow @(\tau_1, \tau_2), prg_{agg} \rightarrow f(msgs)$. Moreover, the correlation condition $cnd_{cr} \rightarrow g(msg, msgs)$ and the completion condition $cnd_{cc} \rightarrow complCount$ are configured accordingly.

External Call Pattern Types We translate an external call pattern p_{call} to the timed db-net with boundaries $\llbracket p_{\text{call}} \rrbracket : \text{color}_{\text{in}}(p_{\text{call}}) \rightarrow \text{color}_{\text{out}}(p_{\text{call}})_1 \otimes \text{color}_{\text{out}}(p_{\text{call}})_2$ shown in Figure 3.41.

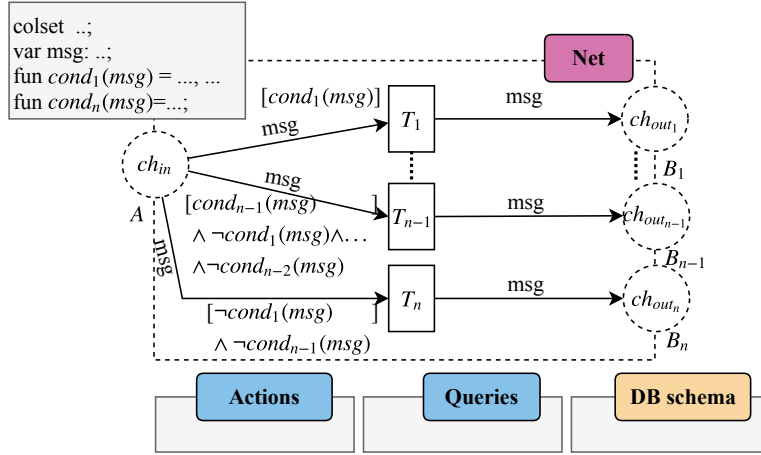


Figure 3.38: Translation of a conditional fork pattern category

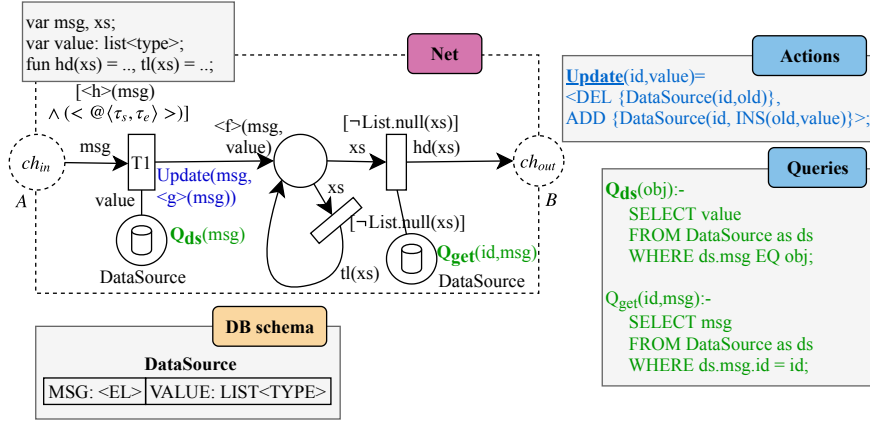


Figure 3.39: Translation of a message processor pattern category

Translating Integration Pattern Contract Graphs

We now show how to translate not just individual nodes from an integration pattern contract graph, but how to also take the edges into account. We first enrich the translations of the single patterns with transitions and guards enabling and enforcing the concepts from the output and input contract respectively, and then prove that composing the translations of individual patterns according to how they are connected in the graph gives rise to a well-formed timed db-net.

Taking Contract Concepts into Account Recall that a pattern contract also represents properties of the exchanged data, e.g., if a pattern is able to process or produce signed, encrypted or encoded data. A message can only be sent from one pattern to another if their contracts match, i.e., if they agree on these properties. To reflect this in the timed db-nets semantics, we enrich all colorsets to also keep track of this information: given a place P with colorset C , we construct the colorset $C \times \{yes, no\}^3$, where the color $(x, b_{sign}, b_{encl}, b_{enc})$ is intended to mean that the data value x is respectively signed, encrypted and encoded or not according to the yes/no

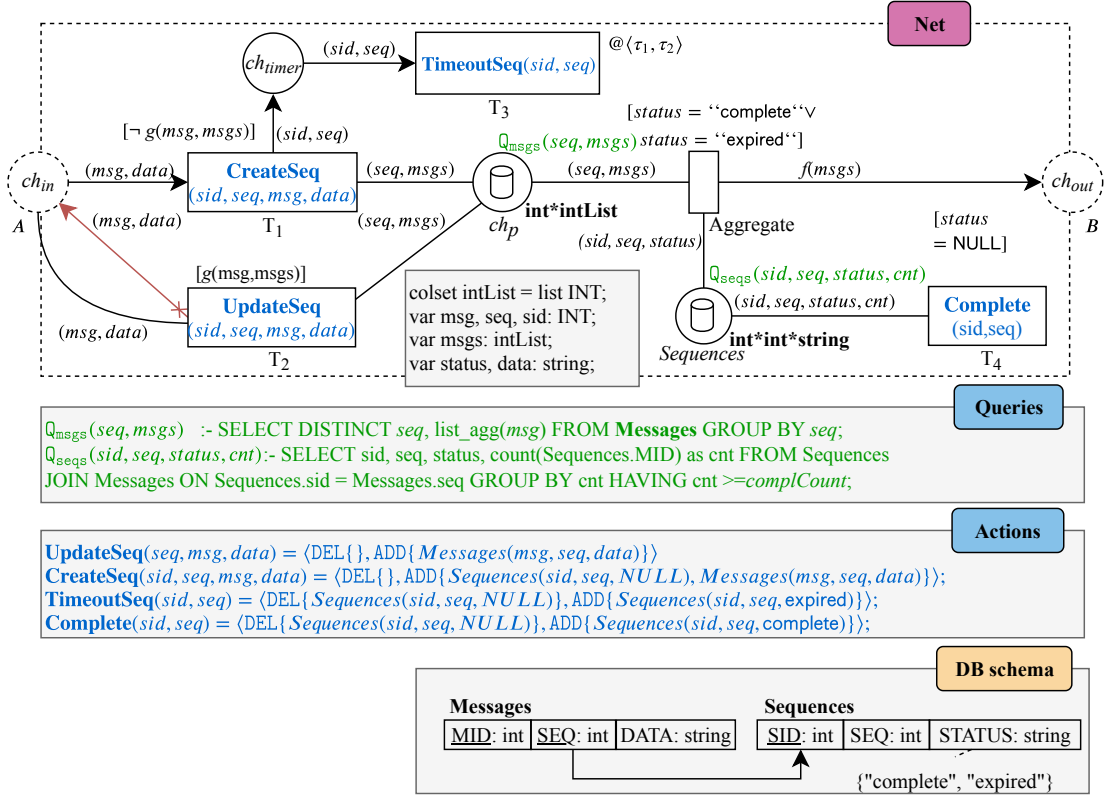


Figure 3.40: Translation of a merge pattern category

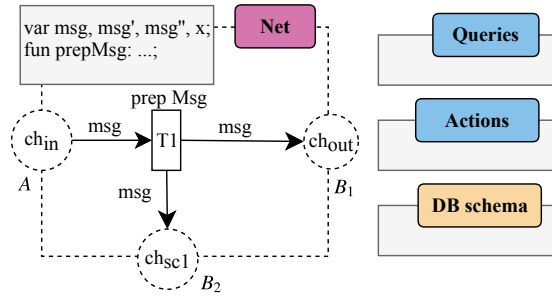


Figure 3.41: Translation of an external call pattern category

values b_{sign} , b_{encr} , and b_{enc} . To enforce the contracts, we also make sure that every token entering an input place c_{in} is guarded according to the input contract by creating a new place ch'_{in} and a new transition from ch'_{in} to ch_{in} , which conditionally copies tokens whose properties match the contract in the form of transition guards. The new place ch'_{in} replaces ch_{in} as an input place. Similarly, for each output place ch_{out} we create a new place ch'_{out} and a new transition from ch_{out} to ch'_{out} which ensures that all tokens satisfy the output contract. The new place ch'_{out} replaces ch_{out} as an output place. Formally, the construction is as follows:

Definition 3.44 (Construction of contract guards). Let $\mathcal{X} = (\mathfrak{D}, \mathcal{P}, \mathcal{L}, \mathcal{N}, \tau) : \otimes_{i < m} c_i \rightarrow \otimes_{i < n} c'_i$ be a timed db-net with boundaries and $\vec{C} = IC_0, \dots, IC_{m-1}, OC_0, \dots, OC_{n-1} \in CPT$. Define the timed db-net with boundaries $\mathcal{X}_{CPT(\vec{C})} = (\mathfrak{D}, \mathcal{P}, \mathcal{L}, \mathcal{N}', \tau) : \otimes_{i < m} (c_i \times \{yes, no\}^3) \rightarrow \otimes_{i < n} (c'_i \times \{yes, no\}^3)$ with

$$\mathcal{N}' = (P', T', F'_{in}, F'_{out}, F'_{rb}, \text{color}', \text{query}, \text{guard}', \text{action}')$$

where

$$\begin{aligned} P' &= P \uplus \{x'_1, \dots, x'_m\} \uplus \{y'_1, \dots, y'_n\} \\ T' &= T \uplus \{t_1, \dots, t_m\} \uplus \{t'_1, \dots, t'_n\} \\ \text{color}' &= [\text{color}'', x'_i \mapsto \text{color}''(x_i), y'_j \mapsto \text{color}''(y_j)] \\ &\quad \text{where } \text{color}''(x) = \text{color}(x) \times \{yes, no\}^3 \text{ with message } x \\ F'_{in}(a, b) &= \begin{cases} F_{in}(p, t) & \text{if } (a, b) = (\text{in}_P(p), \text{in}_T(t)) \\ \{(x, b_{\text{sign}}, b_{\text{encr}}, b_{\text{enc}})\} & \text{if } (a, b) \in \{(x'_i, t_i), (y'_j, t'_j)\} \\ \emptyset & \text{otherwise} \end{cases} \\ F'_{out}(a, b) &= \begin{cases} F_{out}(p, t) & \text{if } (a, b) = (\text{in}_P(p), \text{in}_T(t)) \\ \{(x, b_{\text{sign}}, b_{\text{encr}}, b_{\text{enc}})\} & \text{if } (a, b) = (x'_i, t_i) \\ \{(x, b_{\text{sign}}, b_{\text{encr}}, b_{\text{enc}}) \mid \\ (\forall p)(OC_j)_{CPT}(p) \in \{b_p, any\}\} & \text{if } (a, b) = (y'_j, t'_j) \\ \emptyset & \text{otherwise} \end{cases} \\ F'_{rb}(a, b) &= \begin{cases} F_{rb}(p, t) & \text{if } (a, b) = (\text{in}_P(p), \text{in}_T(t)) \\ \emptyset & \text{otherwise} \end{cases} \\ \text{guard}' &= [\text{guard}, t_i \mapsto \bigwedge_{\{p \mid (IC_i)_{CPT}(p) \neq any\}} y_p = (IC_i)_{CPT}(p), t'_j \mapsto \top] \\ \text{action}' &= [\text{action}, t_i \mapsto -, t'_j \mapsto -] \\ \tau' &= [\tau, t_i \mapsto [0, \infty], t'_j \mapsto [0, \infty]] \end{aligned}$$

■

Intuitively, each message x is annotated in the output flow of a pattern F_{out} with information about the content of x , which can be signed *sign*, encrypted *encr*, or encoded *enc*, according to the output contract OC of a pattern, indicated by values *yes* or *no*. For a subsequent pattern, a transition is created for F_{in} with a guard that is defined according to the input contract IC .

In practice, the pattern contract construction in Definition 3.44, can be realized as template translation on an inter pattern level. The templates in Figure 3.42 and Figure 3.43) denote the translation scheme for the construction of a one-to-one message processor, and a many-to-many pattern category, respectively. In case of the message processor, a token (x, p, q, r) of

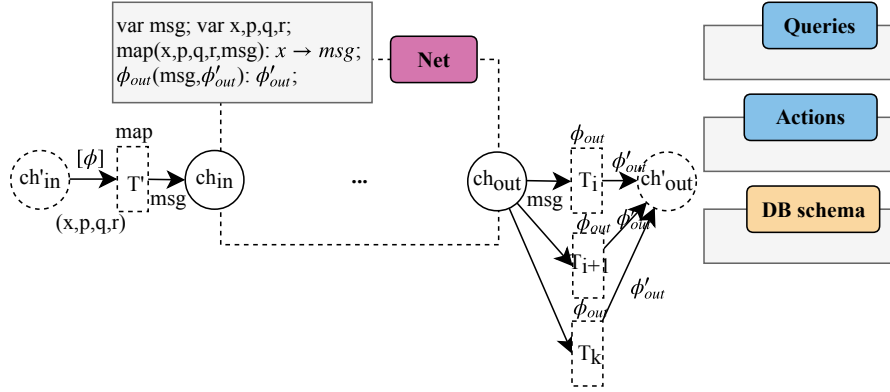


Figure 3.42: 1:1 pattern contract construction

color $(x, b_{\text{sign}}, b_{\text{encr}}, b_{\text{enc}})$ with the actual data x and the signed p , encrypted q , and encoded r information are in the boundary place ch'_{in} and consumed by the boundary transition T' , if the guard ϕ on (p, q, r) is satisfied that is configured precisely according to the contracts of the IPCG pattern. On firing, the transition removes this meta information and emits the actual message $x \rightarrow msg$ to the actual pattern. After the unaltered pattern processing, the boundary transitions T_1, \dots, T_k (i.e., depending on the ANY-logic), supply (p, q, r) information specific for the pattern. The resulting token ϕ'_{out} are then the input for the subsequent pattern. Similarly, the outbound message of the many-to-many category is encoded for each of the o outgoing $ch_{out}, ch_{out}^2, \dots, ch_{out}^o$ as well as the l incoming message channels $ch_{in}, ch_{in}^2, \dots, ch_{in}^l$. The templates for zero-to-one, one-to-zero, one-to-many and many-to-zero boundaries follow the same construction mechanism.

The following lemma is immediate:

Lemma 3.45. If $X : \bigotimes_{i=1}^m c_i \rightarrow \bigotimes_{j=1}^n c'_j$ then $X_{CPT(\vec{C})} : \bigotimes_{i=1}^m c_i \rightarrow \bigotimes_{j=1}^n c'_j$ for every choice of contracts \vec{C} . \square

Example 3.46. Let us consider two examples to gain an understanding of the construction.

Message Translator. Figure 3.44 shows $MT_{CPT(ENC=no, \dots)}$ for a message translator pattern MT with input contract $\{(ENC, any), (ENCR, no), (SIGN, any)\}$ and output contract $\{(ENC, no), (ENCR, no), (SIGN, no)\}$. The input transition T' hence checks the guard $[encr = no]$, and if it matches, the token is forwarded to the actual message translator. After the transformation, the resulting message msg' is not encrypted, the signing is invalid, and not encoded, and thus emits (x, no, no, no) .

Join Router. The join router structurally combines many incoming to one outgoing message channel without accessing the data (cf. $\{(MC, 1 : 1), (ACC, ro), (MG, false), (CND, \emptyset)\}$). While the data format (i.e., the data elements EL) has to be checked during the composition of the boundary, the runtime perspective of the boundary (x, p, q, r) is any for x, p, q in the input and output. Figure 3.45 shows the resulting boundary construction for the join router. The input boundary does not enforce CPT constraints, and thus no guards are defined for the transitions. The output boundary, however, has to deal with the three CPT properties p, q, r set to $\{yes, no\}$, resulting in six different permutations. \blacksquare

Synchronising Pattern Compositions and Correctness of the Translation We are now in a position to define the full translation of a correct integration pattern contract graph G . For

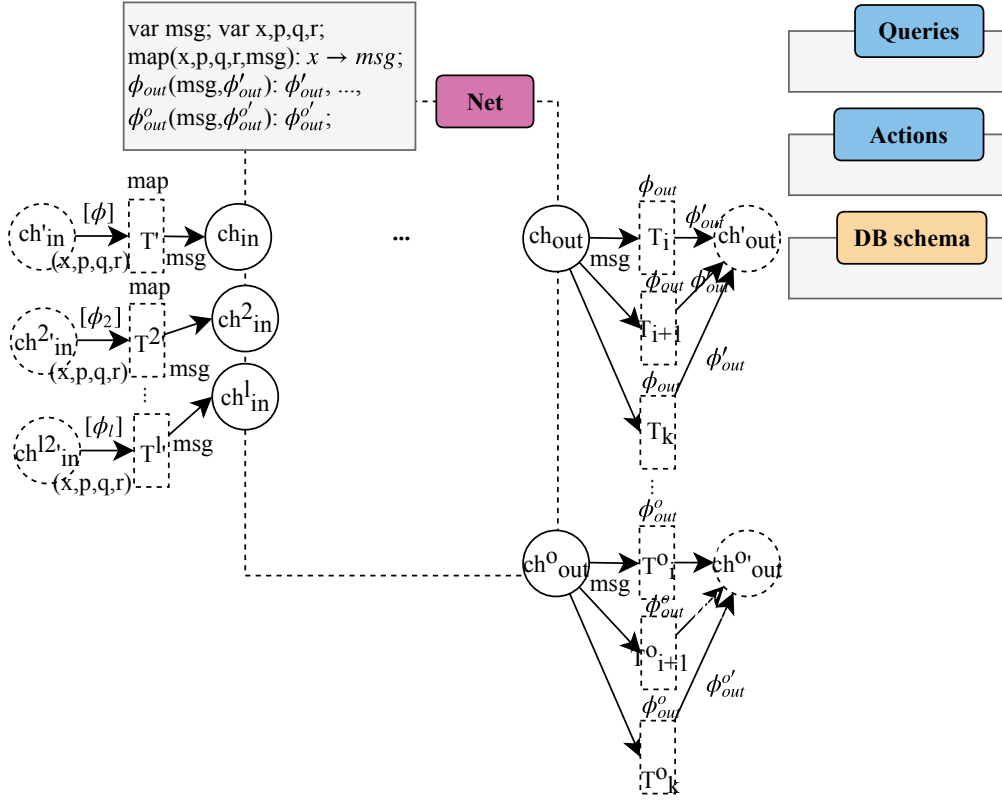


Figure 3.43: l:o pattern contract construction

the translation to be well-defined, we need only data element correctness of the graph. Concept correctness is used to show that in the nets in the image of the translation, tokens can always flow from the translation of the start node to the translation of the end node.

Theorem 3.47. Let a correct integration pattern contract graph G be given. For each node p , consider the timed db-net

$$\llbracket p \rrbracket_{CPT(inContr(p), outContr(p))} : \bigotimes_{i=1}^k \text{color}_{in}(p)_i \rightarrow \bigotimes_{j=1}^m \text{color}_{out}(p)_j$$

Use the graphical language [Sel11] enabled by Lemma 3.43 (on page 114) to compose these nets according to the edges of the graph. The resulting timed db-net is then well-defined, and has the option to complete, i.e., from each marking reachable from a marking with a token in some input place, it is possible to reach a marking with a token in an output place.

Proof. Since the graph is assumed to be correct, all input contracts match the output contracts of the nets composed with it, which by data element correctness means that the boundary configurations match, so that the result is well-defined.

To see that the constructed net also has the option to complete, first note that the interpretations of basic patterns in Section 3.2.4 do (in particular, one transition is always enabled in the translation of a conditional fork pattern in Figure 3.38, and the aggregate transition will always

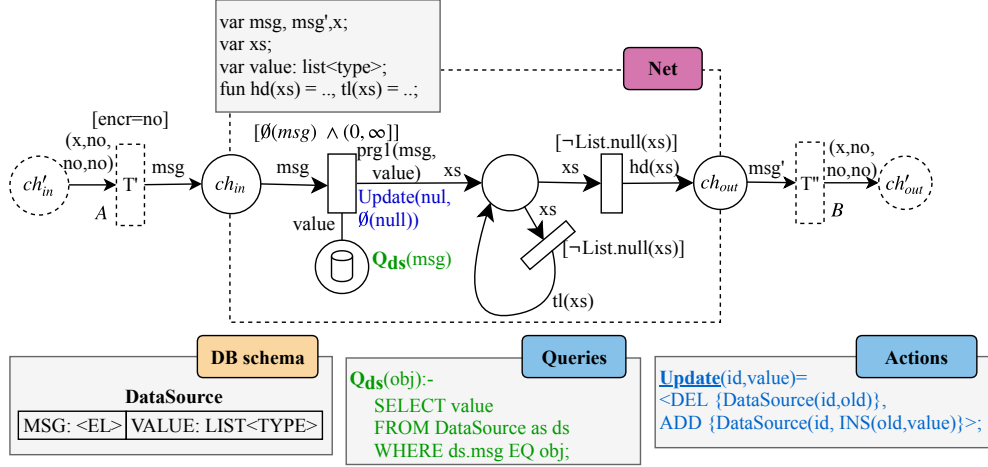


Figure 3.44: Example of a message translator construction

be enabled after the timeout in the translation of a merge pattern in Figure 3.40). By the way the interpretation is defined, all that remains to show is that if N and N' have the option to complete, then so does $N_{CPT(\vec{C})} \circ N'_{CPT(\vec{C}')}$, if the contracts \vec{C} and \vec{C}' match. Assume a marking with a token in an input place of N' . Since N' has the option to complete, a marking with a token in an output place of N' is reachable, and since the contracts match, this token will satisfy the guard imposed by the $N_{CPT(\vec{C})}$ construction. Hence a marking with a token in an input place of N is reachable, and the statement follows, as N has the option to complete. \square

3.2.5 Evaluation: Case Studies

We evaluate the translation in two case studies of real-world integration scenarios: the replicate material scenario from Figure 3.23, and the predictive machine maintenance scenario from Figure 3.26. The former is an example of hybrid integration, and the latter of IoT device integration. Our aim is to observe different aspects of the following hypotheses (according to **RQ2-2(c)**).

- H1** The integration pattern contract graphs allow for structurally correct compositions.
- H2** The execution semantics of the translated timed db-net with boundaries is consistent.

For each of the scenarios, we give an integration pattern contract graph with matching contracts (\rightarrow **H1**), translate it to a timed db-net with boundaries, and show how its execution can be simulated (\rightarrow **H2**). The scenarios are both taken from the SAP Cloud Platform Integration solution catalog of reference integration scenarios, and are frequently used by customers [SAP18a]. For the simulation we use the CPN Tools timed db-net prototype from Section 3.1.3 with the extension for hierarchical PN composition. In CPN Tools hierarchies, the patterns can be represented as sub-groups and pages with explicit in- and out-port type definitions [JKW07], which we use as part of the boundaries defined in Definition 3.41. Thereby the synchronization is checked based on the CPN color sets of the port types. The other boundary checks are performed during the simulation according to the constructed boundaries (see construction mechanism in Definition 3.44).

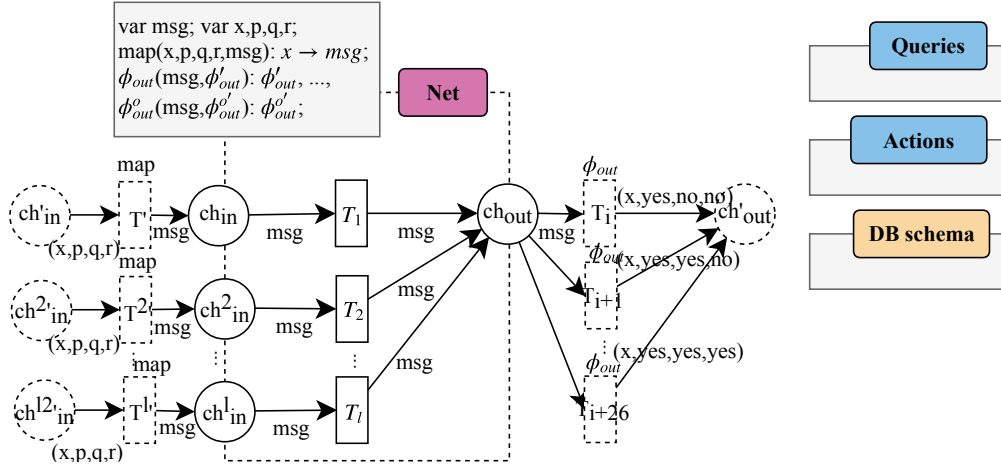


Figure 3.45: Join router construction

Hybrid Integration: Replicate Material

An IPCG representing an integration process for the replication of material from an enterprise resource planning or customer relationship management system to a cloud system was given in Figure 3.34 (on page 109). We now add slightly more data in the form of the pattern characteristics, which provides sufficient information for the translation to timed db-nets with boundaries. Figure 3.46 depicts the enriched IPCG. The adapters are actually message processors, however, for simplicity they are represented as start and end pattern types, $ADPT_s$ denoting *erp* and $ADPT_r$ representing *cod*. The characteristics of the *CE* node includes the tuple $(PRG, (prg1, [0, \infty)))$, with enrichment function *prg1* which assigns the *DOCNUM* payload to the new header field *AppID*. Similarly, the characteristics of the *MT* nodes includes a tuple $(PRG, (prg2, -))$ with mapping program *prg2*, which maps the *EDLDC40-DOCNUM* payload to the *MMRR-BMH-ID* field (the Basic Message Header ID of the Material Mass Replication Request structure), and the *EPM-PRODUCT_ID* payload to the *MMRR-MAT-ID* field (the Material ID of the Material Mass Replication Request structure). The pattern composition is correct according to Definition 3.35 (on page 108) also with these refined pattern characteristics, which shows that hypothesis **H1** is fulfilled for the material replicate scenario.

Translation to a Timed DB-Nets with Boundaries First we translate each single pattern from Figure 3.46 according to the construction in Section 3.2.4. The integration adapter nodes $ADPT_s$ and $ADPT_r$ are translated as the start and end patterns in Figure 3.36(a) and Figure 3.36(b), respectively. The content enricher *CE* node and message translator *MT* node are message processors without storage, and hence translated as in Figure 3.39 with $\langle f \rangle_{CE} = prg1$ and $\langle f \rangle_{MT} = prg2$ (no database values are required). Since no database table updates are needed for either translation, the database update function parameter $\langle g \rangle$ can be chosen to be the identity function in both cases.

In the second step, we refine the timed db-net with boundaries to also take contract concepts into account by the construction in Definition 3.44. The resulting net is shown in Figure 3.47. This ensures the correctness of the types of data exchanged between patterns, and follows directly from the correctness of the corresponding IPCG. Other contract properties such as encryption *encr*, encodings *enc*, and signatures *sign* are checked through transition guards.

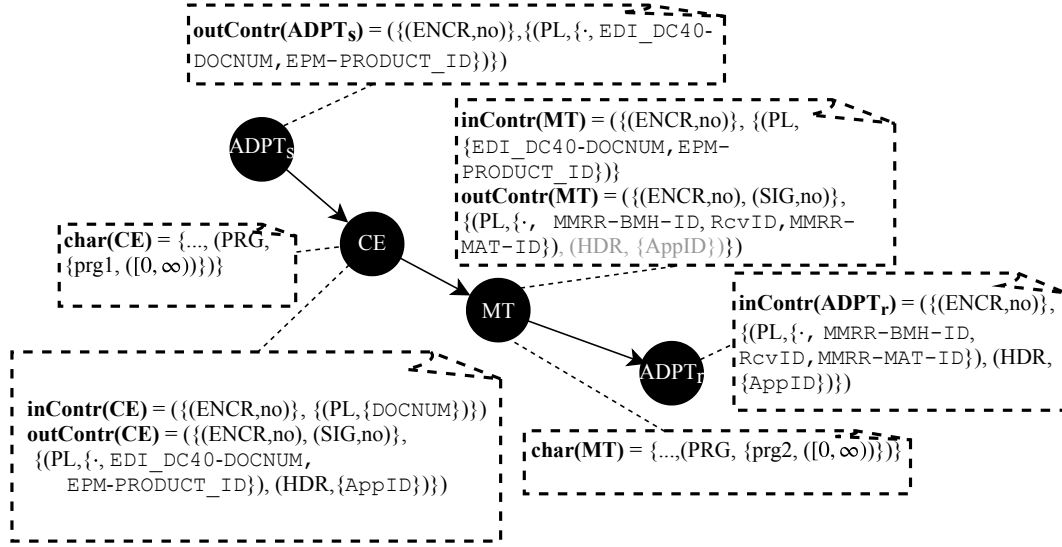


Figure 3.46: Complete integration pattern contract graph of the replicate material scenario

Simulation We test the correctness of the composition construction of the material replicate scenario in Figure 3.47 through simulation in the form of a hierarchical timed db-net model, shown in Figure 3.48. Thereby, the Content Enricher and Message Translator patterns are represented by CPN Tool Subpage elements that are annotated with subpage tags *enricher*, *translator*, respectively.

On arrival of the request *msg* from the ERP system, the boundary configuration is appended to the message in place *erpToCe*. In the replicate material scenario the data is received unencrypted, uncoded and unsigned, leading to a boundary (*msg,no,no,no*), or in our prototype encoded as (*msg,false,false,false*). The extended message *erp_msg* is then moved to the boundary place *ch0* by transition *CheckCeBoundary*, if the *[encr=false]* guard holds, and thus ensures the correctness of the data exchange between patterns. Subsequently only the actual message without the boundary data is moved to place *ch0*, that is linked to the input place *ch0* of the enricher, as in Figure 3.25. We recall, that the *in* port type ensures that the synchronization on the CPN color set level are correct. After the *enricher* processing, the *out* port type ensures the correctness of the synchronization on the CPN color set level and the resulting message *emsg* is moved to the linked output place *ch4*. The constructed outbound boundary, represented by transition *SetCeBoundary* sets the boundary properties of the enricher to (*msg,false,false,false*) for the following pattern. On the input boundary side of the *translator*, transition *CheckMtBoundary* evaluates its guard, before moving the message without the boundary data to the boundary place *ch5*, which proceeds similar to the enricher.

Note that our boundary construction mechanism from Definition 3.44 generated the input boundary, e.g., denoted by place *erpToCe* and transition *CheckCeBoundary*, as well as the output boundary, e.g., transition *SetCeBoundary* and place *ceToMt*, including the transition guards, colorsets, variables, and port type configurations, for the validation by simulation from Section 3.1.2.

Discussion Notably, constructing an IPCG requires less technical knowledge such as particularities of timed db-nets but still enables correct pattern compositions on an abstract level. While

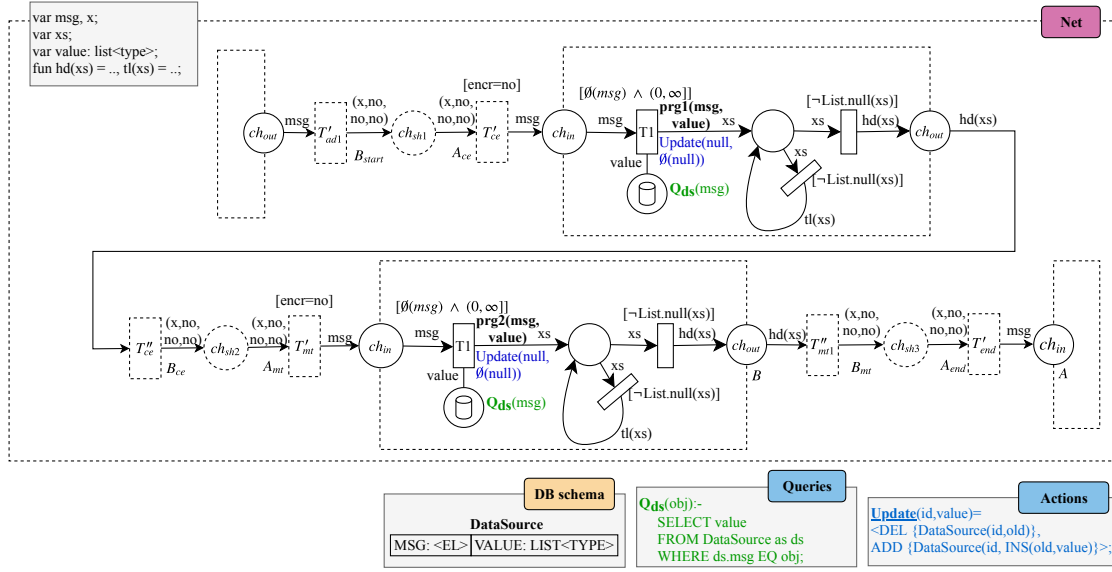


Figure 3.47: Material replicate scenario as a timed db-nets with boundaries

the *CPT* part of the pattern contracts (e.g., encrypted, signed) could be derived and translated automatically from a scenario in a yet to be defined modeling language, many aspects like their elements *EL* as well as the configuration of the characteristics by enrichment and mapping programs requires a technical understanding of IPCGs and the underlying scenarios. As such IPCGs can be considered a suitable intermediate representation of pattern compositions. The user might still prefer a more appealing graphical modeling language on top of IPCGs. The simulation capabilities of the constructed timed db-net with boundaries allow for the experimental validation of the composition correctness of real-world pattern compositions. However, the complexity of the construction highlights the importance of an automation of the construction.

Conclusions. (1) IPCG and timed db-net with boundaries are correct with respect to composition and execution semantics ($\rightarrow H1, H2$); (2) timed db-net with boundaries are even more complex than timed db-net; (3) IPCGs are more comprehensible than timed db-net, and expressive enough for current integration scenarios.

Internet of Things: Predictive Maintenance and Service (PDMS)

The IPCG representing the predictive maintenance create notification scenario that connects machines with enterprise resource planning (ERP) and PDMS systems was given in Figure 3.27. We add all pattern characteristics and data, which provides sufficient information for the translation to timed db-nets with boundaries. Figure 3.49 depicts the corresponding IPCG. The characteristics of the CE_1 node includes an enrichment function $prg1$ that adds further information about the machine in the form of the *FeatureType* to the message that contains machine *ID* and *UpperThresholdWarningValue*. This data is leveraged by the UDF_1 *predict* node, which uses a prediction function $prg2$ about the need for maintenance and adds the result into the *MaintenanceRequestById* field. Before the data is forwarded to the ERP system (simplified by an *End*), the single machine predictions are combined into one message by the AGG_1 node with correlation cnd_{cr} and completion cnd_{cc} conditions as well as the aggregation function $prg3$ and completion timeout (v_1, v_2) as pattern characteristics $\{(\{cnd_{cr}, cnd_{cc}\}), (PRG, prg_4, (v_1, v_2))\}$.

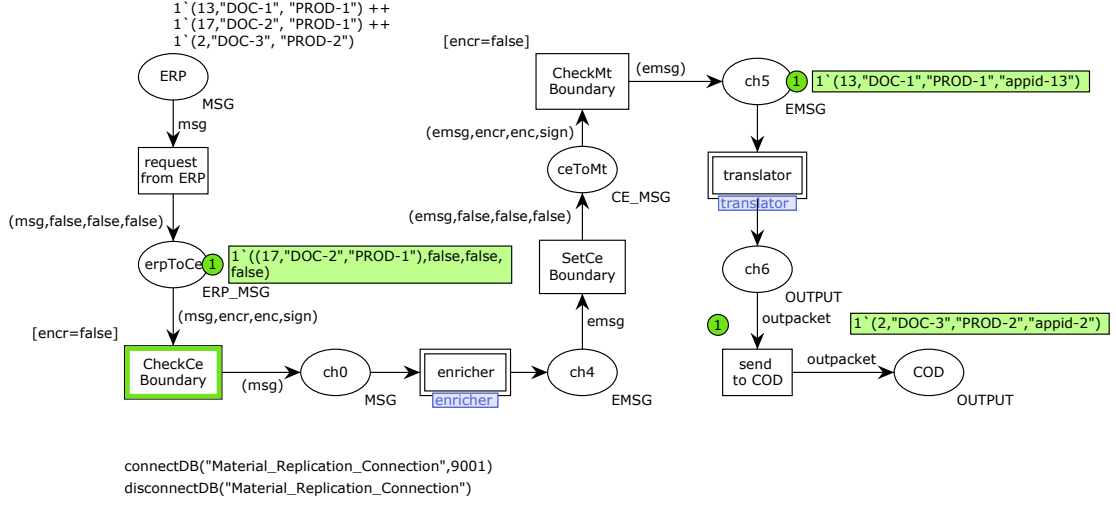


Figure 3.48: Material replicate scenario simulation

The pattern composition is correct according to [Definition 3.35](#) also with these refined pattern characteristics, which shows that hypothesis *H1* is fulfilled for the material replicate scenario.

Translation to Timed DB-Nets with Boundaries Again, we translate each single pattern from [Figure 3.49](#) according to the construction in [Section 3.2.4](#). The *Start* and *End* nodes are translated as the start and end pattern in [Figure 3.36\(a\)](#) and [Figure 3.36\(b\)](#) respectively. The Content Enricher CE_1 and user-defined function UDF_1 nodes are message processors, and hence translated as in [Figure 3.39](#) with $\langle f \rangle_{CE_1} = prg1$ and $\langle f \rangle_{UDF_1} = prg2$. Since no table updates are needed for either translation, the database update function parameter $\langle g \rangle$ can be chosen to be the identity function in all cases. The Aggregator AGG_1 node is a merge pattern type, and thus translated as in [Figure 3.40](#) with $(v_1, v_2) \rightarrow [\tau_1, \tau_2]$, $prg_{agg} \rightarrow f(msgs)$. Moreover, the correlation condition $cnd_{cr} \rightarrow g(msg, msgs)$ and the completion condition $cnd_{cc} \rightarrow complCount$.

In the second step, we refine the timed db-net with boundaries to also take contract concepts into account by the construction in [Definition 3.44](#). The resulting net is shown in [Figure 3.50](#). This ensures the correctness of the types of data exchanged between patterns, and follows directly from the correctness of the corresponding IPCG. Other contract properties such as encryption, signatures, and encodings are checked through the transition guards.

Simulation We illustrate the correctness of the composition construction of the predictive maintenance scenario in [Figure 3.50](#) through simulation in the form of a hierarchical timed db-net model, shown in [Figure 3.51](#). Again, all timed db-net patterns are hierarchically represented by CPN Tool Subpage elements that are annotated with subpage tags *enricher*, *message_aggregator*, respectively, and the user-defined function *predict* is denoted by a transition. The boundaries are constructed from [Figure 3.50](#) by inserting *SetPdmsBoundary* and *pdmsToCe* as output boundary of *get report*, which matches the input boundary of the subsequent *enricher*, denoted by the *CheckCeBoundary* transition. Transition *SetCeBoundary* and place *ceToPredict* represent the output boundary of the enricher, which again match the input boundary of the *predict* user-defined function pattern through transition *CheckPredictBoundary*. Finally, the output boundary of the predict step is ensured by transition *SetPredictBoundary* and place *predictToAgg*. Again, it can

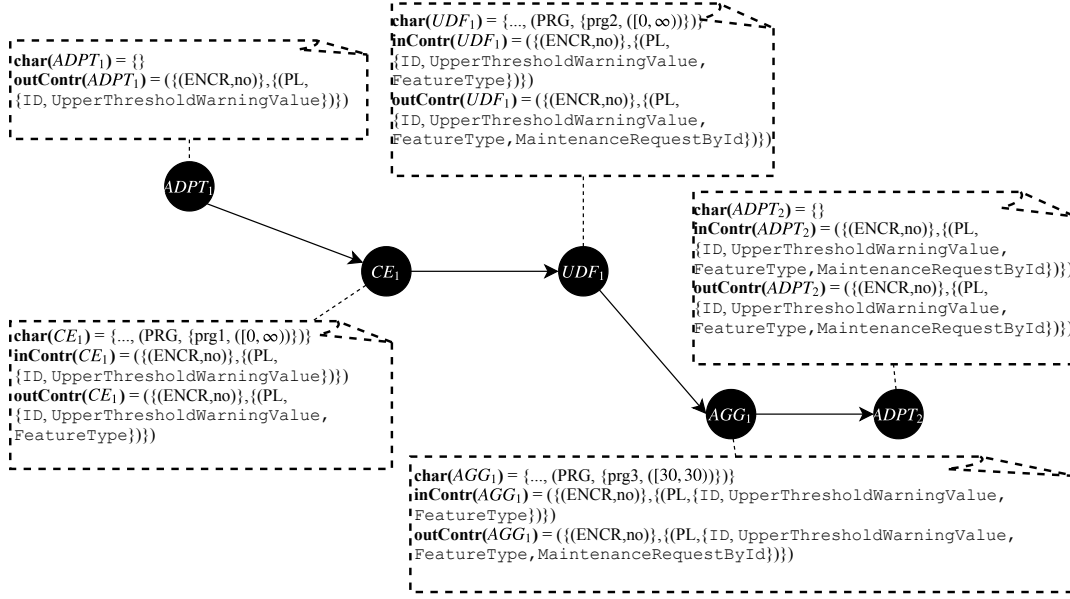


Figure 3.49: Integration pattern contract graph of the predictive maintenance scenario

be easily seen that the input boundary of the aggregator in the form of the *CheckAggBoundary* transition matches, and thus the overall composition is correct. Consequently, the simulation of the timed db-net with these boundaries in Figure 3.51 results in the same, correct output with the timed db-nets without boundaries in Figure 3.29.

Discussion In this slightly more complex scenario, it becomes more obvious that the constructed IPCGs are quite technical as well and require a careful construction of pattern characteristics and contracts. While this seems to be an ideal representation for checking the structural correctness of compositions, this should be no manual task for a user. Especially for more complex scenarios, we found that the re-configurable pattern type-based translation works well. However, the construction of the timed db-nets with boundaries corresponding to an IPCG would benefit from an automatic translation (e.g., through tool support).

Conclusions. (4) IPCGs are still quite technical, especially for more complex scenarios; (5) a tool support for automatic construction and translation is preferable.

3.2.6 Conclusions

To formalize pattern compositions as the foundations of current EAI scenarios, we followed the approach in Figure 3.32 (on page 104). We considered the structural and semantic pattern characteristics to specify an Integration Pattern Typed Graph (IPTG), representing the pattern's internal data flow, and extended the IPTG by inter-pattern composition correctness criteria based on the inter-pattern data flow, called Integration Pattern Contract Graph (IPCG). The contract graphs provide a rich composition context, which might help the user when composing patterns. Moreover, the contract graphs have built-in structural composition correctness checking capabilities based on matching pattern contracts (cf. research question RQ2-2(a): “How can pattern compositions be suitably formalized for compositional correctness?”).

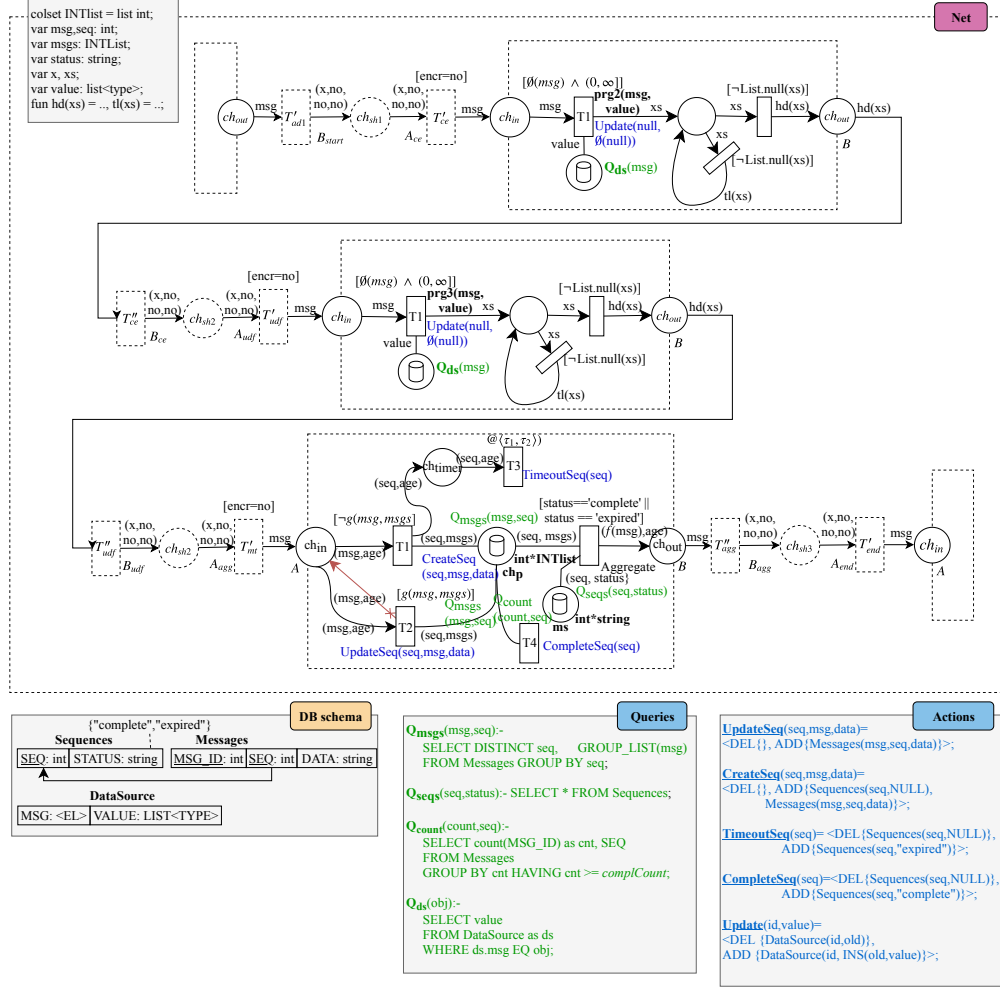


Figure 3.50: Predictive maintenance scenario as a timed db-nets with boundaries

The execution semantics of these compositions are given in the form of timed db-nets, which we extended to timed db-nets with boundaries, that allow checking the compositional correctness through a synchronization of their boundaries (similar to the contracts). With a two-step template-based translation mechanism for single patterns and then for their compositions, we defined a translation from IPCGs to timed db-nets with boundaries and showed the correctness of the translation (cf. RQ2-2(b): “How can the formalized compositions be translated to composed, formalized patterns?”). Finally, we studied the formalization and translation for real-world pattern compositions and their simulation (cf. RQ2-2(c) “How to realize formalized pattern compositions in real-world integration scenarios?”).

The separation of a higher-level graph structure for a more comprehensible representation of integration scenarios, a lower-level, executable timed db-net with boundary formalism, and a translation from the graphs to the PN balance the observed trade-off between comprehensible and simple modeling of integration scenarios on one side and expressiveness and comprehensive coverage of the execution semantics on the other side.

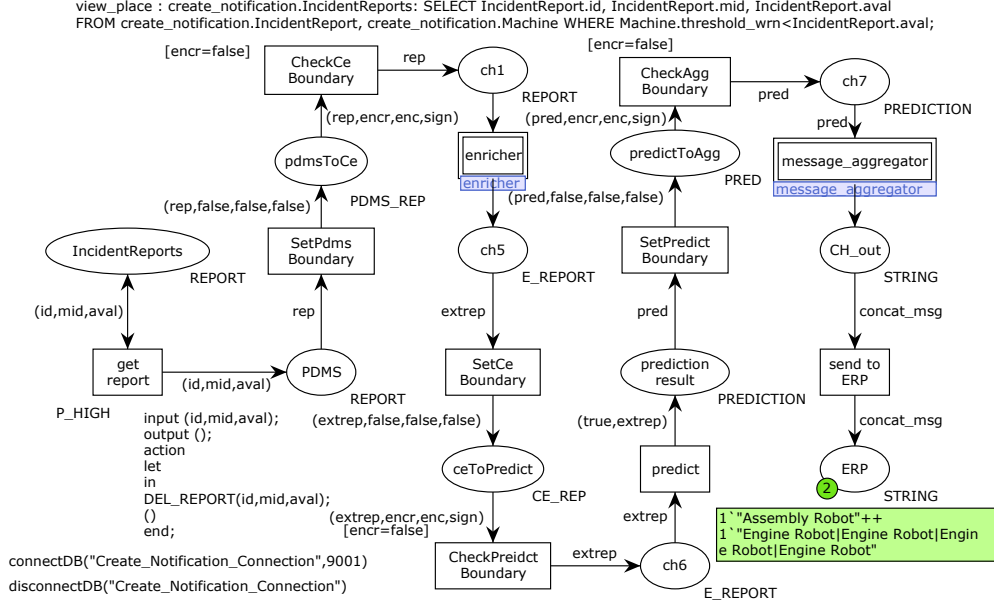


Figure 3.51: Predictive maintenance scenario simulation

The evaluation results into several interesting conclusions, i.e., the suitability of our approach for pattern compositions (cf. conclusions (1,3)), model complexity considerations (cf. conclusions (2,4)) and desirable extensions like automatic translation (cf. conclusion (5)). In particular, with conclusions (1,3), we showed that hypotheses H1 and H2 are fulfilled: “The integration pattern contract graphs allow for structurally correct compositions” and “The execution semantics of the translated timed db-net with boundaries is consistent”. That means, while IPCGs based on timed db-nets with boundaries denote the first comprehensive definition of application integration scenarios with built-in functional correctness and compositional correctness validation and verification, it does not give an appealing modeling language for (non-technical) users (cf. conclusions (2,4)). We envision a novel modeling language and tool support that facilitates a translation from that language to IPCGs (cf. conclusion (5)), which we consider as future work. Based on such a language infrastructure, more advanced compositional aspects like modeling guidelines on the different layers (i.e., language, intermediate IPCG, and simulation timed db-net with boundaries) could be studied. In summary, contract graphs rather denote an intermediate language that abstracts from the underlying, technical formalism, while preserving its semantics, and thus allows for manifold applications like the specification of composition-level optimizations with correctness guarantees.

3.3 Related Work

In this section we set the formalization of patterns and their compositions into the context of related work.

3.3.1 Pattern Formalization

We found in [Chapter 2](#) that the only existing formalization of EIPs is provided in [\[FG13\]](#) using CPNs. In particular, Fahland and Gierds [\[FG13\]](#) define messages as colored tokens and uses PN transition guards as conditions. However, this does not cover all requirements we singled out in [Table 3.1](#). The requirements REQ-x from [Section 3.1.1](#) are referenced during the subsequent discussion, where appropriate.

Petri Net Formalizations

The Reasoning about database transitions in timed db-nets is done similar to van Hee et al. [\[VHSV09\]](#), who define an alternative approach for representing and reasoning on database transactions using special token vectors with identifiers and inhibitor nets. While the latter could also be used similar to db-nets, we build our formalism on db-nets due to their more comprehensive focus on (relational) data, operations, and persistent storage. A slightly different and only loosely related approach to storage of data in PNs is introduced by Baldan et al. [\[BBC⁺18\]](#), who define persistent places, from which tokens can be used several times. However, neither (relational) transactional semantics, nor database operations are covered.

There exists a large body of work for representing temporal semantics in PNs, which cannot be covered completely in this work. In [Section 3.1.2](#) we already discussed the suitability of the implicit temporal support in PNs (i.e., adding places representing the current time) [\[vdA93\]](#), as well as the temporal semantics of adding timestamps to tokens [\[vdA93\]](#), timed places [\[Sif80\]](#), TAPNs [\[JJMS11\]](#) and transitions [\[Ram73, Zub87\]](#), out of which we decided on using TAPNs due to their semantic compatibility to db-nets, e.g., with respect to interleaving concurrency, and well-behaved formal analysis. Although this does not mean that none of the other temporal formalism could be applied for the EIP requirements, their application would require significant adaptations of db-net, e.g., for the simplest timed transition approach by [\[Ram73, Zub87\]](#) this would mean dealing with true concurrency and adapting the db-net LTS to transition firing delays. While ITCPN [\[vdA93\]](#) appear to be too restricted by time intervals with a single global time, the work on stochastic PNs (e.g., [\[Zen85\]](#)) might be additionally of interest for the representation of ordering in REQ-1(a) through priority functions, but it can hardly be used for practical reasoning, due to the inherent non-determinism of stochastic models. However, the study of these approaches helped during the specification of timed db-net.

Service-oriented Architecture and Interaction

In the related service-oriented architecture domain, service interactions and service interaction patterns were formalized. The work on service interactions largely targets formalizations on service orchestration and choreographies (i.e., similar to compositions of patterns), e.g., for web services [\[BCPV04, GGL05, BGG⁺05\]](#), which are all based on process algebras that account for our time requirements, however, lack database transaction semantics (cf. REQ-4). The same is true for π -calculus approaches (e.g., by Decker et al. [\[DPW06\]](#)) and similarly in the workflow domain by Puhlmann et al. [\[PW05\]](#).

Architecture Patterns

The approaches to formalize object-oriented, architectural patterns, or component-based systems (e.g., Alencar et al. [\[ACL96\]](#), Allen et al. [\[All97\]](#)) focus on pattern descriptions up to runtime instantiation, however, do not cover, e.g., time, compensation / transaction and execution semantics (cf. REQ-3, REQ-4).

3.3.2 Pattern Composition Formalization

We now situate our work within the context of other formalizations, beyond the already discussed BPMN and PN approaches (cf. *Open Nets* [BBGM15, BM18]).

Enterprise Application Integration

Similar to the BPMN and PN notations, several domain-specific languages (DSLs) have been developed that describe integration scenarios. Apart from the EIP icon notation [HW04], there is also the Java-based Apache Camel DSL [IA10], and the UML-based Guaraná DSL [FRQC11]. However, none of these languages aim to be optimization-friendly formal integration scenario representations. Conversely, we do not strive to build another integration DSL. Instead we claim that all of the integration scenarios expressed in such languages can be formally represented in our formalism, so that optimizations can be determined that can be used to rewrite the scenarios.

For the verification of service-oriented manufacturing systems, Mendes et al. [MLCR12] uses “high-level” Petri nets as a language instead of integration patterns, similar to the approach of Fahland and Gierds [FG13].

The work by Böhm and Kanne [BK09], specifies a declarative, data-aware language for describing distributed applications and their communication based on message queues. While this approach incorporates data aspects down to the (XML) database level, it does not specify a composition language for integration patterns and does not give verification results for the resulting integration programs.

Pattern Languages

The work on structural pattern composition by Hammouda and Koskimies [HK07] recognize the composition of patterns as relevant for the adoption of pattern languages and systems, and provide a tool for correctness checking [Ham04]. They propose the concept of role-based pattern composition based on the structural relationships and constraints of patterns in a UML-style extension, called *role diagram*, when instantiating patterns for building a system. While the constraints might be comparable to the pattern contracts, they are limited to structural properties and do not take data or other relevant integration pattern characteristics into account.

The approaches based on reasoning over first-order logic by Zhu and Bayley [ZB10, BZ10] and temporal logic by Taibi et al. [TN03, Tai06] give a formal definition of patterns and their compositions based on UML static class diagrams and dynamic sequence diagrams. While both approaches take dynamic, temporal aspects of the patterns into account, they do not consider other integration characteristics, which is probably due to their focus on the GoF patterns [GHJV95].

Porter, Coplien and Winn [PCW05] address the problem of composing patterns from different languages in a single system by sequences (i.e., similar to pipes and filter composition) and temporal ordering. Again important aspects for EAI like data and (transacted) storage or execution semantics are not considered.

Business Process Management and Service Composition

Early algorithmic work by Sadiq and Orlowska [SO00] applied reduction rules to workflow graphs for the visual identification of structural conflicts (e.g., deadlocks) in business processes. Compared to process control graphs, we use a similar base representation, which we extend by pattern characteristics and data contracts. Furthermore, we use graph rewriting for optimization purposes in the next chapter. In Cabanillas et al. [CRRCA11], the structural aspects are extended by a data-centered view of the process that allows to analyze the lifecycle of an object, and check

data compliance rules. Although this adds a view on the required data, it does not propose optimizations for the extended EIPs. The main focus is rather on the object lifecycle analysis of the process.

In the intersection of business process execution and service interaction, languages like the *Web Service Business Process Execution Language* (WS-BPEL) [Cor02] deal with the execution of business process and service engines. Although Scheibler et al. [SRL10] experimentally showed that pipes and filter processing can be simulated by business process execution engines like BPEL. However, to the best of our knowledge, none of the current integration system implementations builds on BPEL, and thus it does not seem to have any practical relevance, to our knowledge. Further, with its focus on the implementation perspective, formalisms for correctness arguments on BPEL flows were not built-in [TBBG07, CRP14] (e.g., in contrast to our approach), and were added later. For example, Ouyang et al. [OVVDA⁺07] specified some of the BPEL semantics at that time using Petri nets, and Fares et al. [FBF11] provided temporal constraints using LTL or WSDL-temporal by [BBM12]. Notably, these extensions mainly target the control flow and do not provide reasoning about data, CRUD operations or database transactions as also found in a survey by Ter Beek et al. [TBBG07].

In the workflow interaction domain, workflow nets and workflow modules are used, e.g., by van der Aalst and Weske [vdAW01], and Martens [Mar05], respectively. Furthermore, service interaction patterns are formalized using the composition capabilities of Petri nets provided by open nets (e.g., by van der Aalst et al. [vdAMSW09] or as open workflow nets by Massuthe et al. [MRS05]). Similar to our approach, van der Aalst et al. and Kindler et al. [vDA02, KMR00] use Open Nets [BBGM15, BM18] to represent and formally study the control flow in inter-operable workflows. Although their work does not consider the data flow and characteristics of integration patterns, it strengthens our design choice to use open nets for integration pattern compositions. Similarly, targeting the representation of service interactions as Petri nets, van der Aalst et al. [vDALM⁺10] use open nets to formally represent compositions. However, the main focus of that work lies on interactions of public and private services and whether the complete interaction is actually implemented by all parties (i.e., called weak termination of the overall process as actually implemented).

Miscellaneous

For the documentation of architectural decisions, when combining patterns, That et al. [TSOB13] propose a description language, which shares the idea of composing patterns in a pipes-and-filters style as also in [HW04].

Since patterns are employed during the design phase of a system, their composition also plays a role during the requirements engineering process. Therefore, Zlatev, Daneva, and Wieringa [ZDW05] propose an approach for the composition of patterns on an abstract level to support this process, but this is only remotely related to our approach.

3.4 Discussion

The comprehensive pattern language from Chapter 2 denotes a DSR artifact in the form of a valuable knowledge base that we leverage for the formalization of the integration patterns and their composition. We studied the structured pattern descriptions from different perspectives to extract the essential properties and characteristics to answer the stated research questions for the formalization of the execution semantics of single patterns (cf. RQ2-1: “*What is a suitable formalism for defining execution semantics of existing and new patterns?*”) as well as their composition (cf. RQ2-2: “*What is a sound and comprehensive formal representation of integration*

patterns that allows for formal validation or integration scenarios and reasoning?”). These questions are answered by the formalization results in this chapter that denote contributions in the form of DSR artifacts for single patterns:

- A list of integration pattern formalization requirements;
- a comprehensive formalism for defining single integration patterns (incl. formal analysis results; → RQ2-1);
- an instructive catalog of formalized integration patterns;
- an instantiation of the formalized patterns in the form of a prototype (for simulation);

and pattern compositions:

- An abstract formalization of pattern compositions with built-in structural composition correctness guarantees based on pattern contracts, denoting an intermediate representation of integration scenarios;
- a formal specification of these contracts as boundaries as extension of the pattern formalization for the formal assessment of semantic correctness;
- a method (algorithm) for translating structurally correct compositions to semantically correct pattern formalism (incl. correctness proofs; → RQ2-2);
- an instantiation of the formalized pattern compositions (translated to timed db-nets with boundaries) in the form of a prototype (for simulation).

Together they give answers to the stated research questions, and thus provide the first comprehensive formalism in EAI with built-in structural and semantic correctness guarantees as well as the means for an end-to-end formal analysis. This will allow for *responsible programming* of integration scenarios and applications, and denotes the foundation for various application areas such as the correctness-preserving, composition-level optimization. The formally defined pattern compositions denotes an intermediate model, e.g., for the development of an integration modeling language or scenario improvements in terms of optimizations.

Nonetheless, the different evaluations indicate limitations and open research challenges. These are subsequently summarized and discussed.

Limitations The limitations concern the assumption of a comprehensive foundation for the formalization in the form of the pattern language. We consider the pattern language comprehensiveness in terms of the current knowledge assembled in this work, which might be incomplete. For example, due to new trends new patterns or variations of existing patterns might be added. Would the proposed formalization still be valid for these patterns? We argue that the number of new concepts (e.g., data, time) will not grow in the same way as the number of new patterns. This is shown by the analysis in [Section 3.1.1](#), which illustrates the addition of twice as many new patterns, however, no new concepts and only three new requirement categories. Consequently, adding new patterns yet again will contribute to existing concepts and requirements, which are already covered by the current formalism. Only the addition of new categories would require a further extension of the formalism, similar to our journey from CPNs to timed db-nets. Since the composition specification is mostly orthogonal to the particular pattern concepts, only minor extensions might be necessary, if at all.

Notably, while the presented formalisms denote an important milestone toward formal foundations in EAI, neither the timed db-net formalism nor the more abstract pattern contract graphs can be considered a suitable modeling language for (non-technical) users. They rather denote a well-defined logical or intermediate representation for a new type of integration modeling language with correctness guarantees. Similarly, while the PN formalism closes the conceptual vs.

implementation gap by simulation, the translation of existing modeling languages and system implementations into our formalisms would make the results accessible in practice. Moreover, to leverage the formal analysis results (beyond validation through simulation), the development of a model checking tool for this new class of data-centric dynamic systems is a logical next step.

The instantiation of our formalism in the form of a prototype allows for simulation, however, does not denote a practically usable pattern solution that would perform well (e.g., in a pattern benchmark for message throughput and processing latency), and thus more efficient implementations have to be constructed, for which we make initial contributions in [Chapter 6](#).

Impact The impact of the formalizations on current research and practice might be enormous, but will take time. The first results of this work are already taken up by industry (e.g., in SAP Cloud Platform Integration [[SAP19a](#)]) to eventually allow for “responsible programming” [[Sta14](#)] when connecting applications and devices, for instance by grounding current modeling approaches on formal structures such as integration pattern contract graphs. However, the deeper rooted theme of *trust* in computer science artifacts used in the everyday lives of (non-technical) users in the process of the digital transformation of the society, only just reached academia and industry (e.g., trustworthy, reliable systems [[Eng18](#)] or trust in services and data [[IBM18](#)]).

Chapter 4

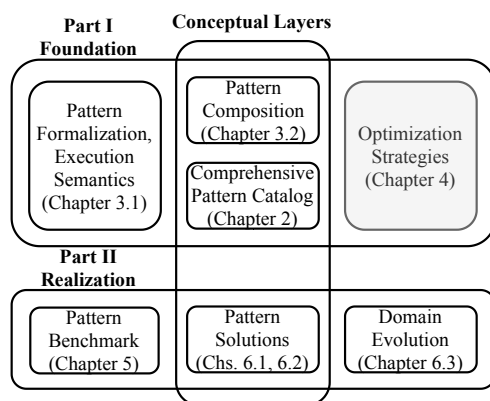
Optimization Strategies for Pattern Compositions

Contents

4.1	Optimization Strategies and Design Choices	139
4.2	Optimization Strategy Formalization	142
4.3	Evaluation	153
4.4	Related Work	160
4.5	Conclusions	162

[...] premature optimization is the root of all evil [...] in programming.
Donald Knuth, 1974 [Knu74]

In this chapter, we build on pattern compositions specified by Integration Pattern Contract Graphs (IPCGs), and we study improvements on the integration scenario-level in the form of



optimization strategies. We recall from [Chapter 3](#) that IPCG pattern compositions are suitable to formally represent integration scenarios by abstracting from the inner workings of the single patterns. However, while IPCGs allow for the analysis of structural and semantic composition correctness down to the validation of single pattern building blocks, they do not guarantee the efficiency of the composition in terms of modeling complexity at design time or efficient execution of integration scenarios at runtime. The complexity of scenarios was identified as a challenge in [Chapter 3](#), and we already discussed the relevance of efficient message processing as well as the current lack of solutions in [Chapters 1](#) and [2](#) (cf. *formalization* and *efficient*

pattern solution research gaps). Through the identified trends, the need to integrate a growing

number of distributed applications and increasingly complex, data-aware integration scenarios emerges, while scenario improvements are often vendor-specific, informal and ad-hoc. In this context the term *data-aware* stands for integration scenarios that have to process increasingly higher volumes of data (cf. research challenge C5 “Volocity”: volume and velocity in [Section 1.2.2](#)). Moreover, it remains unclear, whether an optimization preserves the functional correctness of the improved scenario, potentially resulting in more efficient, but flawed processing. In practice, optimizing such scenarios under the premise of retaining its functional correctness is required, but hard. However, not considering integration scenario improvements will eventually lead to even more complex, non-comprehensible integration scenarios as well as performance degradations during processing, and to overloaded and unavailable EAI systems.

A more efficient execution can either be achieved through (i) scenario-level improvements with a focus on processing latency or message throughput, which we approach in this chapter, or (ii) the development of more efficient pattern solutions (e.g., by leveraging new technologies to improve the message processing), for which we refer to Part II of this thesis (cf. [Chapters 5 and 6](#)). Although we identified several approaches that address improvements of data-aware processing in related domains (e.g., workflows and business processes [[VTM08](#), [KG14](#)], data integration [[BWHL08](#)], distributed applications [[Böh10](#), [BK11](#)]), none of them considers structural and semantic equivalence of integration pattern compositions before and after the improvement. We call this desirable property of an optimization *correctness-preserving* and argue that non-trustworthy optimizations (i.e., not having this property) will compromise the functional correctness of a pattern composition. In other words, if the optimized pattern composition has the same functional behavior as the original composition, the optimization preserves correctness. Therefore, the ability to reason about and verify the correctness of optimizations is required, which in turn must be grounded on a formalization of optimizations that fits to that of the composition (e.g., integration pattern contract graphs). Since a “premature optimization” might be problematic without considering the complete solution as stated by Knuth [[Knu74](#)], we target improvements of the whole integration scenario, instead of single patterns, and thus on scenarios represented by IPCGs. Moreover, since high-level improvements target the whole scenario and not the actual runtime, their results can be leveraged (e.g., by backporting) in current implementations. The subsequent example illustrates such an improvement that preserves the structural correctness of the original scenario and its semantic correctness.

Example 4.1. As a concrete motivation for a formal framework, we recall the material replication scenario from [Figure 3.23](#) (on [page 95](#)) with its corresponding IPCG in [Figure 3.34](#) (on [page 109](#)). Due to the data-independence of the Message Translator *MT* and Content Enricher *CE*, the processing of these patterns could be executed in parallel. To achieve this, a read-only unconditional fork with channel cardinality $1:n$ in the form of a Multicast *MC* pattern has been added. The inbound and outbound contracts of *MC* are adapted to fit into the composition. After the concurrent execution of *CE* and *MT*, a join router *JR* brings the messages back together again and feeds the result into an Aggregator *AGG* that restores the format that integration adapter *ADPT_r* expects. We see that the improved IPCG, shown in [Figure 4.1](#), is still structurally correct, due to the matching pattern contracts (cf. [Definition 3.34](#) (on [page 108](#))), so this would be a sound optimization. ■

Already in this simple scenario an obvious improvement can be applied. This insight would perhaps not be obvious without the data flow in the model, and leads to questions like “are there other optimizations that could also be applied?”, or “are the optimized compositions correct?”, and ultimately to our more general research question RQ2-3: “*What are relevant optimization strategies, and how can they be formally defined on pattern compositions?*”. So far, these questions could not be answered, since approaches for verification and static analysis of

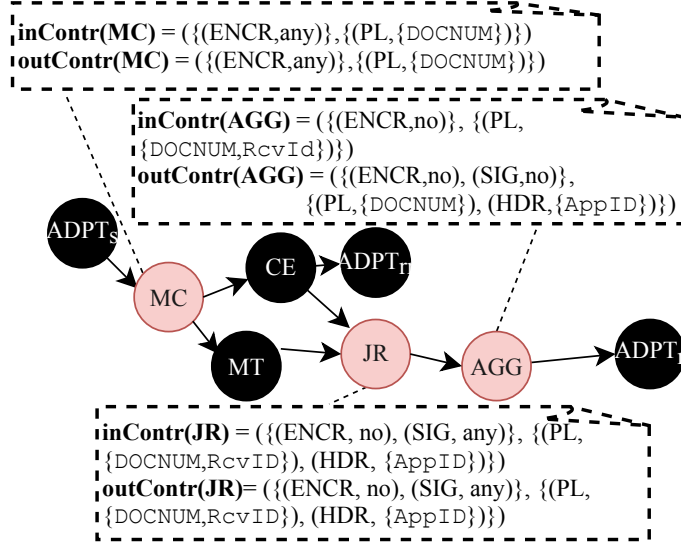


Figure 4.1: IPCG from the material replicate scenario in Figure 3.34 after a “sequence to parallel” optimization

“realistic” data-aware business and integration scenarios like those in Chapter 3 were missing, as recent surveys on event data [AAB⁺17, EGJW17], workflow management [KG14], and in particular application integration in Chapter 2 report. Hence, we aim to fill this gap driven by the following sub-questions of RQ2-3:

- (a) *What are relevant optimization techniques for EAI pattern compositions?*
- (b) *How can optimization strategies be formally defined?*
- (c) *How can the application of optimization strategies preserve the compositional correctness?*

To answer these questions we follow the responsible pattern composition process (from Figure 3.30 (on page 102)), which we extend to *correctness-preserving* optimizations in Figure 4.2 that allows for the formal analysis for pattern compositions during the optimization process. In the *analyze improvements* step, relevant optimization strategies for EAI pattern compositions are identified (cf. research question RQ2-3(a)). We focus on the common EAI optimization objectives, e.g., mentioned in [HW04]: message throughput (on experimental runtime benchmarks), pattern processing latency (on an abstract cost model), and also runtime independent model complexity [SGGM⁺10] from the process modeling domain. Then, in the *define rewrite rule* step, the distinct optimizations are formalized into what we call *rewrite rule* (cf. RQ2-3(b)). These rules are executed iteratively in a combined *match & apply rewrite rules* step, in which an optimization rule is applicable in a scenario, if *match rewrite rule* holds and the compositional correctness is preserved. The compositional correctness that is assumed for the original scenario is preserved through the already introduced responsible pattern composition process (cf. RQ2-3(c)). If the resulting composition is not correct, the improvement is not applied. We stress that we use the word “optimization” here in the sense of, e.g., an optimizing compiler: a process which iteratively improves compositions, but gives no guarantee of optimality.

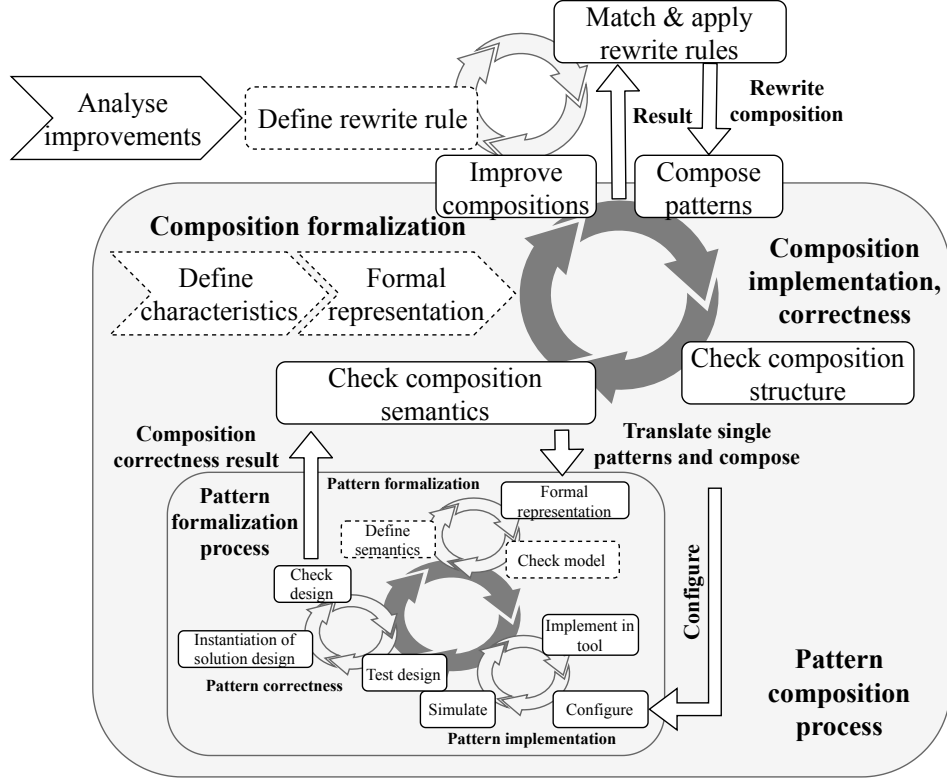


Figure 4.2: Responsible pattern composition optimization process (process-perspective)

Methodologically we first collect existing optimization techniques in related domains, classify single, recurring optimizations as optimization strategies and transfer them to EAI in [Section 4.1](#). For the formalization of optimization strategies, we recall that pattern compositions are formally specified in Integration Pattern Contract Graphs (IPCGs). Based on the analysis and the composition formalism, we select a suitable formal representation for the specification of *rewrite rules*, and formally specify the optimizations per strategy in [Section 4.2](#). Although the IPCGs can be checked for structural and semantic correctness, the *correctness-preserving* property has to be considered during the definition and shown for each distinct optimization. Together, these formalisms allow for the realization of a correctness-preserving application of optimization strategies to pattern compositions, which we evaluate quantitatively and in case studies according to our objectives in [Section 4.3](#). In [Section 4.4](#) we discuss related work, before we conclude in [Section 4.5](#).

The resulting verification and analysis framework for applying and reasoning about optimizations of data-aware integration scenarios does not only allow to show that all optimizations are improving certain aspects of an integration scenario, but that they preserve their correctness down to the execution semantics. A *responsible programming* is essential for the subsequent design of new pattern solutions and further studies on the applicability of the optimization strategies.

Parts of this chapter have previously been published in the proceedings of DEBS 2018 [[RMFR18](#)] (formalizing optimizations) together with parts of the pattern composition formalism in [Section 3.2](#) and a technical report [[RFRM19](#)] (collection of optimization strategies).

Table 4.1: Optimizations in related domains — horizontal search

Keyword	hits	selected	Selection criteria	Selected Papers
Business Process Optimization	159	3	data-aware processes	survey [VTM08], optimization patterns [NRM11, NS11]
Workflow Optimization	396	6	data-aware processes	instance scheduling [ABMR10, BM11, TULA13], scheduling and partitioning for interaction [ALR ⁺ 14], scheduling and placement [BCN ⁺ 12], operator merge [HAMR13]
Data Integration Optimization	61	2	data-aware processes optimization, (no schema-matching)	instance scheduling, parallelization [ZHZW12], ordering, materialization, arguments, algebraic [Get11]
Added	n/a	8	expert knowledge	business process [VSS ⁺ 07], workflow survey [KG14, KGS17], data integration [BWHL08], distributed applications [Böh10, BK11], EAI [RDMRM17, Rit17a]
Removed	-	1		classification only [VTM08]
Overall	616	18		

4.1 Optimization Strategies and Design Choices

We survey recent attempts to optimize integration pattern compositions, in order to motivate the need to formalize their semantics. As a result, we derive three so far unexplored prerequisites **R1–R3** for optimizing compositions of integration patterns, for which we discuss design choices.

4.1.1 Identifying Optimization Strategies

Since a formalization of the EAI foundations in the form of integration patterns for static optimization of data-aware integration scenarios is missing (cf. Chapter 2), we conducted a horizontal literature search [Kit04] to identify optimization techniques in related domains. For EAI, the domains of business processes, workflow management and data integration are of particular interest. The results of our analysis are summarized in Table 4.1. Out of the resulting 616 hits, we selected 18 papers according to the search criteria “data-aware processes”, and excluded work on unrelated aspects. Table 4.2 lists the optimization techniques, already mapped to EAI, and skipping those techniques that do not provide solutions for our optimization objectives or within an integration process. This resulted in the *seven* papers cited in the table. The mapping of techniques from related domains to application integration was done by for instance taking the idea of projection push-downs [BHP⁺11, Get11, HAMR13, NRM11, VSS⁺07] and deriving the early-filter or early-mapping technique in EAI. We categorized the techniques according to their impact (e.g., structural or process, data-flow) in context of the objectives for which they provide solutions.

In the following subsections, we now briefly discuss the optimization strategies listed in Table 4.2, in order to derive prerequisites needed for optimizing compositions of integration patterns. To relate to their practical relevance and coverage so far (in the form of evaluations on real-world integration scenarios), we also discuss existing data-aware message processing solutions for each group of strategies.

Table 4.2: Optimization strategies in context of the objectives

Strategy	Optimization	Throughput	Latency	Complexity	Practical Studies
OS-1:	Redundant Sub-process Removal [BHP ⁺ 11]	+/-	+	+	-
Simplification	Combine Sibling Patterns [BHP ⁺ 11, HAMR13]	+/-	+	+	(Section 6.2)
	Unnecessary conditional fork [BHP ⁺ 11, VSS ⁺ 07]	(+)	+	+	-
OS-2:	Early-Filter [BHP ⁺ 11, Get11, HAMR13, NRM11, VSS ⁺ 07]	+	+/-	+/-	Section 6.2
Data Reduction	Early-Mapping [BHP ⁺ 11, Get11, HAMR13]	+	+/-	+/-	Sections 6.2 and 6.3
	Early-Aggregation [BHP ⁺ 11, Get11, HAMR13]	+	+/-	+/-	Section 6.3
	Early Claim Check [BHP ⁺ 11, Get11]	+	+/-	-	-
	Early-Split [RDMRM17]	+	+/-	-	Sections 6.2 and 6.3
OS-3:	Sequence to parallel [BHP ⁺ 11, NRM11, VSS ⁺ 07, ZHZW12]	+	+/-	-	[Rit15b], Sections 6.1 and 6.2
Paral- lelization	Merge parallel sub-processes [BHP ⁺ 11, NRM11, VSS ⁺ 07, ZHZW12]	+/-	+	+	Section 6.2

+ = improvement, - = deterioration, +/- = no effect, (+) = slight improvement, (-) = slight deterioration.

4.1.2 Process Simplification

We grouped together all techniques whose main goal is reducing model complexity (i.e., the number of patterns) under the heading of process simplification. The cost reduction of these techniques can be measured by pattern processing time (latency, i.e., time required per operation) and model complexity metrics [SGGM⁺10]. Process simplification can be achieved by removing redundant patterns like *Redundant Subprocess Removal* (e.g., remove one of two identical sub-flows), *Combine Sibling Patterns* (e.g., remove one of two identical patterns), or *Unnecessary Conditional Fork* (e.g., remove redundant branching). As far as we know, the only practical study of combining sibling patterns can be found in Ritter et al. (cf. Section 6.2), showing moderate throughput improvements. The simplifications require a formalization of patterns as a **control graph structure (requirement R1)**, which help to identify and deal with the structural change representation. Previous work targeting process simplification include Böhm et al. [BHP⁺11] and Habib, Anjum and Rana [HAMR13], who use evolutionary search approaches on workflow graphs, and Vrhovnik et al. [VSS⁺07], who use a rule formalization on query graphs.

4.1.3 Data Reduction

The reduction of data can be facilitated by pattern push-down optimizations of message-element-cardinality-reducing patterns, which we call *Early-Filter* (for data; e.g., remove elements from the message content), *Early-Mapping* (e.g., apply message transformations), as well as message-reducing optimization patterns like *Early-Filter* (for messages; e.g., remove messages), *Early-Aggregation* (e.g., combine multiple messages to fewer ones), *Early-Claim Check* (e.g., store

content and claim later without passing it through the pipeline), and *Early-Split* (e.g., cut one large message into several smaller ones). Measuring data reduction requires a cost model based on the characteristics of the patterns, as well as the data and element cardinalities. For example, the practical realizations for multimedia (cf. [Section 6.3](#)) and hardware streaming (cf. [Section 6.2](#)) show improvements especially for early-filter, split and aggregation, as well as moderate improvements for early-mapping. This requires a formalization that is able to represent **data or element flow (requirement R2)**. Data reduction optimizations target message throughput improvements (i.e., processed messages per time unit), however, some have a negative impact on the model complexity. Previous work on data reduction include Getta [[Get11](#)], who targets optimization techniques on relational algebra expressions, and Niedermann, Radeschütz and Mitschang [[NRM11](#)], who define optimizations algorithmically for a graph-based model.

4.1.4 Parallelization

Parallelization of processes can be facilitated through transformations such as *Sequence to Parallel* (e.g., duplicate pattern or sequence of pattern processing), or, if not beneficial, reverted, e.g., by *Merge Parallel*. For example, good practical results have been shown for vectorization (cf. [Section 6.1](#)) and hardware parallelization (cf. [Section 6.2](#)). Therefore, again, a **control graph structure (R1)** is required. Although the main focus of parallelization is message throughput, heterogeneous variants also improve latency. In both cases, parallelization requires adding patterns, which negatively impacts the model complexity. The opposite optimization of merging parallel processes mainly improves the model complexity and latency. Previous work on pattern parallelization include Zhang et al. [[ZHZW12](#)], who define a service composition model, to which algorithmically defined optimizations are applied.

4.1.5 Discussion and Design Choices

Due to our objectives and our focus on optimizations within an integration scenario, the collection of optimizations in [Table 4.2](#) is not complete. For instance, we have not treated pattern placement optimizations (pushing patterns to message endpoints, i.e., sender and receiver applications), or optimizations that reduce interaction (helping to stabilize the scenario through making it less dependent). Besides control flow (as used in most of the related domains), a suitable formalization must be able to represent the **control graph structure (R1)** (including reachability and connectedness properties) and the **data element flow (R2)** between patterns (not within a pattern). Furthermore, the formalization must allow verification of **correctness (requirement R3)** on a pattern-compositional level (i.e., each optimization produces a correct pattern composition), taking the inter-pattern data exchange semantics into account.

We recall that formalization requirements **(R1)** and **(R2)** as well as compositional correctness are inherent to IPCGs defined in [Section 3.2](#). With IPCGs as the underlying formalism that can be translated to timed db-nets with boundaries for checking semantic correctness, optimizations are naturally expressed as rewrite rules on the IPCG. Hence, similar to the work by Balogh and Varró [[BV06](#)], who use graph transformation techniques (e.g., [[EEPT06](#), [ERK99](#)]) for the process of composing design patterns, we formally define optimization strategies in an algebraic graph rewriting framework. In other words, each optimization will be defined as a graph transformation or rewrite rule. Note that a formal framework provides base guarantees that are desirable in our case (e.g., no dangling nodes or edges after rewriting [[EPS73](#), [CMR⁺97](#)]). The resulting algebraically grounded rewritings of the IPCGs can be formally analyzed with respect to their compositional and functional correctness (cf. [Chapter 3](#)), which makes them correctness-preserving optimizations in the form of IPCG rewritings, and thus fulfilling requirement **(R3)**.

4.2 Optimization Strategy Formalization

We formally define the optimizations from the different strategies identified in Table 4.2 using a rule-based graph rewriting system. This gives a formal framework, in which different optimizations can be compared. We first introduce the graph rewriting framework and then subsequently define the optimizations.

Graph rewriting or transformation provides a visual framework for transforming graphs in a rule-based fashion, which we subsequently describe based on the monograph by Taentzer et al. [EPT06] and the handbook from Ehrig et al. [ERK99]). We recall that IPCGs are defined as labeled, control flow graphs (cf. Definition 3.28 (on page 106)). To be able to relate graphs in a formal way, we use structure-preserving *graph morphisms* from nodes and edges of one graph to another one, given in Definition 4.2.

Definition 4.2 (Graph morphism). Given graphs G, H over a label alphabet C , a *graph morphism* $f : G \rightarrow H$ is a pair of mappings ($f_V : V_G \rightarrow V_H, f_E : E_G \rightarrow E_H$) that preserve the sources, targets, and labels, i.e., $s_H \circ f_E = f_V \circ s_G, t_H \circ f_E = f_V \circ t_G, m_H \circ f_E = m_G$, and $l_H(f_V(v)) = l_G(v)$ for all nodes v for which $l_G(v)$ is defined (with function composition \circ).

Given graph morphisms $f : F \rightarrow G$ and $g : G \rightarrow H$, the composition $g \circ f : F \rightarrow H$ is defined as $g \circ f = \langle g_V \circ f_V, g_E \circ f_E \rangle$. ■

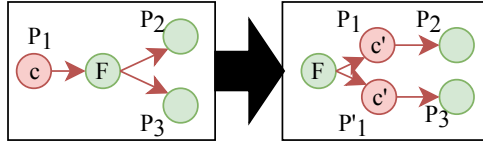
We denote injective morphisms $f : G \rightarrow H$ by a hooked arrow $f : G \hookrightarrow H$. A graph morphism f is *injective* (*surjective*), if both f_V and f_E are injective (surjective).

A graph rewriting rule is given by two embeddings of graphs $L \hookleftarrow K \hookrightarrow R$, where L represents the left hand side of the rewrite rule, R the right hand side, and K their intersection (the parts of the graph that should be preserved by the rule) as defined in Definition 4.3.

Definition 4.3 (Graph rule). A rule $r : \langle L \hookleftarrow K \hookrightarrow R \rangle$ over a label alphabet C comprises partially labeled graphs $L, K, R \in G(C)$, and inclusions $K \hookrightarrow L$ and $K \hookrightarrow R$. Then L denotes the *left hand side*, R the *right hand side*, and K the *interface* of the rule r . ■

A rewrite rule can be applied to a graph G after a match of L in G has been given as an embedding $L \hookrightarrow G$; applying the rule replaces the match of L in G by R . The application of a rule is potentially non-deterministic: several distinct matches can be possible [EPT06]. Visually, we represent a rewrite rule by a left hand side and a right hand side graph colored green and red: green parts are shared and represent K , while the red parts are to be deleted in the left hand side, and inserted in the right hand side respectively.

Example 4.4. The following rewrite rule moves the node P_1 past a fork by making a copy in each branch of a labeled graph based on the graph morphism, changing its label from c to c' in the process:



Formally, the rewritten graph is constructed using a double-pushout (DPO) [EPS73, CMR⁺97] from category theory. We use DPO rewriting since rule applications are side-effect free (e.g., no “dangling” edges) and local (i.e., all graph changes are described by the rules). Given rule r and a partially labeled graph G , an injective graph morphism $g : L \hookrightarrow G$ is a match for r , if it satisfies

the *dangling condition* in Definition 4.5, which intuitively means that if r deletes a node in the match, then all edges incident to it are also in the match and would be deleted by r . We write $g(G)$ for the component-wise image $(g_V(V), g_E(E))$ of a graph morphism $g = (g_V, g_E)$ to a graph $G = (V, E)$.


Definition 4.5 (Dangling condition; match). Let $r : \langle L \leftarrow K \hookrightarrow R \rangle$ be a rule, G a labeled graph, and $g : L \hookrightarrow G$ an injective graph morphism. The embedding $g : L \hookrightarrow G$ is a *match* for r , if it satisfies the *dangling condition*: no edge in $G - g(L)$ is incident to any node in $g(L - K)$. ■

The application of a rule is given in Definition 4.6, which describes the deletion of everything in the match that does not preserve the interface, and eventually adding everything in R that is not in the interface.

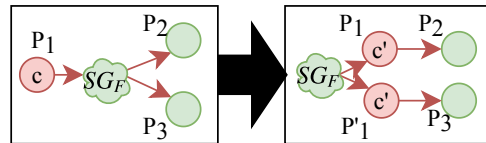
Definition 4.6 (Rule application). With a rule $r : \langle L \leftarrow K \hookrightarrow R \rangle$ and a match $g : L \hookrightarrow G$ is constructed from G as follows:

- Remove all nodes and edges in $g(L) - g(K)$, obtaining a graph D ;
- Add disjointly to D all nodes and edges from $R - K$ retaining their labels, obtaining a graph H . For $e \in E_R - E_K$, $s_H(e) = s_R(e)$, if $s_R(e) \in V_R - V_K$, otherwise $s_H(e) = g_V(s_R(e))$;
- Analogously target functions are defined.

We write $G \Rightarrow_{r,g} M$, if applying rule r to match s in g results in M . ■

The rather operationally defined rule application with steps as well as individual nodes and edges mapped by a match should be sufficient for the understanding of our approach. Alternatively, the underlying abstract, algebraic characterisation of the DPO approach, which treats the graphs as algebras and models the rule application as two pushouts can be further studied in [EPS73, CMR⁺97]. We additionally use Habel and Plump’s relabeling DPO extension [HP02] to facilitate the relabeling of nodes in partially labeled graphs (cf. node relabeling in Example 4.4). This allows for changing the node labels during the transformation of the graph as part of the DPO. Essentially, the node labels in the graph on the right hand side are preserved, else taken from the left hand side of the rule (e.g., shown in Examples 4.4 and 4.7). For more details we refer to [HP02]. In our motivating example in Figure 4.1, we showed contracts and characteristics in dashed boxes, but in the rules that follow, we will represent them as (schematic) labels inside the nodes for space reasons. We also consider rewrite rules parameterized by graphs, where we draw the parameter graph as a labeled cloud  (see e.g., Example 4.7 for an applied example). A cloud represents any graph, sometimes with some side-conditions that are stated together with the rule. When looking for a match in a given graph G , it is of course sufficient to instantiate clouds with subgraphs of G — this way, we can reduce the infinite number of rules that a parameterized rewrite rule represents to a finite number. Parameterized rewrite rules can formally be represented using substitution of hypergraphs [PH94] or by !-boxes in open graphs [DD09, KMS12].

Example 4.7. With a labeled “cloud”, the rewrite rule in Example 4.4 can be written as subsequently depicted. Still the rule moves the node P_1 past a fork by making a copy in each branch of a labeled graph based on the graph morphism, changing its label from c to c' in the process. However, this time the unconditional fork pattern node F is replaced by an arbitrary subgraph SG_F , which is moved instead for F .



In the same way, several nodes can be grouped to subgraph nodes. Note that SG_F can be instantiated to F to recover the previous example. ■

Since graph rewriting is naturally applicable to IPCGs and important base properties of such rewritings (e.g., no dangling nodes or edges), we subsequently formally define all optimizations by rewrite rules. Methodologically, the rules are specified by *pre-conditions*, *change primitives*, *post-conditions* and an optimization effect, where the pre- and post-conditions are implicit in the applicability and result of the rewriting rule. Note that the optimization effects denote an intermediate discussion of when the optimization is beneficial. Moreover, since pre- and post-conditions are implicit in the rule, we only discuss the more complicated change primitives and the resulting effects in more detail.

Remark 4.8. In [Example 4.4](#) the pre-conditions are denoted by the left hand side and the post-condition by the right hand side of the rule. In other words, if and only if the left hand side matches, the rule is executed and produces the right hand side, which denotes the desired outcome of the rewriting. The change primitives are represented by the whole rule. ■

4.2.1 OS-1: Process Simplification

We first consider the process simplification strategies from [Section 4.1 OS-1](#) that mainly strive to reduce the model complexity and latency.

Redundant Sub-Process

This optimization removes redundant copies of the same sub-process within an IPCG.

Change primitives: The rewriting is depicted by the rule in [Figure 4.3](#)¹, where SG_1 and SG_2 are isomorphic pattern graphs with in-degree n and out-degree m , which is depicted by $1, \dots, n$ preceding patterns i of SG_1 and the same number of preceding patterns j of SG_2 . Similarly, this is

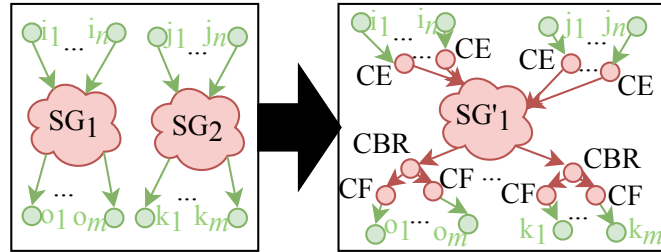


Figure 4.3: Redundant sub-process rule

illustrated for the $1, \dots, m$ succeeding patterns o of SG_1 and k of SG_2 . The Content Enricher (CE) node is a message processor pattern from [Figure 3.39](#) (on [page 117](#)) with a pattern characteristic $(PRG, (\text{addCtxt}, [0, \infty)))$ for an enrichment program addCtxt (not shown) which is used to add content to the message, helping to answer the question “does the message come from any pattern i_1, \dots, i_n from the left or j_1, \dots, j_n from the right subgraph?”. Similarly, the Content Filter (CF) is a message processor, with a pattern characteristic $(PRG, (\text{removeCtxt}, [0, \infty)))$ for an enrichment

¹We briefly repeat the notation for a better understanding: nodes denote patterns $(i_1, \dots, i_n, j_1, \dots, j_n, o_1, \dots, o_m, k_1, \dots, k_m)$ and node clouds stand for subgraphs of several nodes (SG_1, SG_2) . Edges represent message channels. The coloring depicts the interface or intersection of the left and right hand sides of the rule: red elements change, green elements remain unchanged.

program `removeCtx` which is used to remove the added content from the message again. Note that the additional Content Enricher and Content Filter patterns are required to construct the optimization rule. Moreover, the Content-based Router (CBR) node is a conditional fork pattern from Figure 3.38 with a pattern characteristic $(CND, \{\text{fromLeft}\})$ for a condition `fromLeft` which is used to route messages depending on their added context. In the right hand side of the rule, the CE nodes add the context of the predecessor node to the message in the form of a content enricher pattern, and the CBR nodes are content-based routers that route the message to the correct recipient based on the context introduced by CE . The graph SG'_1 is the same as SG_1 , but with the context introduced by CE copied along everywhere, and thus a copy of SG_1 with the context as additional element in the message. This context is stripped off the message by a content filter CF .

Effect: The optimization is beneficial for model complexity when the isomorphic subgraphs contain more than $n + m$ nodes, where n is the in-degree and m the out-degree of the isomorphic subgraphs. The latency reduction is by that of the subgraphs minus the latency introduced by the n extra nodes CE , m extra nodes CBR and k extra nodes CF .

Combine Sibling Patterns

Sibling patterns have the same parent node in the pattern graph (e.g., they follow a non-conditional forking pattern) with channel cardinality of 1:1. Combining them means that only one copy of a message is traveling through the graph instead of two — for this transformation to be correct in general, the siblings also need to be side-effect free, i.e., no external calls.

Change primitives: The rule is given in Figure 4.4, where SG_1 and SG_2 are isomorphic pattern subgraphs, and F is a fork. After the execution of the rewrite rule, one of the redundant subgraphs — in this case SG_1 — is moved between pattern P_1 and the fork F , while the latter is now connected directly to the subsequent patterns P_2 and P_3 .

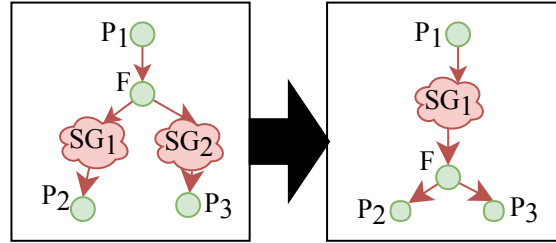


Figure 4.4: Combine sibling patterns rule

Effect: The model complexity and latency are reduced by the model complexity and latency of SG_2 , which also saves processing resources.

4.2.2 OS-2: Data Reduction

Now, we consider data reduction optimization strategies, which mainly target improvements of the message throughput (incl. reducing element cardinalities). These optimizations require that pattern input and output contracts are regularly updated with snapshots of element data sets EL_{in} and EL_{out} (cf. Definition 3.32 (on page 107)) from live systems (cf. abstract cost model Section 4.3.1 (on page 153)), e.g., from experimental measurements through pattern benchmarks (cf. Chapter 5).

Early-Filter

A filter pattern can be moved to or inserted prior to some of its successors to reduce the data to be processed. The following types of filters have to be differentiated:

- A *message filter* removes messages with invalid or incomplete content. It can be used to prevent exceptional situations, and thus improves stability.
- A *content filter* removes elements from messages, thus reduces the amount of data passed to subsequent patterns.

Both patterns are message processors in the sense of Figure 3.39 (on page 117). The Content Filter (CF) is a message processor, with a pattern characteristic $(PRG, (prg, [0, \infty)))$ for some filter function prg that removes elements from a message, while the Message Filter (MF) is a pattern with characteristic $(CND, \{c1\})$ for some filter condition $c1$ (not shown).

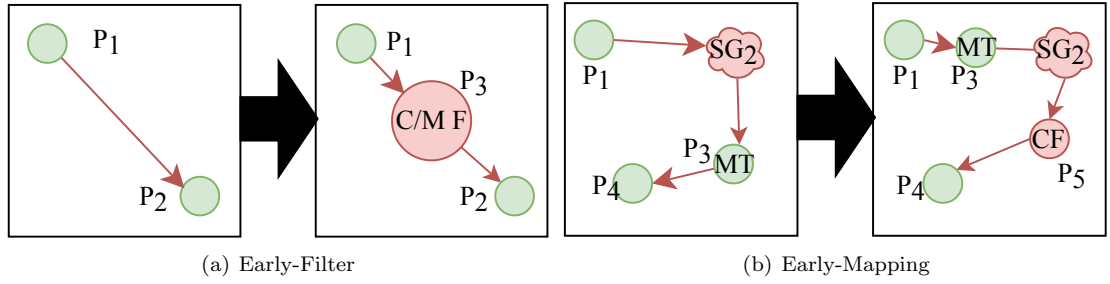


Figure 4.5: Rules for early-filter and early-mapping

Change primitives: The rule is provided in Figure 4.5(a), where P_3 (C/MF) is either a content or message filter matching the output contracts of P_1 and the input contract of P_2 , removing the data not used by P_2 . More precisely, if there are message elements in the output contract of pattern P_1 that are not needed in the input contract of pattern P_2 (or any other subsequent pattern), then a filter pattern P_3 can be inserted that removes these elements, and thus avoid the transport of unnecessary data.

Effect: The message throughput increases by the ratio of the number of reduced data elements that are processed per second, unless limited by the throughput of the additional pattern.

Early-Mapping

A mapping that reduces the number of elements in a message can increase the message throughput.

Change primitives: The rule is given in Figure 4.5(b), where P_3 is an element reducing message mapping compatible with both SG_2 , P_4 , and P_1 , SG_2 , and where P_4 does not modify the elements mentioned in the output contract of P_3 . Furthermore P_5 is a content filter, which ensures that the input contract of P_4 is satisfied (i.e., bridging from SG_2 to P_4 , if necessary). The Message Translator (MT) node is a message processor pattern from Figure 3.39 (on page 117) with a pattern characteristic $(PRG, (prg, [0, \infty)))$ for some mapping program prg which is used to transform the message. The subgraph SG_2 is changed after the rule execution, since it is moved behind P_3 and its data exchange is adjusted according to the filtered elements, formally represented by the output contract of P_3 (not shown).

Effect: The message throughput for the subgraph subsequent to the mapping increases by the ratio of the number of unnecessary data elements processed.

Early-Aggregation

A micro-batch processing region is a subgraph which contains patterns that are able to process multiple messages combined to a multi-message (cf. Chapter 5) or one message with multiple segments with an increased message throughput. The optimal number of aggregated messages is determined by the highest batch-size for the throughput ratio of the pattern with the lowest throughput, if latency is not considered.

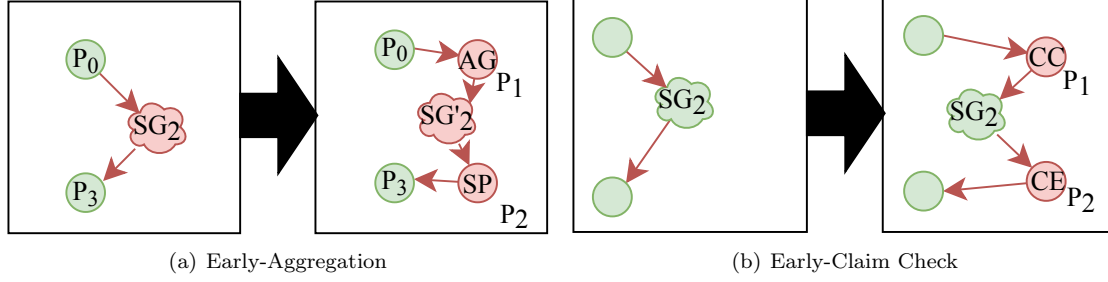


Figure 4.6: Rules for early-aggregation and early-claim check

Change primitives: The rule is given in Figure 4.6(a), where SG_2 is a micro-batch processing region, P_1 an Aggregator, P_2 a Splitter which separates the batch entries to distinct messages to reverse the aggregation, and finally SG_2 is rewritten to SG'_2 , which processes the aggregated messages. The Aggregator (AG) node is a merge pattern from Figure 3.40 (on page 118) with a pattern characteristic $\{(CND, \{cnd_{cr}, cnd_{cc}\}), (PRG, prg_{agg}, (v_1, v_2))\}$ for some correlation condition cnd_{cr} , completion condition cnd_{cc} , aggregation function prg_{agg} , and timeout interval (v_1, v_2) which is used to aggregate messages. The Splitter (SP) node is a message processor from Figure 3.39 (on page 117) with a pattern characteristic $(PRG, (prg, [0, \infty)))$ for some split function prg which is used to split the message into several ones.

Effect: The message throughput is the minimal pattern throughput of all patterns in the micro-batch processing region. If the region is followed by patterns with less throughput, only the overall latency might be improved.

Early-Claim Check

If a subgraph does not contain a pattern with message access, the message payload can be stored intermediately persistently or transiently (depending on the quality of service level) and not moved through the subgraph. For instance, this applies to subgraphs consisting of data independent control-flow logic only, or those that operate entirely on the message header (e.g., header routing).

Change primitives: The rule is given in Figure 4.6(b), where SG_2 is a message access-free subgraph, P_1 a claim check that stores the message payload and adds a claim to the message properties (and possibly routing information to the message header), and P_2 a content enricher that adds the original payload to the message. The Claim Check (CC) node is a message processor from Figure 3.39 (on page 117) with a pattern characteristic $(PRG, (-, [0, \infty)))$, which stores the message for later retrieval.

Effect: The main memory consumption and CPU load decreases, which could increase the message throughput of SG_2 , if the claim check and content enricher pattern throughput is greater than or equal to the improved throughput of each of the patterns in the subgraph.

Early-Split

Messages with many segments can be reduced to several messages with fewer segments, and thereby reducing the processing required per message. A segment is an iterable part of a message like a list of elements. When such a message grows bigger, the message throughput of a set of adjacent patterns might decrease, compared the expected performance for a single segment. We call this a *segment bottleneck sub-sequence*. Algorithmically, these bottlenecks could be found, e.g., using max flow-min cut techniques based on workload statistics of a scenario.

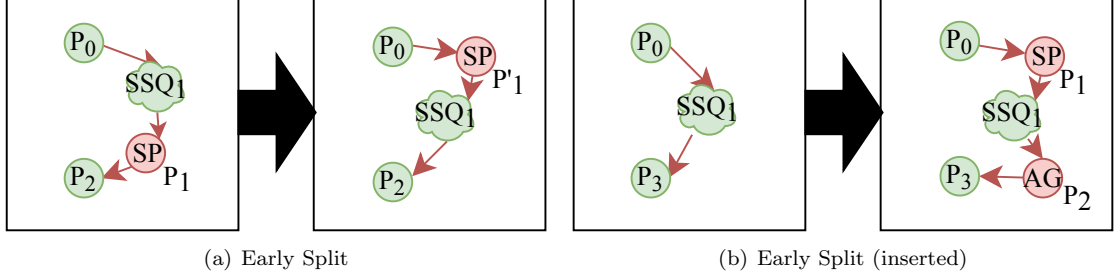


Figure 4.7: Rules for early-split

Change primitives: The splitter (SP) node is a message processor from Figure 3.39 (on page 117) with a pattern characteristic $(PRG, (prg, [0, \infty)))$, for some split program prg . The rule is given in Figure 4.7, where SSQ_1 is a segment bottleneck sub-sequence. If SSQ_1 already has an adjacent splitter, Figure 4.7(a) applies, otherwise Figure 4.7(b). In the latter case, SP is a splitter and P_2 is an aggregator that re-builds the required segments for the successor in SG_2 . For an already existing splitter P_1 in Figure 4.7(a), the split condition has to be adjusted to the elements required by the input contract of the subsequent pattern in SSQ_1 . In both cases we assume that the patterns in SSQ_1 deal with single- and multi-segment messages; otherwise all patterns have to be adjusted as well.

Effect: The message throughput increases by the ratio of increased throughput on less message segments, if the throughput of the moved or added splitter (and aggregator) \geq message throughput of each of the patterns in the segment bottleneck sub-sequence after the segment reduction.

Note that the Early-Split optimization can be similarly used to reduce the allocated main memory per message during processing in SSQ_1 , which might help to avoid memory allocation errors for larger messages.

4.2.3 OS-3: Parallelization

Parallelization optimization strategies increase message throughput. Again, these optimizations require experimentally measured message throughput statistics, e.g., from benchmarks (cf. Chapter 5).

Sequence to Parallel

A bottleneck sub-sequence with channel cardinality 1:1 can also be handled by distributing its input and replicating its logic. The parallelization factor is the average message throughput of the predecessor and successor of the sequence divided by two, which denotes the improvement potential of the bottleneck sub-sequence. The goal is to not overachieve the mean of predecessor and successor throughput with the improvement to avoid iterative re-optimization. Hence the

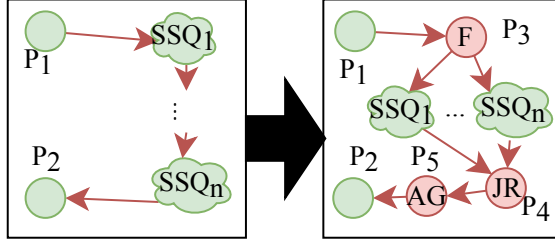


Figure 4.9: Heterogeneous sequence to parallel

Effect: Synchronization latency can be improved, but the model complexity increases by 3. The latency improves from the sum of the sequential pattern latencies to the maximal latency of all sub-sequence parts plus the fork, join, and Aggregator latencies.

In summary, since we describe optimization strategies as graph rewrite rules, we can be flexible with when and in what order we apply the strategies. We apply the rules repeatedly until a fixed point is reached, i.e., when no further changes are possible, making the process idempotent. Nevertheless, the rules can be stratified according to their effects for the application (i.e., “simplification before parallelization” and “structure before data”). Each rule application preserves IPCG correctness in the sense of [Definition 3.35](#) (on [page 108](#)), because input contracts do not get more specific, and output contracts remain the same. That means that only rewrite rules are allowed which lead to correct compositions on the right hand side, thus preserving the structural correctness of the composition. The semantic correctness requires more elaborate considerations, which we subsequently discuss more formally for each of the formalized optimizations.

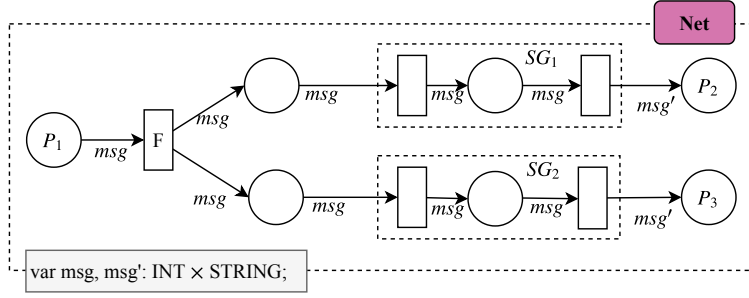
4.2.4 Optimization Correctness

We now show that the optimizations do not change the input-output behaviour of the pattern graphs in the timed db-nets semantics, i.e., if we have a rewrite rule from graph G to graph G' as $G \Rightarrow_{r,g} G'$ (cf. [Definition 4.6](#) (on [page 143](#))), then the constructed timed db-net with boundaries $\llbracket G \rrbracket$ has the same observable behaviour as that of $\llbracket G' \rrbracket$ (cf. [Section 3.2.4](#)). More formally, having the “same input-output behavior” requires that the transition systems of the original and the rewritten graphs are bisimilar in a certain sense (cf. timed db-net transition system based on db-net in [Sections 3.1.2](#) and [3.2.3](#)), as more formally defined in [Definition 4.9](#). For being bisimilar, it is required that the left hand side of a rewrite can simulate the right hand side, and vice versa.

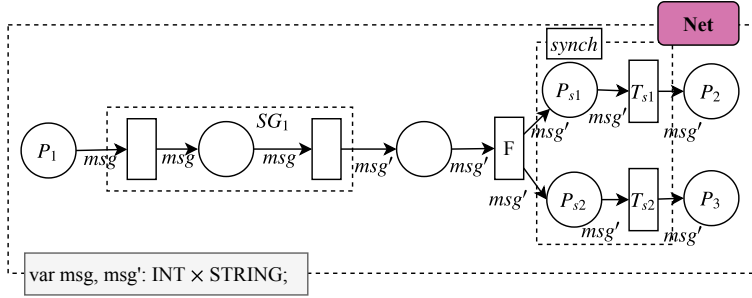
Definition 4.9 (Bisimulation). Let $\Gamma_{s_0}^B = \langle S, s_0, \rightarrow \rangle$ and $\Gamma_{s'_0}^{B'} = \langle S', s'_0, \rightarrow' \rangle$ be the associated labelled transition systems of two timed db-nets B and B' with the same boundaries from [Sections 3.1.2](#) and [3.2.3](#), respectively. We say that a B -snapshot (I, m) is functionally *equivalent* to a B' -snapshot (I', m') , $(I, m) \approx (I', m')$, if $I = I'$, and m and m' agree on output places, i.e., for every output place p with $m(p) = (\alpha, \gamma, age)$ and $m'(p) = (\alpha', \gamma', age')$, we have $\alpha = \alpha'$, for the elements that are in the message α , and those that are not required by any endpoint γ (usually $\gamma = \emptyset$), and age is the timed db-net age information.

Further we say that $\Gamma_{s_0}^B$ is functionally *equivalent* to $\Gamma_{s'_0}^{B'}$, $\Gamma_{s_0}^B \sim \Gamma_{s'_0}^{B'}$, if whenever $s_0 \rightarrow^* (I, m)$ then there is (I', m') such that $s'_0 \rightarrow'^* (I', m')$ and $(I, m) \approx (I', m')$ and $\Gamma_{(I, m)}^B \sim \Gamma_{(I', m')}^{B'}$, and whenever $s'_0 \rightarrow'^* (I', m')$ then there is (I, m) such that $s_0 \rightarrow^* (I, m)$ and $(I', m') \approx (I, m)$. ■

Note that this bisimulation definition neglects the unused fields γ as well as the age of the tokens age , since γ is used for deciding on the efficiency of a data reducing optimization only and



(a) Before applying the rewrite rule



(b) After applying the rewrite rule

Figure 4.10: Timed db-net translation of IPCGs before and after applying the “combine sibling patterns” rewrite rule

the age information is anyway reset when moved to a new place (cf. Section 3.1.2). Let us discuss an explanatory example of bisimulation.

Example 4.10. Figure 4.10 shows the interpretation of a simple IPCG as a timed db-net before and after applying the rewrite rule for the combining sibling patterns from Figure 4.4 (for simplicity without boundaries). The improvement of the optimization is to move SG_1 (isomorphic to SG_2) in front of the forking pattern F and leave out SG_2 , which reduces the modeling complexity on the right hand side (cf. Figure 4.10(b)). The synchronization subnet *synch* is required to show bisimilarity between the original and the resulting net, since tokens might be moved independently in SG_1 and SG_2 before applying the optimisation. The subnet (essentially transitions T_{s1}, T_{s2}) compensates for that to ensure that places P_2 and P_3 can be reached independently as well. Hence according to Definition 4.9, the timed db-net B representing Figure 4.10(a) and B' Figure 4.10(b) are bisimilar $\Gamma_{(I,m)}^B \sim \Gamma_{(I,m')}^{B'}$ for any database instance I , and any markings m and m' with $m(P_1) = m'(P_1)$ and $m(p) = \{\emptyset\}$, $m'(p) = \{\emptyset\}$ for all other p . ■

With the following congruence relation for the composition, we subsequently show the correctness of the optimizations by discussing the bisimilarity of the right and left hand sides of the respective optimization rules (first left to right, then right to left).

Lemma 4.11. The relation \sim is the congruence relation with respect to composition of timed db-nets with boundaries, i.e., it is reflexive, symmetric and transitive. If $\Gamma_{s_0}^{B_1} \sim \Gamma_{s_0}^{B'_1}$, $\Gamma_{s_0}^{B_2} \sim \Gamma_{s_0}^{B'_2}$ for all s_0 on the shared boundary of B_1 and B_2 , then $\Gamma_{s_0}^{B_1 \circ B_2} \sim \Gamma_{s_0}^{B'_1 \circ B'_2}$. □

Lemma 4.12. Let G and G' be IPCGs. If $G \Rightarrow_{r,g} G'$ for an optimisation r , then $\llbracket G \rrbracket$ and $\llbracket G' \rrbracket$ have the same boundary.

Theorem 4.13 (Optimization Correctness). With Lemma 4.12, let $s_0 = (I_0, m_0)$ be a snapshot for its timed db-net with boundaries $\llbracket G \rrbracket$, and $s'_0 = (I'_0, m'_0)$ a snapshot for $\llbracket G' \rrbracket$, such that $I_0 = I'_0$, and $m_0(p) = m'_0(p)$ for all places p on the (shared) boundary of $\llbracket G \rrbracket$ and $\llbracket G' \rrbracket$, with m_0 and m'_0 otherwise empty. If $G \Rightarrow_{r,g} G'$ for some optimization r and match g , then $\Gamma_{s_0}^{\llbracket G \rrbracket}$ is equivalent to $\Gamma_{s'_0}^{\llbracket G' \rrbracket}$.

Proof. In each step, the rewriting modifies certain nodes. By Lemma 4.11, it is enough to show that the affected parts of the interpretation of the graphs are equivalent, which is done for each optimization subsequently.

Redundant Sub-Processes. Each move on the left hand side of the optimization rule in Figure 4.3 (on page 144) either moves tokens into a cloud, out of a cloud, or inside a cloud. In the first two cases, this can be simulated by the right hand side by moving the token through the CE or CBR and CF respectively followed by a move into or out of the cloud, while in the latter case the corresponding token can be moved in SG'_1 up to the isomorphism between SG'_1 and the cloud on the left.

Similarly, a move on the right hand side into or out of the cloud can easily be simulated on the left hand side. Suppose a transition fires in SG'_1 . Since all guards in SG'_1 have been modified to require all messages to come from the same enriched context, the corresponding transition can either be fired in SG_1 or SG_2 .

Combining Sibling Patterns. Suppose the left hand side of Figure 4.4 (on page 145) takes a finite number of steps and ends up with $m(P_2)$ tokens in P_2 and $m(P_3)$ tokens in P_3 . There are three possibilities: (i) there are tokens of the same color in both P_2 and P_3 ; or (ii) there is a token in P_2 with no matching token in P_3 ; or (iii) there is a token in P_3 with no matching token in P_2 . For the first case, the right hand side can simulate the situation by emulating the steps of the token ending up in P_2 , and forking it in the end. For the second case, the right hand side can simulate the situation by emulating the steps of the token ending up in P_2 , then forking it, but not moving one copy of the token across the boundary layer in the interpretation of the fork pattern. The third case is similar, using that SG_2 is isomorphic to SG_1 .

The right hand side can easily be simulated by copying all moves in SG_1 into simultaneous moves in SG_1 and the isomorphic SG_2 .

Early-Filter. By construction, the filter removes the data not used by P_2 , so if the left hand side of Figure 4.5(a) (on page 146) moves a token to P_2 , then the same token can be moved to P_2 on the right hand side and vice versa.

Early-Mapping. Suppose the left hand side of Figure 4.5(b) (on page 146) moves a token to P_4 . The same transitions can then move the corresponding token to P_4 on the right hand side, with the same payload, by construction. Similarly, the right hand side can be simulated by the left hand side.

Early-Aggregation. The interpretation of the subgraph SG_2 is equivalent to the interpretation of P_1 followed by SG'_2 followed by P_3 , by construction in Figure 4.6(a) (on page 147), hence the left hand side and the right hand side are equivalent.

Early Claim Check. Since the claim check CC + CE in Figure 4.6(b) (on page 147) simply stores the data and then adds it back to the message in the CE step, both sides can simulate each other.

Early-Split. By assumption, P_1 followed by SSQ_1 (P_1 followed by SSQ_1 followed by P_2 for the inserted early split in Figure 4.7(a) (on page 148)) is equivalent to SSQ_1 followed by P_1 , from which the claim immediately follows.

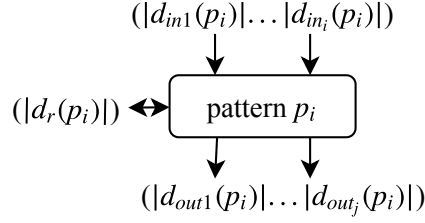


Figure 4.11: Data inputs, outputs and external resources of a pattern

Sequence to Parallel. The left hand side of Figure 4.8(a) (on page 149) can be simulated by the right hand side by copying each move in SSQ_1 by a move each in SSQ'_1 to SSQ'_n . If the right hand side moves a token to an output place, it must move a token through SSQ'_1 , and the same moves can move a token through SSQ_1 in the left hand side.

Merge Parallel. When left hand side of Figure 4.8(b) moves a token to the output place, it must move a token through SSQ'_1 , and the same moves can move a token through SSQ_1 in the right hand side. The right hand side can be simulated by the left hand side by copying each move in SSQ_1 by a move each in SSQ'_1 to SSQ'_n .

Heterogeneous Sequence to Parallel. We assume side-effect free sub-sequences SSQ_1 to SSQ_n for the rest of the proof. The right hand side of Figure 4.9 (on page 150) can simulate the left hand side as follows: if the left hand side moves a token to an output place, it must move it through all of SSQ_1 to SSQ_n . The right hand side can make the same moves in the same order. For the other direction, the left hand side can reorder the moves of the right hand side to first do all moves in SSQ_1 , then in SSQ_2 and so on. This is still a valid sequence of steps because of side-effect-freeness. \square

Note that for each new optimization to be correctness-preserving, the optimization itself has to be correct, and thus has to be proven according to Definition 4.9 (on page 150).

4.3 Evaluation

For the evaluation we define an abstract cost model, apply the optimizations to integration scenarios from a commercial cloud integration system in a quantitative analysis, and exemplify the results by two real-world case studies.

4.3.1 Abstract Cost Model

In order to decide if an optimization is an improvement or not, we want to associate abstract costs to integration patterns. We do this on the pattern level (cf. structural and data properties in Figure 3.33 (on page 105)), similar to the work on data integration operators [BHLW09]. The cost of the overall integration pattern can then be computed as the sum of the cost of its constituent patterns. Costs are considered parametrized by the cardinality of data inputs $|d_{in_i}|$ ($1 \leq i \leq n$, if the pattern has in-degree n), data outputs $|d_{out_j}|$ ($1 \leq j \leq m$, if the pattern has out-degree m), and external resource data sets $|d_r|$, as illustrated in Figure 4.11 and specified in Definition 4.14. The costs can also refer to the pattern characteristics.

Definition 4.14 (Cost model). A *cost assignment* for an IPCG $(P, E, type, char, inContr, outContr)$ is an function $cost(p) : \mathbb{N}^n \times \mathbb{N}^k \times \mathbb{N}^r \rightarrow \mathbb{Q}$ for each $p \in P$, where p has in-degree n , out-degree k and r external connections. The cost $cost(G) : \mathbb{N}^N \times \mathbb{N}^K \times \mathbb{N}^R \rightarrow \mathbb{Q}$ of an IPCG pattern graph $G = (P, E, type, pc, ic, oc)$ with a cost assignment, where N is the sum of the in-degrees of its patterns, K the sum of their out-degrees, and R the sum of their external connections, is defined to be the sum of the costs of its constituent patterns:

$$cost(G)(d_{in}, d_{out}, d_r) = \sum_{p \in P} cost(p)(|d_{in}(p)|, |d_{out}(p)|, |d_r(p)|)$$

where we suggestively have written $|d_{in}(p)|$ for the projection from the tuple d_{in} corresponding to p , similarly for $|d_{out}(p)|$ and $|d_r(p)|$. ■

We have defined the abstract costs of the patterns discussed in this work in Table 4.3 — these will be used in the subsequent evaluation. We now explain the reasoning behind them. Routing patterns such as content based routers, message filters and aggregators mostly operate on the input message, and thus have an abstract cost related to its element cardinality $|d_{in}|$. For example, the abstract cost of the Content-based Router is $cost(CBR) = \frac{\sum_{i=0}^{n-1} |d_{in,i}|}{2}$, since it evaluates on average $\frac{n-1}{2}$ routing conditions on the input message. More complex routing patterns such as aggregators evaluate correlation and completion conditions, as well as an aggregation function on the input message, and also on sequences of messages of a certain length from an external resource. Hence the cost of an aggregator is $cost(AGG) = 2 \times |d_{in}| + \frac{|d_{in}| + |d_r|}{avg(len(seq))}$, where $len(seq)$ denotes the length of a Message Sequence [HW04] as for example used by an Aggregator pattern. In contrast, message transformation patterns like content filters and enrichers mainly construct an output message, hence their costs are determined by the output cardinality $|d_{out}|$. For example, content enrichers create a request message from the input message with cost $|d_{in}|$, conducts an optional resource query $|d_r|$, and creates and enriches the response with cost $|d_{out}|$. Finally, the cost of message creation patterns such as external calls, receivers, and senders arise from costs for transport, protocol handling, and format conversion, as well as decompression. Hence the cost depends on the element cardinalities of input and output messages $|d_{in}|, |d_{out}|$.

Example 4.15. We return to the claimed improved composition in Example 3.36 (on page 109). The latency of the composition G_1 in Figure 3.34, calculated from the constituent pattern latencies, is $cost(G_1) = t_{CE} + t_{MT}$ with latency t_p and pattern p . The latency improvement potential given by switching to the composition G_2 in Figure 4.1 (on page 137) is given by $cost(G_2) = \max(t_{CE}, t_{MT}) + t_{MC} + t_{JR} + t_{AGG}$. Obviously it is only beneficial to switch if $cost(G_2) < cost(G_1)$, and this condition depends on the concrete values involved. At the same time, the model complexity increases by three nodes and edges. ■

4.3.2 Quantitative Analysis

We applied the optimization strategies OS-1–3 to 627 integration scenarios from the 2017 standard content of the SAP CPI (called ds17 below), and compared with 275 scenarios from 2015 (called ds15). Our goal is to show the applicability of our approach to real-world integration scenarios, as well as the scope and trade-offs of the optimization strategies. The comparison with a previous content version features a practical study on content evolution. To analyze the difference between different scenario domains, we grouped the scenarios into the following categories according to Chapter 2: On-Premise to Cloud (OP2C), Cloud to Cloud (C2C), and Business to Business (B2B). Since hybrid integration scenarios such as OP2C target the extension or synchronization of business data objects, they are usually less complex. In contrast native cloud application

Table 4.3: Abstract costs of relevant patterns

Pattern p	Abstract Cost $cost(p)$	Factors
Content-based Router [HW04]	$\frac{\sum_{i=0}^{n-1} d_{in,i} }{2}$	$n = \# \text{channel conditions}$, half of them evaluated on average
Message Filter [HW04]	$ d_{in} $	input data condition $ d_{in} $
Aggregator [HW04]	$2 \times d_{in} + \frac{ d_{in} + d_r }{avg(len(seq))}$	correlation, and completion conditions $ d_{in} $, aggregation function $\frac{ d_{in} + d_r }{avg(len(seq))}$ and length of a sequence $length(seq) \geq 2$, and (transacted) resource d_r
Claim Check [HW04]	$2 \times d_r $	resource insert and get $ d_r $
Splitter [HW04]	$ d_{out} $	output data condition $ d_{out} $
Multicast, Join Router [RMRM17]	$\sum_{i=0}^n cost(procunit_i)$	costs of processing units $cost(procunit_i)$, e.g., threading in software, for n channels
Content Filter [HW04]	$ d_{out} $	output data creation $ d_{out} $
Mapping [HW04]	$ d_{in} + d_{out} $	output data creation $ d_{out} $ from input data $ d_{in} $
Content Enricher [HW04]	$ d_{in} + d_r + d_{out} $	request message creation on $ d_{in} $, resource query $ d_r $, response data enrich $ d_{out} $
External	$ d_{out} + d_{in} $	request $ d_{out} $ and reply data $ d_{in} $
Call [RMRM17]	$ d_{in} $	input data $ d_{in} $
Receive [HW04]	$ d_{in} $	input data $ d_{in} $
Send [HW04]	$ d_{out} $	output data $ d_{out} $

scenarios such as C2C or B2B mediate between several endpoints, and thus involve more complex integration logic [RMRM17]. The process catalog also contained a small number of simple Device to Cloud scenarios; none of them could be improved by our approach, due to their simplicity.

Setup: Construction and analysis of IPCGs For the analysis, we constructed an IPCG for each integration scenario following the workflow sketched in Figure 4.12. The integration scenarios are stored as process models in a BPMN-like notation similar to Figure 3.23 (on page 95). The process models reference data specifications (short *ds*) such as schemas (e.g., XSD, WSDL), mapping programs, selectors (e.g., XPath) and configuration files. For every pattern used in the process models, runtime statistics are available from benchmarks (cf. Chapter 5). The data specifications are picked up from the 2015 content archive and from the current 2017 content catalog, while the runtime benchmarks are collected using the open-source integration system *Apache Camel* [IA10] as used in SAP CPI [SAP19a]. All measurements were conducted on a HP Z600 workstation, equipped with two Intel X5650 processors clocked at 2.67GHz with a 12 cores, 24GB of main memory, running a 64-bit Windows 7 SP1 and a JDK version 1.7.0, with 2GB heap space. The mapping and schema information is automatically mined and added to the patterns as contracts, and the rest of the collected data as pattern characteristics. For each integration scenario and each optimization strategy, we determine if the strategy applies, and if so, if the cost is improved. We continue until no further strategy applies. This analysis runs in about two minutes in total for all 902 scenarios on our workstation.

We now discuss the improvements for the different kinds of optimization strategies identified

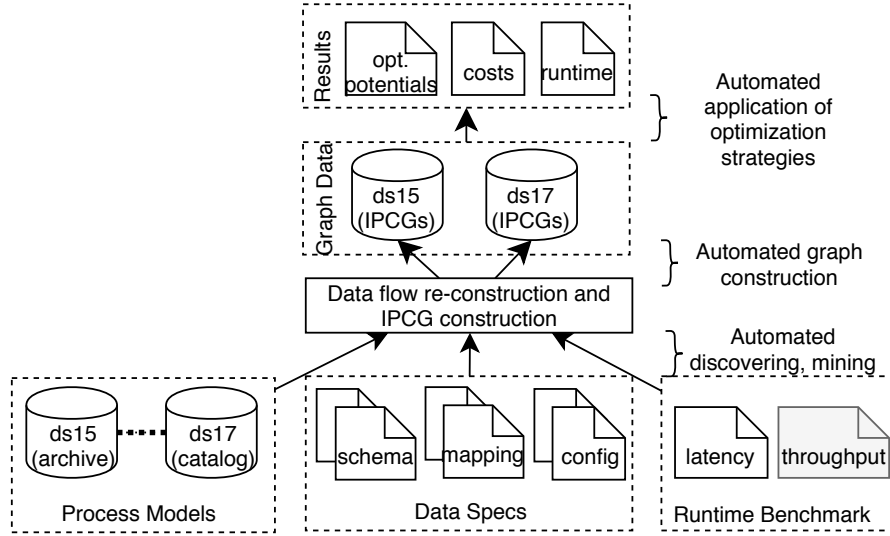


Figure 4.12: Pattern composition evaluation pipeline

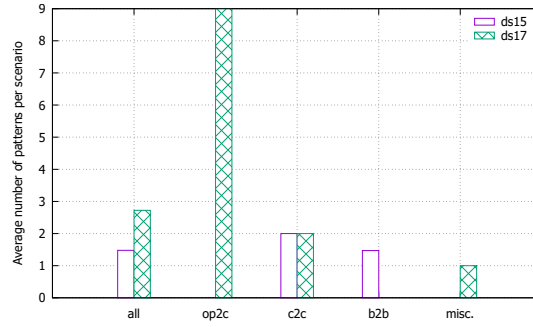


Figure 4.13: Pattern reduction per scenario

in Section 4.1.

Improved Model Complexity: Process Simplification (OS-1). The relevant metric for the process simplification strategies from OS-1 is the model complexity, i.e. the average number of pattern reductions per scenario, shown in Figure 4.13.

Results. Although all scenarios were implemented by integration experts, who are familiar with the modeling notation and the underlying runtime semantics, there is still a small amount of patterns per scenario that could be removed without changing the execution semantics. On average, the content reduction for the content from 2015 and 2017 was 1.47 and 2.72 patterns / IPCG, respectively, with significantly higher numbers in the OP2C domain.

Conclusions. (1) Even simple process simplifications are not always obvious to integration experts in scenarios represented in a control-flow-centric notation (e.g., current SAP CPI does not use BPMN Data Objects to visualize the data flow); and (2) the need for process simplification does not seem to diminish as integration experts gain more experience.

Improved Bandwidth: Data Reduction (OS-2). Data reduction impacts the overall band-

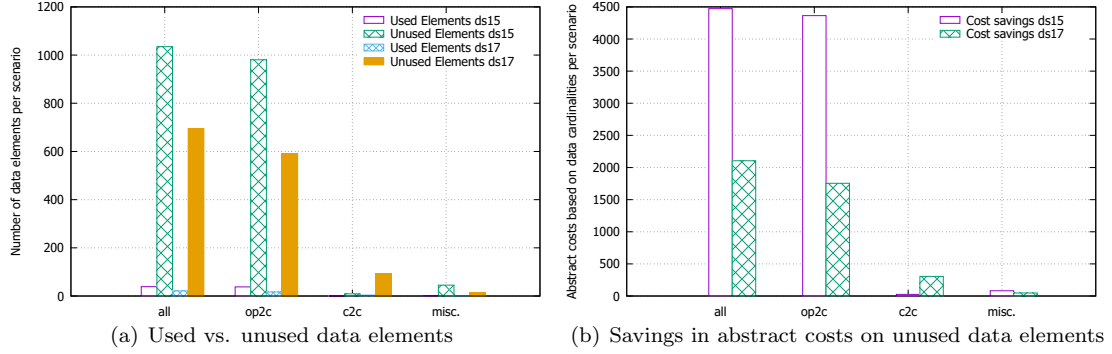


Figure 4.14: Unused elements in integration scenarios

width and message throughput (cf. Section 6.2). To evaluate data reduction strategies from OS-2, we leverage the data element information attached to the IPCG contracts and characteristics, and follow their usages along edges in the graph, similar to “ray tracing” algorithms [Gla89]. We collect the data elements that are used or not used, where possible — we do not have sufficient design time data to do this for user-defined functions or some of the message construction patterns, such as request-reply. Based on the resulting data element usages, we calculate two metrics: the comparison of used vs. unused elements in Figure 4.14(a), and the savings in abstract costs on unused data elements in Figure 4.14(b).

Results. There is a large amount of unused data elements per scenario for the OP2C scenarios; these are mainly web service communication and message mappings, for which most of the data flow can be reconstructed. This is because the predominantly used EDI and SOA interfaces (e.g., SAP IDOC, SOAP) for interoperable communication with on-premise applications define a large set of data structures and elements, which are not required by the cloud applications, and vice versa. In contrast, C2C scenarios are usually more complex, and mostly use user-defined functions to transform data, which means that only a limited analysis of the data element usage is possible.

When calculating the abstract costs for the scenarios with unused fields, there is an immense cost reduction potential for the OP2C scenarios as shown in Figure 4.14(b). This is achieved by adding a content filter to the beginning of the scenario, which removes unused fields. This results in a cost increase $|d_{in}| = \# \text{unused elements}$ for the content filter, but reduces the cost of each subsequent pattern.

Conclusions. (3) Data flows can best be reconstructed when design time data based on interoperability standards is available; and (4) a high number of unused data elements per scenario indicates where bandwidth reductions are possible.

Improved Latency: Parallelization (OS-3). For the sequence-to-parallel optimization strategies from OS-3, the relevant metric is the processing latency of the integration scenario. Because of the uncertainty in determining whether a parallelization optimization would be beneficial, we first report on the classification of parallelization candidates in Figure 4.15. We then report both the improvements according to our cost model in Figure 4.16(a), as well as the actual latency improvement in Figure 4.16(b).

Results. Based on the data element level, we classify scenario candidates as **parallel**, definitely **non parallel**, or **potentially parallel** in Figure 4.15. The uncertainty is due to the incomplete data flow information available for these scenarios. From the 2015 catalog, 81% of the

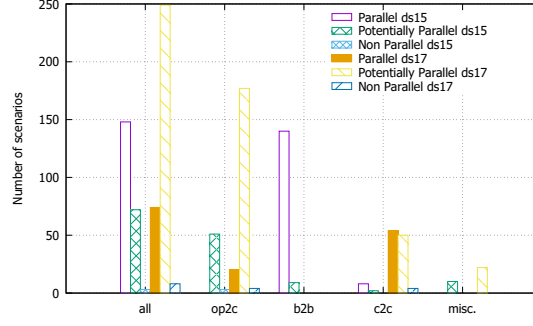


Figure 4.15: Parallelization scenario candidates

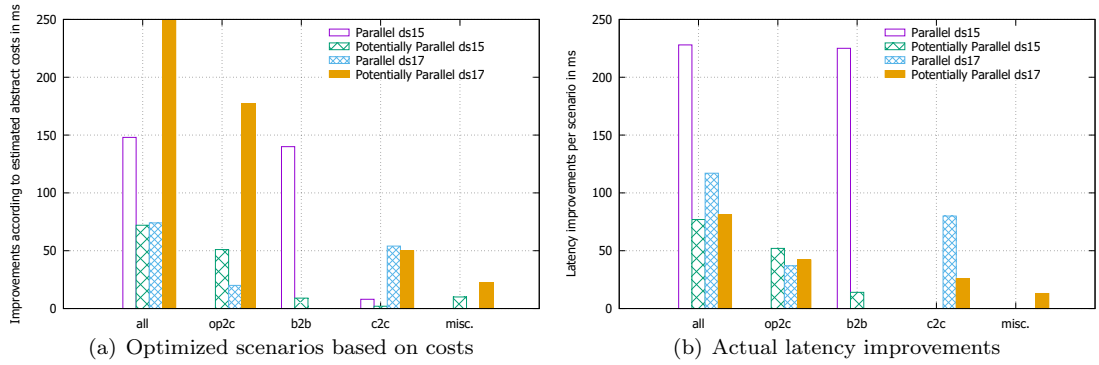


Figure 4.16: “Sequence to parallel” optimization candidates on (a) integration flows, (b) optimization selection based on abstract cost model, and (3) actual latency improvements

scenarios are classified as **parallel**, or **potentially parallel**, while the number for the 2017 catalog is 53%. In both cases, the OP2C and B2B scenarios show the most improvement potential. Figure 4.16(a) shows the selection based on our cost model, which supports the pre-selection of all of these optimization candidates. The actual, average improvements per impacted scenario are shown in Figure 4.16(b). The average improvements of up to 230 milliseconds per scenario must be understood in the context of the average runtime per scenario, which is 1.79 seconds. We make two observations: (a) the costs of the additional fork and join constructs in Java are high compared to those implemented in hardware (cf. Section 6.2), and the improvements could thus be even better, and (b) the length of the parallelized pattern sequence is usually short: on average 2.3 patterns in our scenario catalog.

Conclusions. (5) The parallelization requires low cost fork and join pattern implementations; and (6) better runtime improvements might be achieved for scenarios with longer parallelizable pattern sequences.

4.3.3 Case Studies

We apply, analyze and discuss the proposed optimization strategies in the context of two case studies: the Replicate Material on-premise to cloud scenario from Figure 3.23 (on page 95), as well

as an SAP eDocument invoicing cloud to cloud scenario. These scenarios are part of the SAP CPI standard, and thus many users (i.e., SAP’s customers) benefit immediately from improvements. For instance, we additionally implemented a content monitor pattern (cf. [Chapter 2](#)) that allowed analysis of the SAP CPI content. This showed the Material Replicate scenario was used by 546 distinct customers in 710 integration processes copied from the standard into their workspace — each one of these users is affected by the improvement.

Replicate Material (revisited). Recall the Replicate Material scenario is concerned with enriching and translating messages coming from a CRM before passing them on to a Cloud for Customer service, as in [Figure 3.23](#). As already discussed, the content enricher and the message translator can be parallelized according to the sequence to parallel optimization from OS-3. The original and resulting IPCGs are shown in [Figures 3.34](#) and [4.1](#) (on [page 109](#) and [page 137](#), respectively). No throughput optimizations apply.

Latency improvements. The application of this optimization can be considered, if the latency of the resulting parallelized process is smaller than the latency of the original process, i.e. if

$$\begin{aligned} \text{cost}(MC) + \max(\text{cost}(CE), \text{cost}(MT)) + \text{cost}(JR) + \text{cost}(AGG) \\ < \text{cost}(CE) + \text{cost}(MT) \end{aligned}$$

Subtracting $\max(\text{cost}(CE), \text{cost}(MT))$ from both sides of the inequality, we are left with

$$\text{cost}(MC) + \text{cost}(JR) + \text{cost}(AGG) < \min(\text{cost}(CE), \text{cost}(MT))$$

If we assume that the content enricher does not need to make an external call, its abstract cost becomes $\text{cost}(CE)(|d_{in}|, |d_r|) = |d_{in}|$, and plugging in experimental values from a pattern benchmark (cf. [Chapter 5](#)), we arrive at the inequality (with latency costs in seconds)

$$0.01 + 0.002 + 0.005 \not< \min(0.005, 0.27)$$

which tells us that the optimization is not beneficial in this case — the additional overhead is larger than the saving. However, if the content enricher does use a remote call, $\text{cost}(CE)(|d_{in}|, |d_r|) = |d_{in}| + |d_r|$, and the experimental values now say $\text{cost}(CE) = 0.021$. Hence the optimization is worthwhile, as

$$0.01 + 0.002 + 0.005 < \min(0.021, 0.27) .$$

Model Complexity. Following Sánchez-González et al. [[SGGM⁺10](#)], we measure the model complexity as the node count. Hence, in this case, the optimization increases the complexity by three (nodes).

Conclusions. (7) The pattern characteristics are important when deciding if an optimization strategy should be applied (e.g., local vs. remote enrichment); and (8) there are goal conflicts between the different objectives, as illustrated by the trade-off between latency reduction and increasing model complexity.

eDocuments: Italy Invoicing. The Italian government accepts electronic invoices from companies, as long as they follow regulations — they have to be correctly formatted, signed, and not be sent in duplicate. Furthermore, these regulations are subject to change. This can lead to an ad-hoc integration process such as in [Figure 4.17](#) (simplified). Briefly, the companies’ *Fattura Elettronica* is used to generate a *factorapa* document with special header fields (e.g., *Paese*, *IdCodice*), then the message is signed and sent to the authorities, if it has not been sent previously. The multiple authorities respond with standard *Coglienza*, *Risposta* acknowledgments, that are transformed to a *SendInvoiceResponse*. We transformed the BPMN model to an IPCG, tried to apply optimizations, and created a BPMN model again from the optimized IPCG.

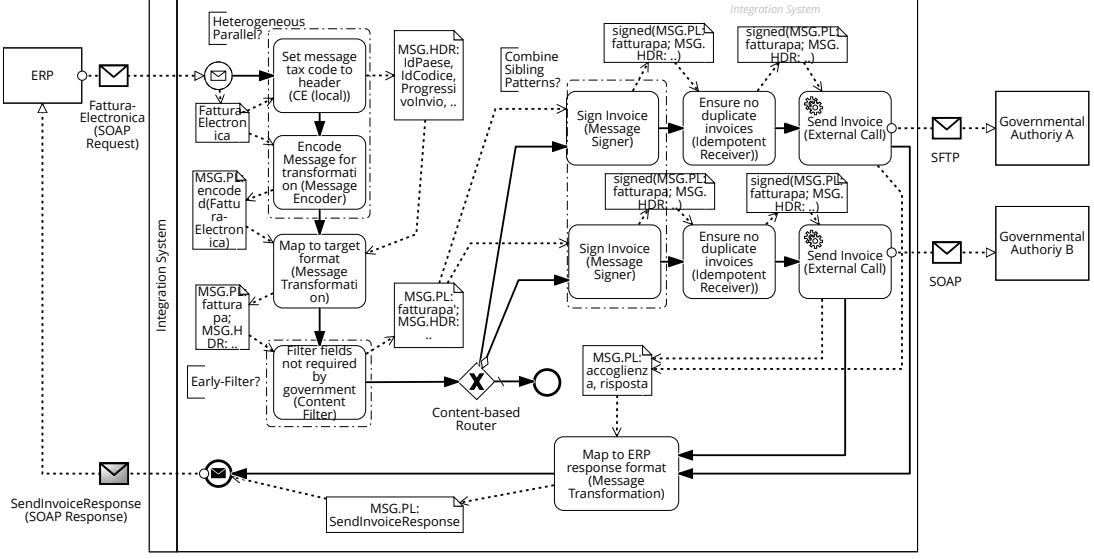


Figure 4.17: Country-specific invoicing (potential improvements as BPMN Group)

Our heuristics for deciding in which order to try to apply different strategies are “simplification before parallelization” and “structure before data”, since this seems to enable the largest number of optimizations. Hence we first try to apply OS-1 strategies: the *combine siblings* rule matches the sibling Message Signers and Idempotent Receivers, since the preceding Content-based Router is a fork. Next we try OS-3 strategies. Although *heterogeneous parallelization* matches for the CE and the Message Encoder, it is not applied since

$$\text{cost}(MC) + \text{cost}(JR) + \text{cost}(AGG) \not\leq \min(\text{cost}(CE), \text{cost}(ME)),$$

i.e., the overhead is too high, due to the low-latency, local CE. Finally, the *early-filter* strategy from OS-2 is applied for the Content Filter, inserting it between the Content Enricher and the Message Encoder. No further strategies can be applied. The resulting integration process translated back from IPCTG to BPMN is shown in Figure 4.18.

Conclusions. (9) The application order OS-1, OS-3, OS-2 seems most beneficial (“simplification before parallelization”, “structure before data”); (10) an automatic translation from IPCGs to concepts like BPMN could be beneficial for connecting with existing solutions.

4.4 Related Work

We presented related optimization techniques in Section 4.1 and adapted them to application integration. These techniques mostly consider correctness on a structural level, e.g., using data dependency constraints [BHP⁺11, KG14, KGS17] that are somewhat comparable to contract graphs. While most of them used a (direct acyclic) graph representation, as in our case, none of them specifies execution semantics and shows guarantees down to the execution level. Subsequently, we compare our approach to related work on correctness, (data) dependency techniques and graph transformation in the context of processes.

There is work on formal representations of integration patterns, e.g., Mederly et al. [MLZN09] represents messages as first-order formulas and patterns as operations that add and delete formulas,

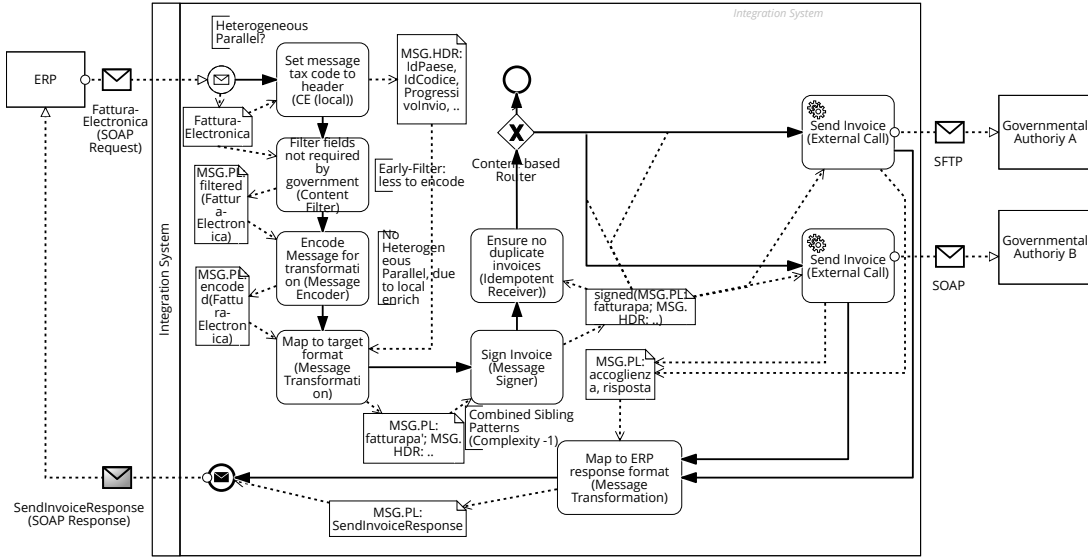


Figure 4.18: Invoice processing from Figure 4.17 after application of strategies OS-1–3

and then applies AI planning to find an process with a minimal number of components. While this approach shares the model complexity objective, our approach applies to a broader set of objectives and optimization strategies.

Semantic Program Correctness The semantic correctness plays a bigger role in the compiler construction and analysis domain, for optimizing compilers as in our work. For example, Muchnick [Muc97] provides an exhaustive catalog of optimizing transformations and states that the proof of the correctness of rewritings must be based on the (execution) semantics. For semantics-based program manipulation, the more theoretical work on abstract interpretation by Cousot and Cousot [CC77] and the more pragmatic by Aho et al. [ALSU06], e.g., by Lacey et al. [LJVWF04] using temporal logic to perform and analyze optimizing transformations like dead code elimination on imperative programs. For a semantics-based transformation correctness Nielson [Nie81] provides semantic correctness proofs using data-flow analysis (e.g., for constant folding) and Cousot [CC02] provides a general framework for designing program transformations analyzing abstract interpretations. While our contract graphs and optimizations are far simpler than those more general programming language transformations, our translation of IPCGs to timed db-nets with boundaries can be seen as a concretization in the sense of an abstract interpretation, and thus giving a similar notion of semantic correctness.

Analysis and Optimization Structures Transformation techniques for optimization have been employed by compiler construction, e.g., for parallel [KMC72] or pipeline processing [KKLW80]. Thereby dependence graph representations became especially useful. For example, Kuck et al. [KKP+81] construct dependence graphs with (data) output-, anti-, and flow-dependencies of a program as a foundation for the optimizing transformations. These kind of dependence graphs were also used by [BHP+11], however, are “linearized” in form of our pattern contracts. This makes the decision of the optimization “local” and does not require dependence graph abstractions like intervals [Coc70] or scoping [ZB74]. More recently these techniques have been applied for business process optimization by Sadiq [SO00], Niedermann et al. [NRM11, NS11] or reductions to process tree structures [VVK09] with incremental transformations [HFKV08]. In

our case the scope of the analysis is a local match of pattern contracts.

Graph Transformations Similar to our approach, graph transformations have been used in related domains, e.g., formalizing parts of the BPMN semantics by Dijkman et al. [DG10], who specify the execution semantics as graph rewrites. Conformance is checked experimentally and verification is left for future work. For the optimizations, we use the same visual notation and double-pushout rule application approach. However, our execution semantics are given as timed db-net and can be verified.

4.5 Conclusions

For correctness-preserving optimizations of integration scenarios (cf. RQ2-3: “*What are relevant optimization strategies and how can they formally defined on pattern compositions?*”), we started according to sub-question RQ2-3(a): *What are relevant optimization techniques for EAI pattern compositions?* with a compilation of optimizations from the literature that we mapped to application integration and categorized by their impact to optimization strategies: complexity (cf. OS-1: process simplification), message throughput through reduced element cardinalities (cf. OS-2: data reduction), as well as throughput and processing latency (cf. OS-3: parallelization). We then developed a formalization of optimizations that we base on the pattern composition formalism DSR artifact from Chapter 3 in order to precisely define optimization for sub-question RQ2-3(b): *How can optimization strategies be formally defined?*. Due to the selected algebraic optimization formalism as well as the formal foundations (down to execution semantics), the optimizations preserve the correctness of a pattern composition (cf. RQ2-3(c): “*How can the application of optimization strategies preserve the compositional correctness?*”). In summary, these questions are answered by the formalization of optimization strategies in this chapter that denote contributions in form of DSR artifacts:

- A catalog of optimization strategies, documented in [RFRM19] (\rightarrow RQ2-3(a)),
- A formalism for defining and applying correctness-preserving optimizations (incl. proofs; \rightarrow RQ2-3(b) and RQ2-3(c)),
- An instantiation of the formalized optimizations in the form of a prototype.

Then we evaluated our approach on data sets containing in total over 900 real world integration scenarios, and two case studies. We conclude that formalization and optimizations are relevant even for experienced integration experts (conclusions 1–2), with interesting choices (conclusions 3–4, 6), implementation details (conclusions 5, 10) and trade-offs (conclusions 7–9).

While we developed the formal foundations of pattern compositions further to a correctness-preserving transformation approach that is illustrated by but not limited to optimizations, we subsequently summarize and discuss limitations and open research challenges.

Limitations and Open Research Challenges Limitations of the approach concern the literature review, for which the search was led by the selection of keywords and criteria due to the vast amount of existing work and in order to not lose focus of this study. Nonetheless, conducting further *vertical* searches and expert additions that were not found based on the keywords could be included in the analysis. Consequently, the catalog of optimization strategies (cf. [RFRM19]) is not complete and will be subject to additions over time. However, more recent additions like “Ignore failing endpoints” or “Reduce interactions” could be formally defined and their correctness proven using our approach (cf. Appendix A).

The abstract cost model used for the evaluation is based on the abstract data flow. With the translations down to timed db-nets, more sophisticated cost models can be developed that take

the inherent complexities of the patterns into account. Moreover, more emphasis has to be put on studies that incorporate dynamic aspects into the formalization of patterns, again for a more precise cost semantics. In addition, purely data related techniques like message indexing, fork path re-ordering and merging of conditions can be analyzed for their effects. Finally, multi-objective optimizations and heuristics for graph rewriting on the process level like partitioning approaches have to be further studied.

The optimization rules are stratified according to their effects for the application (i.e., “simplification before parallelization” and “structure before data”). More elaborate rule application and execution schemes could be investigated to give better optimality guarantees.

So far, our approach targets scenario-level improvements only. Especially when considering technology trends, EAI systems and solutions could be realized more efficiently leveraging new technology suitable for message processing. Since this leaves a wide range of opportunities to improve current solutions, we study the most promising technology trends with respect to challenge C5 “Volocity” (i.e., Volume and Velocity) in [Chapter 6](#).

Impact The impact of correctness-preserving optimizations of integration pattern compositions lays the ground for trustworthy rewritings of integration scenarios beyond optimizations. This could become a key principle for responsible programming in shared responsibility environments like cloud or mobile computing. Moreover, together with the formalization of pattern compositions in [Chapter 3](#), it provides a comprehensive foundation for the pattern-based translation of integration scenarios (e.g., [\[Rit15a\]](#)), and was already partially picked up by industry (e.g., SAP Cloud Platform Integration [\[SAP19a\]](#)).

Part II

Realization

Chapter 5

Benchmarking Patterns

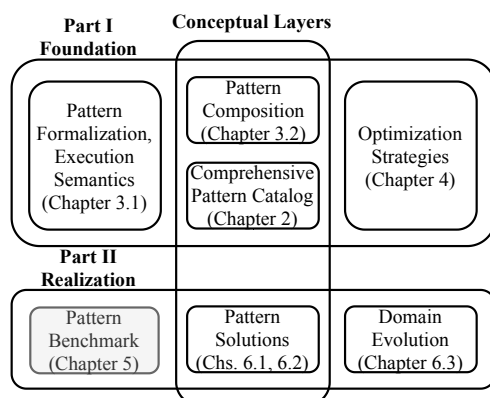
Contents

5.1	Integration Pattern System Implementations	170
5.2	Integration Scenario Analysis	178
5.3	Integration Pattern Benchmark	182
5.4	Evaluation	191
5.5	Related Work	195
5.6	Conclusions	198

Observation, reason, and experiment make up what we call the scientific method. [...] philosophers have said before that one of the fundamental requisites of science is that whenever you set up the same conditions, the same thing must happen. This is simply not true, it is not a fundamental condition of science.
Richard Feynman, 1963 [FLS63]

In Part I of this thesis (Chapters 2 to 4) we provided formalisms and prototypical implementations with the goal to validate and verify the functional correctness of patterns and their compositions (addressing challenges C1–C4).

The developed prototypes focused on *trustworthy* application integration through responsible programming and not on efficient message processing. In Chapters 1 and 2 we identified a strong need for fast, high volume messaging (cf. challenges C5 “Volocity”) and new message formats (cf. challenge C6 “Variety”). We partially addressed challenge C5 in Chapter 4, where we developed optimization strategies that improve the modeling and message processing (e.g., through data reduction, parallelization), while preserving the functional and compositional correctness of an improved scenario. While these improvements were achieved on an abstract integration scenario level, we also acknowledged



that more efficient pattern realizations or solutions on the system level could be beneficial, which we will study in [Chapter 6](#). However, to record and compare improvements on a system level for challenges C5 and C6, a well-defined method of assessment for pattern solutions is required, which we identified as a challenge in [Chapters 1 and 2](#) (cf. challenge C4 “Justification”) due to the current absence of such a method (cf. [Chapter 2](#) and [Section 5.5](#)).

In this chapter we define and develop concepts and a prototype for assessing the efficiency of pattern realizations in the form of a runtime benchmark of pattern realizations, called *EIPBench*, so that results can be reproduced under the same conditions (cf. [\[FLS63\]](#)). The notion of *justification* is a part of our scientific design science research method as *evaluation* with the deliverable *performance measures*, which we methodologically follow in this thesis (cf. [Section 1.3](#)). The discourse on scientific methods in the natural sciences (e.g., [\[FLS63, p. 1\]](#)) can also be found in information systems research. We recall that technology is embodied by implementations or artifacts, instead of nature itself as in the natural sciences, however, can be conceptual as well. In this way, information technology is the technology used to acquire and process information in support of human purposes, and thus instantiated as complex organizations of hardware, software, procedures and data in an information system [\[MS95\]](#). While information systems research targets questions like “does it work?” or “is it an improvement?”, natural science is concerned with how and why things are [\[MS95\]](#). However, in both design and natural sciences *justification* is required that tests claims for validity [\[MS95\]](#). In design science research [\[HC10\]](#) an artifact (e.g., design concept, prototype) has to be evaluated in what is called *deduction* as part of the cognitive research process [\[VK15\]](#). There, a tentative design is prototyped, which is studied and measured during the evaluation phase (cf. [Section 1.3](#)).

One way to allow for a justification and comparison of integration pattern realization improvements is by a *benchmark*. Since the development of a benchmark is difficult and requires careful design and implementation, we consider the basic design principles for benchmarking by Gray [\[Gra93\]](#), who stresses the need for domain-specific benchmarks, like pattern realizations in our case. The key benchmarking principles according to [\[Gra93\]](#) are:

Relevance Typical operations of the problem domain must be performed, when measuring the peak performance of systems,

Portability The benchmark shall be implemented on different systems and architectures without difficulty,

Scalability The benchmark should allow for scaling from smaller to larger as well as parallel computer systems,

Simplicity The benchmark must be comprehensible.

To be *relevant*, an integration pattern benchmark has to target common, domain-specific objectives of message throughput and processing latency (corresponding to challenge C5 “volatility” in [Section 1.2.2](#) and according to [\[HW04\]](#)) and derive characteristic properties from the respective domain challenges (e.g., message size). Complex routing patterns have to process the messages of diverse and complex data formats (i.e., nested, multi-format), and constitute a critical performance aspect of integration systems, which we showed in previous work on data-aware message processing [\[Rit15b\]](#). Aiming for reliable messaging guarantees, expressed through configurable message delivery semantics, adds a non-functional complexity to the message processing (cf. *relevance*). For example, the new trends discussed in [Chapter 2](#) (e.g., mobile and cloud computing) challenge classical integration systems because massive numbers of concurrent users, devices (i.e., message endpoints) and messages — with message sizes up to several hundred megabytes [\[SAP19a\]](#) — have to be processed (cf. *scalability*). Different challenges might need different solutions, which requires the applicability of the benchmark in different technologies

or systems (cf. *portable*). Finally, the comprehension of a pattern benchmark can be improved by staying close to integration domain aspects, which is supported by knowing the patterns and their characteristics (cf. *simplicity*).

Given the new challenges, researchers and practitioners in related areas have defined more data-aware benchmarks that fostered novel solutions and allow for comparing them. For instance, in the area of data integration TPC-DI [PRC14] was recently standardized. For analytical and (business) application processing, e.g., BigBench [DGV⁺14, BBN⁺12] targets end-to-end analytics processing, however, under-represents the integration aspect. Complementary, complex event- and stream processing benchmarks have been defined (e.g., [ACG⁺04, MBM08]). They focus on small portions of frequent data and analytical (stream) queries on actual and historic data. Recently developed IoT and cyber-physical system benchmarks [JNB14] add new notions and can be seen as variants of the existing benchmarks. They specifically target the analytical processing of data within these applications (e.g., mostly event processing). On the application integration side, some efforts were made as part of the SOA benchmark [SPE10], from which we take the ideas for the macroscale factors *concurrent client* and *flexible payload size*.

Despite the importance of application integration, many functional- and non-functional, performance-related questions cannot be answered today, due to the absence of a benchmark (cf. Chapter 2 and Section 5.5). To close this gap, we address the *measurability* and *comparability* sub-question RQ3-2 “How can the benefits and improvements of pattern implementations be measured, validated and compared?” of the general research question RQ3 “Which related concepts and technology trends can be used to improve integration processing and how can the resulting integration solutions be practically realized and compared?”. To account for the benchmarking principles, an answer to this question is driven by the following sub-questions:

- (a) What is the impact of complex message routing and transformation?
- (b) What is the impact of non-functional aspects such as message volume and concurrency?
- (c) What is the potential of new message processing approaches, and how can they be compared?

Question RQ3-2(a) targets *relevance* by considering complex routing, route-branchings and message delivery semantics, question RQ3-2(b) guides the design of the benchmark with respect to *scalability* with configurable message sizes and concurrent users, and question RQ3-2(c) with respect to *portability*, which includes the prototypical implementation of the benchmark. To answer these questions and to measure enhancements of current pattern implementations, we define a micro-benchmark for EIPs, including functional and non-functional aspects with a focus on message throughput for data-aware integration scenarios. The evaluation is conducted based on a prototypical instantiation of the benchmark, with which we assess one state-of-the-art system (cf. questions RQ3-2(a),(b)) that we compare to a novel message processing approach (cf. RQ3-2(c)). First, as system under test, we select Apache Camel [IA10] from our system overview in Section 2.2.3, since it is open-source and implements all of the original EIPs from [HW04]. Moreover, to the best of our knowledge, Apache Camel is used in production as an EAI system in SAP Cloud Integration (cf. Section 2.2.3) and MuleESB [Mul19], and thus denoting a state-of-the-art application integration system. Alternatives would be the open source integration systems Apache Flume and Nifi that we identified in Section 2.2.3. However, neither of these two systems covers all EIPs nor are any current, productive usages known. Second, we select *vectorization* of data (e.g., as described in [BHP⁺11]) as a message processing approach for the comparison with the current Apache Camel runtime. Vectorization denotes the processing of multiple instead of single records — or messages in our case — at the same time which is also called *micro-batching* or *horizontal parallelization* [Gra90]. While we already studied the scenario-level optimization strategies of data reduction and parallelization in Chapter 4, we argue that vectorization denotes

a more system level, data-centric message processing style. Besides streaming, which we address in [Section 6.2](#), horizontal data parallelization is also found in related, contemporary data flow systems like Apache Spark [[ZCF⁺10](#)] and Flink [[CKE⁺15](#)], and is thus a typical way to improve data processing. Moreover, we provided an extension of Apache Camel [[Rit15b](#)] for horizontal parallelization, which makes it a good fit for the comparison with the current Camel runtime. Note that the portability of our benchmark is not shown by this, but addressed in [Chapter 6](#).

For a better understanding, we briefly introduce Apache Camel as our benchmarked system candidate as well as the Camel vectorization extension in [Section 5.1](#). Furthermore, we set the formal concepts of integration pattern type and contract graphs and timed db-nets from [Chapter 3](#) into the context of the introduced, practical concepts. Then we further analyse and classify the scenario categories from [Chapter 2](#) in order to pay more attention to integration domain-specificities like “what are the most practically used patterns?” in [Section 5.2](#). This helps us to focus on the practically most relevant patterns during the evaluation. Following some ideas from the SPEC SOA initiative [[SPE10](#)], in [Section 5.3](#) we base the design of the benchmark on flexible, configurable scale-factors, such as a pattern or micro level and a general, macro benchmark level. In [Section 5.4](#) we describe our prototypical benchmark implementation, which we use to evaluate the current and extended Camel runtimes. We discuss related work in [Section 5.5](#), before concluding in [Section 5.6](#).

With this we define the first integration pattern benchmark that addresses the discussed challenges C4–5. The extension to multimedia data according to challenge C6 is described in [Section 6.3](#). As such, a portable, scalable benchmark is instrumental for the pattern realizations in the following chapter.

Parts of this chapter have previously been published in the proceedings of BICOD 2015 [[Rit15b](#)] (Apache Camel vectorization extension) and DEBS 2016 [[RMSRM16](#)] (pattern benchmark).

5.1 Integration Pattern System Implementations

While the benchmark definition constitutes an artifact that is independent of concrete system implementations, we briefly introduce two integration system realizations (i.e., Apache Camel, and a vectorization extension) for the evaluation of a prototypical implementation of the benchmark. Notably, the system concepts denote practical representations of our formal pattern and composition specifications, which we briefly discuss.

5.1.1 Apache Camel

We recall that Apache Camel [[IA10](#)] is a good fit as system-under-test for our integration pattern benchmark because it implements all original EIPs, being openly available and in practical use in large EAI system offerings.

Camel offers a runtime-near integration language in form of an internal domain specific language with language bindings into Java, Scala and others that denote an abstraction of the common application integration concepts. The integration language allows for the description of integration programs, called *Camel Routes*, which can be executed in the built-in Java-based integration pipeline and adapter runtimes. These routes denote executable runtime artifacts similar to timed db-nets from [Section 3.1.2](#) (without boundaries) translated from integration scenarios that are represented by integration pattern type graphs from [Section 3.2](#) (without pattern contracts). The routes can be executed similar to the timed db-net definitions, which can be simulated in our CPN Tools extension. In the same way the *Integration Pattern Type Graphs* (IPTGs) from [Section 3.2.2](#) allow for the composition of integration patterns that are then translated to executable timed db-nets, routes in Camel compose *Camel Processors* and

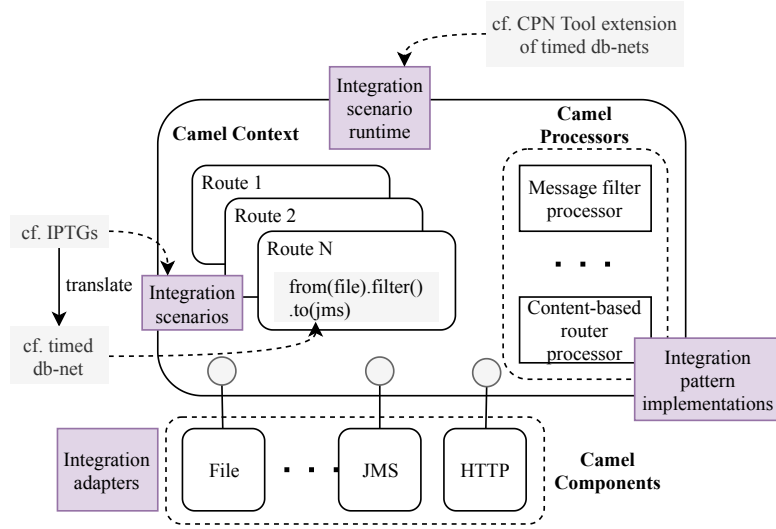


Figure 5.1: Apache Camel system architecture (adapted from [IA10])

integration adapters, called *Camel Components*, which denote a subset of the integration patterns. We argue that similar to the translation of IPCGs (i.e., IPTG with pattern contracts) to timed db-nets with boundaries, Camel routes could be translated from IPTGs. For example, based on our work in Chapter 3, something similar is done in SAP Cloud Platform Integration (i.e., on Camel runtime), however, without pattern contracts. The same is true for Camel, which does not have the concept of contracts or boundaries.

An overview of the architecture of Apache Camel is depicted in Figure 5.1, where *Camel Context* denotes the integration process engine (i.e., similar to the simulation capabilities in our CPN Tools extension) and Camel processors represent the *filters* that can either be Camel components, or the pattern implementations. For example, the Message Filter maps to the *filter* and the Splitter pattern to the *split* statement (latter not shown). The more than 150 integration adapters are specified in *from*, *to* statements for the inbound, outbound adapters, respectively.

During runtime, several of the routes, consisting of these statements, run in one context. Apart from the SAP CPI standard content [SAP18a], which we use for our analysis, there are no usable in production, pre-defined Camel routes. A *Service Registry* allows for the binding to services like persistence or security in the components and user-defined processors. A set of built-in and user-defined *Type Converters* automatically convert a message from one type to another (e.g., *InputStream* to *String*). For the materialization of the different data formats, *Marshallers* can be provided, if the built-in ones are not sufficient (not shown).

The data or message content is routed to Camel wrapped in an *Exchange*, shown in Figure 5.2, which, besides a unique identifier, a fault flag, properties and a message exchange pattern (e.g., *InOut* for request-reply), essentially has an *In* and *Out* message. Each processor in the pipeline processes the data in the *In* message and afterwards produces an *Out* message. If no *Out* message is given, the *In* message is copied and forwarded. The message is given by a message *body*, a set of message *header* entries, describing the data, and a set of *attachments* (e.g., binary data). During the processing, the messages are (fully) materialized in each processor. This can be improved by processing streams of single messages, if the integration adapters and message processors allow for streaming.

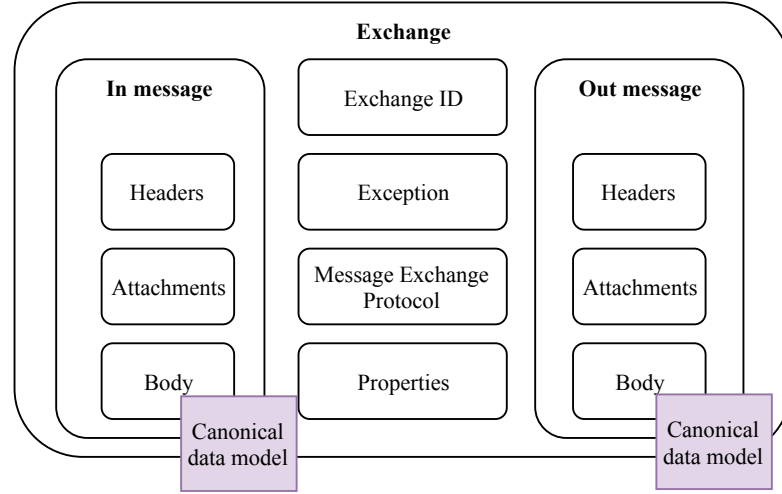


Figure 5.2: Apache Camel exchange (adapted from [IA10])

Example 5.1. The Camel exchange is created by a *Consumer Component*, which puts the data into the *In* message in the received format. Subsequent unmarshallers and type converters can be used to convert the data into a Canonical Data Model (CDM) pattern [HW04] (e.g., `JSONObject`), which can be evaluated subsequently by the processors. Thereby, the CDM denotes a lingua franca, which allows various message processor implementations (e.g., *Splitter*, *Aggregator*) to not work on arbitrary message formats, but to focus on classes of formats like XML, JSON or CSV, and thus solves the variety problem for integration systems. Once implemented for a format, conditions and expressions can be implemented in the respective languages like XPath, JSONPath. Figure 5.3 shows an example JSON message, accessible via user-defined functions (UDFs) and JSONPath. The resulting message can again be in JSON format. ■

Notably, since Apache Camel implements the original EIPs, a translation of IPTGs to Camel routes (incl. configuration) similar to our construction method Definition 3.44 (on page 119) should be possible. For compositional correctness checking using pattern contracts of IPCGs, an extension of Camel is required. However, due to our focus on processing efficiency and benchmarking, we leave such an extension as future work.

5.1.2 Vectorization: Micro-batching Integration Processing

We recall vectorization as a contemporary improvement of data processing. In particular, Gräfe [Gra90] classifies possible solutions as *vertical* (pipelining between processes) or *horizontal parallelization* (several CPUs process the same operation on subsets of the data like in MapReduce [DG08]). Previous work on *table-centric integration processing* (short *TIP*) [Rit15b] considers horizontal parallelization as a form of vertical parallelization. While it is implicitly assumed by the integration patterns that each pattern processes one message after another, the general idea behind the vectorization in TIP is that each message processor instance processes several messages of the same type at the same time in a SIMD-style (single instruction, multiple data). Therefore, the current approach to the Canonical Data Model (CDM) pattern is redefined as relational tables, which are constructed from the message either in the integration adapters or before the first message access (e.g., in the unmarshal or type conversion steps in Camel). Furthermore, several incoming messages are mapped to the table, such that each table entry denotes a single message.

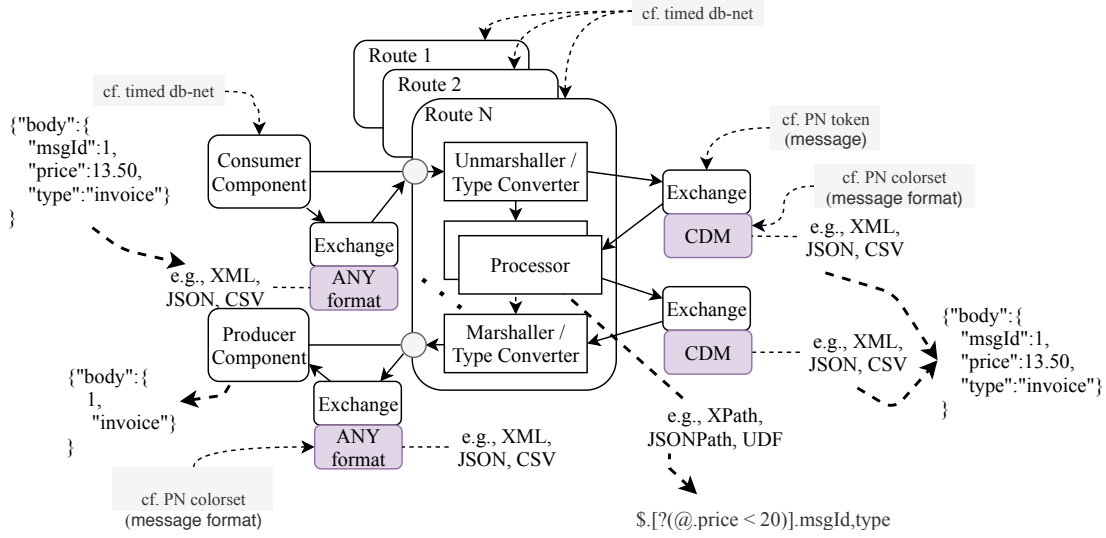


Figure 5.3: Message processing in Apache Camel by abstract example

Therefore, the TIP implementation proposed in [Rit15b] uses **Open-Next-Close (ONC)**-style *TableIterator* types as CDM and Datalog as a sufficiently expressive, recursive condition and expression language for access from the different message processing patterns. This micro-batching approach can be considered similar to the processing in Apache Spark [ZCF⁺10], which later developed a distributed collection of data organized into named columns, called **DataFrame** that is conceptually equivalent to a table in a relational database as well to the representation in the TIP approach. This was further taken up by Apache Flink as **Table API** [CKE⁺15] or the Apache Beam programming model for batching and streaming¹ as **PCollection**. However, the influence of TIP on these systems and languages remains unknown.

Example 5.2. The TIP implementation [Rit15b] leverages Apache Camel and only redefines the unmarshalling and the type conversion that constructs ONC-TableIterators from the incoming format. Figure 5.4 depicts an example of two incoming messages that are aggregated into one exchange and converted into a table of messages, later referred to as *Bulk-MSG* in Section 5.3. The message processors are configured with conditions and expressions in Datalog on these relations. While each processor within the route accesses the message using Datalog, the resulting output message can again be in JSON format (i.e., through type conversion and marshalling). ■

The subsequent introduction of TIP is taken from [Rit15b], if not stated otherwise.

Message Vectorization using Datalog

While the example gives an intuition of the approach, we subsequently define TIP for message routing and transformation patterns more formally. But before that, let us recall the encoding of some relevant, basic database operations / operators in Datalog, as used in [Rit15b]: **join**, **projection**, **union**, and **selection**. The join (e.g., inner or natural join) of two relations $r(x, y)$ and $s(y, z)$ on parameter y is encoded as $j(x, y, z) \leftarrow r(x, y), s(y, z)$, which projects all three parameters to the resulting predicate j . More explicitly, a projection on parameter x of

¹Apache Beam, visited 5/2019: <https://beam.apache.org/>.

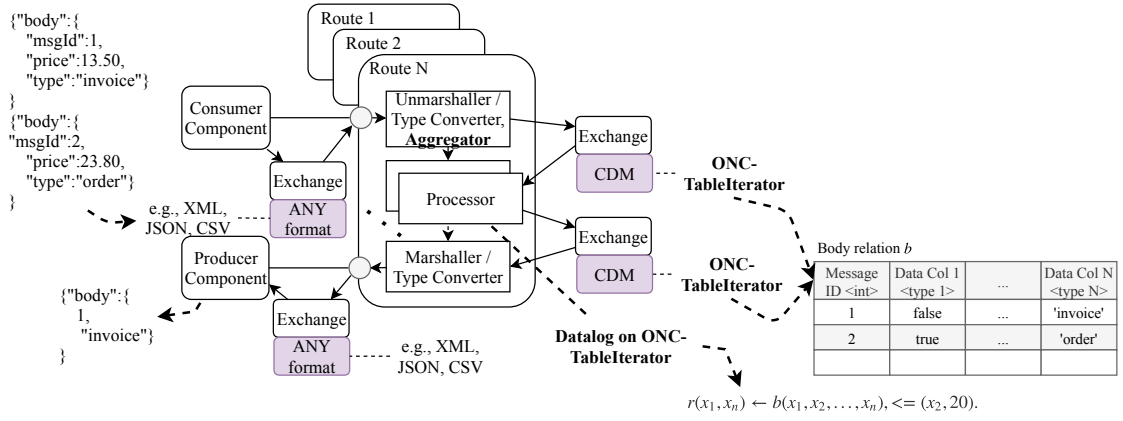


Figure 5.4: Vectorized processing with Apache Camel

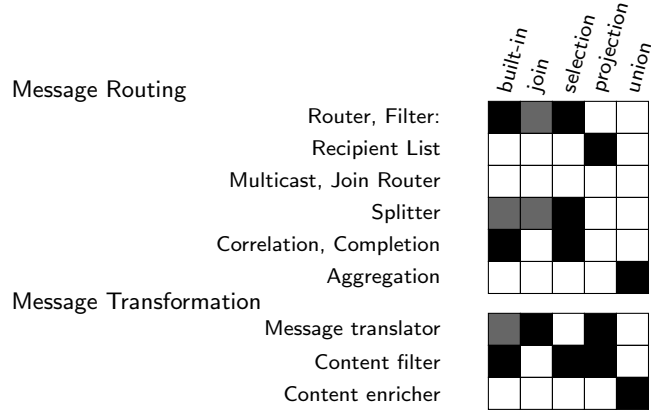


Figure 5.5: Message routing and transformation patterns mapped to Datalog. Most common Datalog operations for a single pattern are marked “black”, less common ones “light grey”, and possible but uncommon ones “white”.

relation $r(x, y)$ is encoded as $p(x) \leftarrow r(x, y)$. The union of $r(x, y)$ and $s(x, y)$ is $u(x, y) \leftarrow r(x, y) \cup s(x, y)$, which combines several relations to one. The selection from $r(x, y)$ according to a built-in predicate $\phi(x)$, where $\phi(x)$ can contain constants and free variables, is encoded as $s(x, y) \leftarrow r(x, y), \phi(x)$. Built-in predicates can be binary relations on numbers such as $<, \leq, =$, binary relations on strings such as *equals*, *contains*, *startswith* or predicates applied to expressions based on binary operators like $+, -, *, /$ (e.g., $x = p(y) + 1$), and operations on relations like $z = \max(p(x, y), x)$, $z = \min(p(x, y), x)$, which would assign the maximal or the minimal value x of a predicate p to a parameter z .

Although the TIP approach allows each single pattern definition to evaluate arbitrary, recursive Datalog operations and built-in predicates, the Datalog to pattern mapping tries to identify and focus on the most relevant table-centric operations for a specific pattern. An overview of the mapping of all discussed message routing and transformation operations to Datalog constructs is shown in Figure 5.5 and is subsequently discussed, by enumerating common integration patterns and separating system- from content-related parts for the TIP definition by example of Datalog.

Canonical Data Model

When connecting applications, various operations are executed on the transferred messages in a uniform way. The arriving message instances are converted into an internal format understood by the pattern implementation, called the CDM, before the messages are transformed to the target format. Hence, if a new application is added to the integration solution, only conversions between the CDM and the application format have to be created. Consequently, for a table-centric re-definition of integration patterns, we define a CDM similar to relational database tables as *Datalog programs*, which consists of a collection of facts / a table, optional (supporting) rules as message body and an optional set of meta-facts that describes the actual data as header. For instance, the data-part of an incoming message in JSON format is transformed to a collection of ONC-style table iterators, each representing a table row or fact. These ONC-operators are part of the evaluated execution plan for more efficient evaluation.

Message Routing Patterns

We first recall the routing patterns and then give their TIP re-definitions, which can be seen as control and data flow definitions of an integration channel pipeline. For that, the patterns access the message to route it within the integration system and eventually to its receiver(s). They influence the channel and message cardinality as well as the content of the message.

Content-based Router / Message Filter The most common routing patterns that determine the message’s route based on its body are the Content-based Router and the Message Filter. The stateless router has a channel cardinality of 1: n , where n is the number of leaving channels, while one channel enters the router, and a message cardinality of 1:1. The entering message constitutes the leaving message according to the evaluation of a *routing condition*. This condition is a function rc , with $\{bool_1, bool_2, \dots, bool_n\} := rc(msg_{in}, conds)$, where msg_{in} is the entering message. The function rc evaluates to a list of Boolean output $\{bool_1, bool_2, \dots, bool_n\}$ based on a list of conditions $conds$ of the same arity for each of the $n \in N$ leaving channels. In case several conditions evaluate to **true**, only the first matching channel receives the message.

Example 5.3. Listing 5.1 denotes an example of a routing condition on the TPC-H [TPC19] schema as a Datalog rule *cbr-order*. The rule matches all records with a certain *OPRIORITY* and *OTOTALPRICE* and then projects the order *ID* and *OTOTALPRICE* to the resulting set. For the Content-based Router this means that the corresponding message will be routed, if the result set is not empty.

Listing 5.1: Example routing condition

```

1      cbr-order(id, -, OTOTALPRICE, -):-
2      order(id, otype, -, OTOTALPRICE, -OPRIORITY, -) ,
3      =(OPRIORITY, "1-URGENT") , >(OTOTALPRICE, 100000.00) .

```

Note that multi-relational or multi-format messages can be processed in the same way. Listing 5.2 shows a corresponding example condition on TPC-H data, which defines a join over customer and nation tables.

Listing 5.2: Example routing condition with join over “multi-format” message

```

1      cbr-cust(CUSTKEY, -):-
2      customer(cid, ctype, CUSTKEY, -, CNATIONKEY, -, ACCTBAL, -) ,
3      nation(nid, ntype, NATIONKEY, -, NREGIONKEY, -) ,
4      >(ACCTBAL, 3000.0) , =(CNATIONKEY, NATIONKEY) , =(NREGIONKEY, 3) .

```

■

Through the separation of concerns (i.e., system implementation and user configuration), a system-level routing function provides the entering message msg_{in} to the content-level implementation (i.e., in CDM representation), which is configured by $conds$. Since standard Datalog rules are truth judgments, and hence do not directly produce Boolean values, we decided, for performance and generality considerations, to add an additional function $bool_{rc}$ to the integration system. The function $bool_{rc}$ converts the output list $fact$ of the routing function from a truth judgment to a Boolean by emitting **true** if $fact \neq \emptyset$, and **false** otherwise. Accordingly we define the TIP routing condition as $fact := rc_{tip}(msg_{in}, conds)$, while being evaluated for each channel condition (e.g., selection / built-in predicates). The integration system will then use the function $bool_{rc}$ to convert this into a Boolean value. For the message filter, which is a special case of the router that differs only from its channel cardinality of 1:1 and message cardinality of 1:[0]1], the filter condition is equal to rc_{tip} .

Multicast / Recipient List / Join Router The stateless Multicast and Recipient List patterns route multiple messages to several leaving channels, which gives them a message and channel cardinality of 1: n . While the multicast statically routes messages to the leaving channels (i.e., no re-definition required), the recipient list determines the channels dynamically. The receiver determination function rd , with $\{out_1, out_2, \dots, out_n\} := rd(msg_{in}, [header.y|body.x])$, computes $n \in \mathbb{N}$ receiver channel configurations $\{recv_1, recv_2, \dots, recv_n\}$ by extracting their key values either from an arbitrary message header field $header.y$ or from a message body field $body.x$. The integration system has to implement a receiver determination function that takes the list of key-strings $\{recvId_1, recvId_2, \dots, recvId_m\}$ as input, for which it looks up receiver configurations $recv_0, recv_1, \dots, recv_n$, where $m, n \in \mathbb{N}$ and $m > n$, and routes copies of the entering message $\{msg'_{out}, msg''_{out}, \dots, msg^{n'}_{out}\}$.

In terms of TIP, rd_{tip} is a projection of message body or header values to a unary, output relation. For instance, the receiver configuration keys $recvId_1$ and $recvId_2$ have to be part of the message body like $body(x, 'recvId_1').body(x, 'recvId_2')$. Then the rd_{tip} would evaluate a Datalog rule similar to $config(y) \leftarrow body(x, y)$, while the keys $recvId_1$ and $recvId_2$ correspond to receiver configurations $\{recv_1, recv_2\}$.

Splitter / Aggregator The antipodal Splitter and Aggregator patterns both have a channel cardinality of 1:1 and create new, leaving messages. The splitter breaks the entering message into multiple (smaller) messages (i.e., message cardinality of 1: n) and the aggregator combines multiple entering messages to one leaving message (i.e., message cardinality of n :1). Hereby, the stateless splitter uses a split condition sc on the content-level, with $\{out_1, out_2, \dots, out_n\} := sc(msg_{in}, conds)$, which accesses the entering message's body to determine a list of distinct body parts $\{out_1, out_2, \dots, out_n\}$, based on a list of conditions $conds$, that are each inserted to a list of individual, newly created, leaving messages $\{msg_{out1}, msg_{out2}, \dots, msg_{outn}\}$ with $n \in \mathbb{N}$ by a splitter function. The header and attachments are copied from the entering message to each leaving message.

The re-defined split condition sc_{tip} evaluates a set of Datalog rules as $conds$ (i.e., mostly selection, and sometimes built-in and join constructs). Each part of the body out_i with $i \in \mathbb{N}$ is a set of facts that is passed to a split function, which wraps each set into a single message.

The stateful aggregator defines a correlation condition, completion condition and an aggregation strategy. The correlation condition crc , with $coll_i := crc(msg_{in}, conds)$, determines the aggregate collection $coll_i$, to which the message is stored, based on a list of conditions $conds$. The completion condition cpc , with $cp_{out} := cpc(msg_{in}, [header.y|body.x])$, evaluates to a Boolean output cp_{out} based on header or body field information (similar to a message filter). If cp_{out} equals **true**, then the aggregation strategy as , with $agg_{out} := as(msg_{in}^1, msg_{in}^2, \dots, msg_{in}^n)$, is called by an

implementation of the messaging system and executed, else the current message is added to the collection $coll_i$. The aggregation strategy as evaluates the correlated entering messages $coll_i$ and emits a new message msg_{out} . For that, the messaging system has to implement an aggregation function that takes agg_{out} (i.e., the output of as) as input.

These functions are re-defined as crc_{tip} and cpc_{tip} such that the $conds$ are Datalog rules mainly with selection and built-in constructs. The cpc_{tip} makes use of the defined $bool_{rc}$ function to map its evaluation result (i.e., list of facts or empty) to the Boolean value $cpout$. The aggregation strategy as is re-defined as as_{tip} , which mainly uses **union** to combine lists of facts from different messages to one. The message format remains the same. To transform the aggregates' formats, a message translator is used to keep the patterns modular. However, the combination of the aggregation strategy with translation capabilities could lead to runtime optimizations.

Message Transformation Patterns

The transformation patterns exclusively target the content of the messages in terms of format conversations and content modifications.

The stateless Message Translator changes the structure or format of the entering message without generating a new one (i.e., channel, message cardinality 1:1). For that, the translator computes the transformed structure by evaluating a mapping program mt , with $msg_{out}.body := mt(msg_{in}.body)$. Thereby the field content can be altered.

Example 5.4. Listing 5.3 denotes an example of a simple transformation program on the TPC-H schema as a Datalog rule *conv-order*, which filters some of the order's fields.

Listing 5.3: Example message translation program

```

1      conv-order(id,otype,ORDERKEY,CUSTKEY,SHIPRIORITY):-
2      order(id,otype,ORDERKEY,CUSTKEY,-,SHIPRIORITY,-).
```

■

The related Content Filter and Content Enricher patterns can be subsumed by the general Content Modifier pattern and share the same characteristics as the translator pattern. The filter evaluates a filter function mt , which only filters out parts of the message structure (e.g., fields or values) and the enricher adds new fields or values as *data* to the existing content structure using an enricher program ep , with $msg_{out}.body := ep(msg_{in}.body, data)$.

The re-definition of the transformation function mt_{tip} for the message translator uses **join** and **projection** (plus **built-in** for numerical calculations and string operations, thus marked “light grey” in Figure 5.5) and **selection**, **projection** and **built-in** (mainly numerical expressions and character operations) for the content filter. While projections allow for static, structural filtering, the built-in and selection operators can be used to filter more dynamically based on the content. The resulting Datalog programs are passed as $msg_{out}.body$. In addition, the re-defined enricher program ep_{tip} uses **union** operations to add additional *data* to the message.

Pattern Composition

Since the TIP definitions target the content-level, all patterns can still be composed into more complex integration programs (i.e., integration scenarios or pipelines). From the many combinations of patterns, we briefly discuss two important structural patterns that are frequently used in integration scenarios: (1) scatter-gather and (2) splitter-gather. The scatter-gather pattern (with a 1:n:1 channel cardinality) is a multicast or recipient list that copies messages to several, statically or dynamically determined pipeline configurations, which each evaluate a sequence of

patterns on the messages in parallel. Through an Aggregator pattern, the messages are structurally and content-wise joined. The splitter/gather pattern (with a $1:n:1$ message cardinality) splits one message into multiple parts, which can be processed in parallel. In contrast to the scatter-gather the pattern sequence is the same for each instance. A subsequently configured Aggregator combines the messages to one.

The vectorization extension will be used for comparison with Apache Camel in [Section 5.4](#). Notably, the vectorization extension does not require to change the general pattern composition (cf. [Figure 5.1](#)), and thus a vectorization extension of IPCGs has no impact on their structure. The actual difference lies in the definition of the cardinality and partitioning of the exchanged data. This corresponds to changes in the data elements *EL* of the IPCGs and the correspondingly constructed colorsets in the timed db-nets with boundaries. In summary, IPCGs have a realization in timed db-nets and conceptually relate to current integration systems through their grounding on the integration patterns. However, existing implementations like Apache Camel were not build with IPCGs or timed db-net in mind (i.e., no concept of contracts or boundaries), and thus do not implement IPCGs.

5.2 Integration Scenario Analysis

We recall that due to the large number of integration patterns, for the definition of our benchmark, we focus on the most practically relevant patterns. This does not mean that implementations of other patterns cannot be added and benchmarked later on. To identify the most frequently used patterns, we conduct an analysis, in which we set the usage of integration patterns in real-world cloud integration scenarios into context to generally known integration types and styles. The analysis denotes a further sophistication of our analysis in [Chapter 2](#), and thus it is based on several cloud solutions, running in production on the SAP Cloud Platform Integration solution [[SAP19a](#)]. Moreover as a by-product, the analysis identifies pre-dominantly used message formats, which is relevant for our practical evaluation in [Section 5.4](#). More than 149 distinct integration scenarios with 934 common EIP usages out of 1429 were analyzed (w/o adapters). To derive the most relevant patterns of these scenarios, they are categorized according to their location in current enterprise architectures, their integration style and scenario type. [Figure 5.6](#) shows the scenario types (*ST*) which are relevant for the message exchange between applications, users and devices to chain business processes of current enterprise integration architectures. Similar to [[HW04](#)], we define an integration style according to its purpose of message exchange (e.g., invoking business functions, synchronizing data), and we distinguish six scenario types, *ST1*–*ST6*; each of which denotes the type of endpoints that participate in the exchange (e.g., cloud application or device). The scenario types can follow different integration styles. An integration scenario can be seen as specific description of one type and style, composed of diverse integration patterns.

5.2.1 Integration Scenario Styles

According to [[Lin00](#)] the classical *Application-to-Application* (*A2A*) integration styles are: *Process Invocation* (e.g., communicate creation or status updates of a business object) and *Data Movement* (i.e., synchronization and replication of a business object record). In particular, scenario type *ST1* “OP2OP” uses the integration style *Data Movement* which is typically realized using EIPs like *Message Translator* (MT). In [Tables 5.1](#) and [5.2](#) we summarize our analysis, and we also mention predominant message formats as well as example applications for each integration style. As scenario types may use the same EIPs and message formats for different applications, we discuss the EIPs and message formats below.

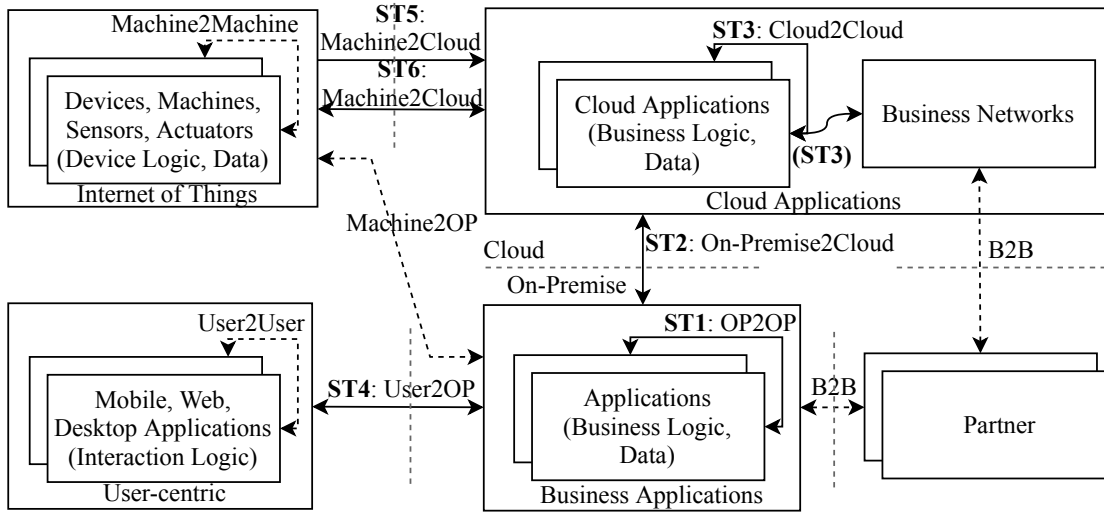


Figure 5.6: Overview of integration scenario types $ST1$ – $ST6$; dashed lines on arrows mark aspects that are out of scope

We continue the analysis of integration scenario types with $ST2$ “OP2C”, another application-to-application type. Unlike $ST1$ this integration scenario type focuses on the integration applications hosted in the cloud with on-premise applications. This type has become more prominent as applications are moving into the cloud, but they still need to be integrated with legacy on-premise applications. Furthermore, we identify $ST3$ “C2C” which deals with the integration of different cloud applications. As indicated in Tables 5.1 and 5.2 all three scenario types share the same integration styles: *Process Invocation* and *Data Movement*.

In addition, integration systems are often used in the area of *User-centric Application Integration* [Gme12] (e.g., display customer financial status) for the consumption of business data by users. We call this integration style *User-Centric Consumption*, and it maps to scenario type $ST4$ “User2OP”.

Furthermore, integrating physical devices with (business) applications becomes more important (e.g., medical [Zal09] or connected car device integration). Since the term “Device Integration” is still not consistently defined, we apply the classical styles process invocation and data movement to the devices and call them *Device Data Movement* for $ST5$ “M2C” and *Device Invocation* for $ST6$ “C2M”. Additionally, for scenario type $ST6$ we include a new scenario style, *Data Processing*, which is a combination of message processing and exchange as it is motivated by the related field of data analytics.

Figure 5.6 shows these six integration scenario types ($ST1$ – $ST6$) that we consider in this thesis. Although technically covered by other scenario types, the cases of cross-partner ($B2B$), *User-to-User* and *Machine-to-OP* message exchange are out of scope of this thesis, and thus depicted by “dashed-lines”.

5.2.2 Analysis of Real-World Applications

For every scenario type discussed above, we now analyze how real-world applications realize integration scenarios using certain integration patterns and message formats (also see Tables 5.1 and 5.2).

Table 5.1: Integration scenarios grouped by their integration styles and examples from SAP, Ariba and Success Factors (SFSF) applications: from CBR to AGG

Integration Style	Scenario Type	Msg. Format	Patterns					Example Applications
			CBR	MF	MC	SP	AGG	
Process Invocation, Data Movement	ST1: OP2OP	XML	✓	-	-	-	-	SAP ERP / CRM
	ST2: OP2C	XML, JSON	✓	✓	-	✓	-	SAP C4C, SAP S/4 HANA
	ST3: C2C	JSON, XML, (image, video)	✓	✓	-	✓	-	SFSF Employee Central, Ariba Quadrum Network, SAP S/4 HANA
User-centric consumption	ST4: User2OP, User2C	XML, JSON	-	-	✓	✓	✓	Hybris Social Marketing
Device Data Movement, Device Invocation, Data Processing	ST5: M2C	JSON, CSV	✓	-	-	-	-	Vehicle Logistics, Connected Cars
	ST6: C2M	JSON, CSV	✓	-	-	✓	✓	Sports Management, SAP Convergent Invoicing

Usual pattern occurrences are marked by ✓. Pattern abbreviations: Content-based Router (CBR) and Message Filter (MF) (\mapsto 11.35% of 934), Multicast (MC \mapsto 0.75%), Splitter (SP \mapsto 8.14%), Aggregator (AGG \mapsto 0.32%).

On-Premise Integration (OP2OP): The application-to-application message exchange between business applications (*ST1*) within one corporate network, referred to as *On-Premise* (OP), denotes the classical integration case (e.g., between SAP ERP and CRM solutions) with moderate message throughput requirements per integration scenario up to several 10,000 msgs/sec. The message formats are still mostly XML-based. In [Tables 5.1](#) and [5.2](#) we summarize the study of real-world scenarios from different integration styles in SAP Cloud Platform Integration [[SAP19a](#)], setting them into context to the used integration patterns. Accordingly, the classical *OP2OP* scenarios mostly use Content-based Router (CBR) and Message Translator (MT) patterns.

On-Premise-to-Cloud Integration (OP2C): Through the trend of building cloud applications or moving existing applications to cloud environments, there is a growing need for communication with on-premise applications (*ST2*). For instance, SAP ERP / CRM on Demand and SAP S/4 HANA applications require status changes of on-premise applications as well as data replication, while existing on-premise applications tend to delegate integration with governmental organizations and institutions, e.g., for legal aspects, to cloud environments. In addition to XML, JSON gains importance for those scenarios that reach peak throughput of up to several 100,000 msgs/sec, depending on the integration style. The patterns used in these scenarios (cf. [Table 5.1](#)) are again mostly MT, CBR, but also Message Filter (MF), Splitter (SP) and *User-defined Functions* (UDFs).

Cloud-to-Cloud Integration (C2C): The fast-growing field of *Cloud-to-Cloud* (Cloud2Cloud) integration (*ST3*; incl. micro-services [[New15](#)]), connects all kinds of business (e.g., Success Factors, Salesforce), social media (e.g., Twitter, Facebook), and business network applications (e.g., Ariba). Depending on the application domain, the message formats are mostly JSON-based, and the scenarios reach an even higher throughput, e.g., LinkedIn generates 100's of GB of new data in the form of one billion messages per day, Facebook generated 6 TB of user activity data

Table 5.2: Integration scenarios grouped by their integration styles and examples from SAP, Ariba and Success Factors (SFSF) applications: from MT to UDF

Integration Style	Scenario Type	Msg. Format	Patterns					Example Applications
			MT	CF	CE	CM	UDF	
Process Invocation, Data Movement	ST1: OP2OP	XML	✓	-	-	-	-	SAP ERP / CRM
	ST2: OP2C	XML, JSON	✓	-	-	✓	-	SAP C4C, SAP S/4 HANA
	ST3: C2C	JSON, XML, (image, video)	✓	-	✓	✓	✓	SFSF Employee Central, Ariba Quadrum Network, SAP S/4 HANA
User-centric consumption	ST4: User2OP, User2C	XML, JSON	✓	-	✓	-	-	Hybris Social Marketing
Device Data Movement Device Movement Invocation, Data Processing	ST5: M2C	JSON, CSV	✓	-	✓	✓	-	Vehicle Logistics, Connected Cars
	ST6: C2M	JSON, CSV	✓	✓	✓	✓	✓	Sports Management, SAP Convergent Invoicing

Usual pattern occurrences are marked by ✓. Pattern abbreviations: Message Translator (MT \mapsto 32.86%), Content Filter (CF \mapsto 2.99%), Content Enricher (CE) and Content Modifier (CM) (\mapsto 42.93%), user defined functions (UDF \mapsto 0.64%).

per day in 2012². Besides the previously discussed patterns, the most important integration patterns are Content Modifier (CM), Multicast (MC), e.g., for parallel message processing, and Content Enricher (CE), e.g., adding additional data to the message from a data store. Especially in cloud scenarios there are several auxiliary patterns like encoders or decoders, signer, verifier, decrypt or encrypt, which are mainly handled by integration adapters, e.g., WS-Security, thus out of scope for this work.

User-to-On-Premise (User2OP) and Cloud (User2C) Integration: The user-centric scenarios (ST4) are mostly about scheduled or ad-hoc, message-based queries that gather data from different data sources according to a user context and report back to the user. The queries are latency- and message throughput bound (e.g., usually less than two seconds). To meet these requirements, a combination of MC and CE patterns are used to gather data in parallel and enrich the response message. The message transformation pattern is required in case of different source and target formats.

Machine-to-On-Premise and Cloud (M2C) Integration: Recently, the case of device invocation, data processing (ST6), and data movement (ST5) has gained more importance. Scenarios like the convergent invoicing and vehicle logistics produce large amounts of messages, while requiring the Aggregator (AGG) pattern in addition to the previously discussed patterns to form common, map-reduce-like patterns, such as Scatter-Gather (i.e., MC, AGG) and Composite Message Processor (i.e., SP, AGG) [IA10].

General: The scenarios of all discussed integration styles potentially require reliable messaging with service quality guarantees [RH15], called message delivery semantics. The most common message delivery semantics are *At-least-Once* (i.e., ALO; *Message Redelivery* pattern [RH15]), *Exactly-Once* (i.e., EO; ALO with Idempotent Receiver pattern), and *EO-In-Order* (i.e., EOIO; EO with Resequencer pattern).

²Log processing metrics, relevant for message-based integration, visited 5/2019; last update 2012: <http://www.solacesystems.com/techblog/deconstructing-kafka>

5.2.3 Summary

The study shows the relevance of integration patterns along classical and new application integration styles and scenario types, and thus allows for the identification of the most relevant patterns. From the currently known routing and transformation patterns, the studied cloud integration scenarios mainly use the Content-based Router, Message Translator, Splitter, Content Modifier, and Content Enricher. Consequently, these patterns are considered as relevant for EIPBench. In addition to these patterns, new integration scenarios require even more data-aware operations (e.g., CM, CE) and patterns for parallel, map-reduce-style processing (e.g., MC, SP, AGG). Furthermore, we include other branching patterns (i.e., Multicast and Recipient List) for variety and leave out the content modifier, due to its similarity to the message translator. Finally, we include all patterns from the message delivery semantics discussion. In addition, many scenarios use User-defined Functions (UDFs) which indicates that current patterns do not satisfy all requirements.

Especially new domain-operations (e.g., arithmetic operations) and the flexibility required for more diverse message formats and new scenarios introduce new challenges. Notably, less verbose message formats like JSON are used, which influences our message format selection in EIPBench.

5.3 Integration Pattern Benchmark

We subsequently define an integration pattern benchmark based on the benchmark principles by Gray [Gra93] and the integration scenario analysis, called *EIPBench*.

5.3.1 Benchmark Design

First we discuss general EIPBench design choices, for the message format and macroscale factor criteria (incl. concurrent users). We use configurable scale factors for the EIP benchmark definitions to allow the specification of a data-aware message processing benchmark. Subsequently, the message generation and general, macroscale factors are discussed. The EIP microscale factors are discussed in the next section.

Data Set and Message Creation

An important aspect of EAI systems is the message format (e.g., [Lin00, HW04]). The analysis of scenario types in Tables 5.1 and 5.2 (on page 180) indicates that mostly textual message formats are used (e.g., XML, JSON, CSV), while binary data (e.g., images, videos) is currently limited to a few social media applications (cf. Cloud2Cloud). For the textual formats, there seems to be a move from XML to JSON (and CSV) formats. Hence, we define the message body format as textual, JSON and specify the integration pattern content accordingly.

Data Set The messages can have an arbitrary format, however, current business application data and even social media data look similar to existing TPC data sets. Hence, we decided to start with a standard, PDGF-generated [PRFD11] TPC-H data set that provides different scale levels and — similar to *BigBench* [RDF⁺15, RJ13] — extend the generation for our purpose. The TPC-H data describes business object formats, which can be found within exchanged messages (i.e., resulting to less conversions). Although the generated data sets cannot be directly used as messages for the benchmark, they provide basic business objects such as `ORDERS`, `CUSTOMER` and do not require further explanation in the benchmark community. The TPC-H scale-level one generates 1.5 million `ORDERS`, 150k `CUSTOMER`, 25 `NATION` and 5 `REGION` records as CSV files.

Message Models The message models are generated from the data sets using the following operations: join, union, append (\oplus), and scale. Following the edges, Figure 5.7 shows from left to right how the generated TPC-H source relations are combined to message formats. Subsequently

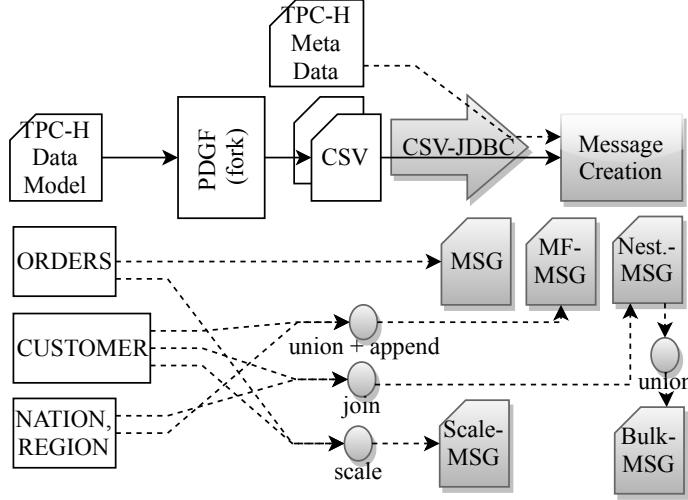


Figure 5.7: Extended PDGF-based message creation

a message MSG is defined as $MSG := (B, H, A)$, with an arbitrary message content or body B , an optional list of name-value pairs denoting the message header H , describing the content, and a list of name-binary value pairs for the attachments A (optional). Usually the format of B is typed to one message model (e.g., TPC-H **ORDERS**). For our (source) message model MM we focus on the TPC-H order to customer processing. We select the foreign key related relations **ORDERS**, **CUSTOMER**, **NATION** and **REGION** in the CSV message protocol, transform the single records to JSON and add two additional columns: a unique message identifier and type information that specifies the name of the source relation. The single JSON objects are combined to one JSON array and stored as a source model for the benchmark execution. To sufficiently support data-aware scenarios, the focus of EIPBench lies on the message body and not the header. The source order messages are defined as $MSG_{Ord.B} := \{msgId, type\} \oplus OBJ_{Ord}.fields$, while $MSG_{Ord}.[H|A] := \emptyset$, where $OBJ_{Ord}.fields$ are the fields of the order object. Analogously, customer MSG_{Cust} , nation MSG_{Nat} , and region MSG_{Reg} are defined.

While the first tranche of messages results in a message body B with a single message model, some scenarios require *Multi-format* (MF) messages (e.g., convergent invoicing requires additional information added to the message in a different format). A MF message (MSG_{MF}) is defined as a list of potentially different message models MM . For EIPBench MSG_{MF}^{CNR} messages with one **CUSTOMER** record and all **NATION** and **REGION** records are created. Hence, the source messages are defined as $MSG_{MF}^{CNR} := \{msgId, type\} \oplus OBJ_{Cust}.fields \oplus OBJ_{Nat}.fields \oplus OBJ_{Reg}.fields$. Multi-format messages transport additional, joinable information as message content, while cyclic dependencies are allowed.

In addition, tree-like messages play a role, e.g., for OP2Cloud scenarios. Hereby the foreign key relations between **CUSTOMER** and **NATION** relation are replaced within the customer record beforehand, leading to nested message structures N . For instance, SAP Intermediate Document (IDoc) Types allow the definition of segments, which are a parent-child-like structure³. The nested

³SAP IDoc structure, visited 5/2019: <http://scn.sap.com/docs/DOC-34785>

source messages N of customer and nation objects are defined as $MSG_{Nest}^{CN}(n) := \{msgId, type\} \oplus_1^n (OBJ_{Cust}.fields \setminus C_NATIONKEY) \oplus (OBJ_{Nat}.fields \setminus N_NATIONKEY)$, where n defines the number of nested customer entries as message content.

To support a message scaling over orders with a list of nested customer records $MSG_{Nest}^C(n)$, we define $MSG_{scale}^{OC} := MSG_{Nest}^{OC}(m)$, with $m > 1$. Messages of several hundred MB, e.g., as required for Financial Services Network Cloud-OP messaging, not only help to test the message throughput but also the capability of an integration system to handle bigger data volumes (i.e., not only “fast”, but “big” data).

Each single message model can be stashed into a message collection $Col_\lambda(MSG)$, where $MSG := \{MSG_{Ord}, MSG_{MF}, MSG_N, MSG_{scale}\}$ with collection size λ , which specifies the number of messages within the collection.

Macroscale Factors

The integration patterns need to scale along different dimensions. Consequently, we define the following *macroscale factors*: (i) messages with different user contexts (i.e., number of concurrent users), (ii) micro-batching, and (iii) message size (implicit and explicit).

The scale factor *concurrent users* (i) tests the ability of pattern implementations to handle concurrent requests. The generic, concurrent user load pattern for a particular scale level can be freely configured and is defined as:

$$scale_{cu}(\omega) = 2^\omega \quad (5.1)$$

For example, when transferring the settings of the ESB Performance benchmark [Adr13], ω varies between 0 and 11. In our experiments, we use $0 \leq \omega \leq 6$, which already sufficiently shows the impact of this scale factor to answer question Q5.

Furthermore, we define the *micro-batching* factor (ii). With this parameter we intend to show the benefit of batched processing for the message throughput in integrations systems (cf. Q6). In this context, the data-aware processing approach is a newly developed mechanism that allows to send collections of messages ($Col_\lambda(MSG)$) instead of single messages [Rit15b], called *micro-batching*. Currently only the patterns discussed in [Rit15b] such as message transformation are micro-batch enabled. The batch scale levels β , with $0 \leq \beta$, denote the number of distinct messages in one message collection $Col_\lambda(MSG)$:

$$\lambda := scale_{batch}(\beta) = 2^\beta \quad (5.2)$$

The ESB Performance benchmark [Adr13] does not specify such a test. In EIPBench β is configurable, and we choose β with $0 \leq \beta \leq 10$ to show the general impact of micro-batching.

Especially in cloud-to-cloud integration scenarios and also business network solutions we observe that various *message sizes* (iii) are used. EIPBench addresses this challenge by constructing larger message sizes (cf. Q5) of multi-format MSG_{MF} and nested $MSG_{Nest}(n)$ messages, as they are used by various applications. For a given message of type θ we define a function $size([msg|obj]_\theta)$, which determines the message size in kB. For instance, the size of MSG_{Ord} is approximately 0,354 kB and for the nested customer object $size(OBJ_{cust}) \approx 0,293$ kB. For the size of nested messages, the type of the nested business object obj_τ can be specified. The extended function $size(msg, \theta, \tau)$ calculates the size inclusive the nested object. Since the nesting is calculated by a foreign key fk relation between the business objects, a function $size([msg|obj]_\theta, fk)$ returns the size of a message or object without the foreign key field. The

generic size calculation of objects and messages is defined as:

$$\begin{aligned}
size(MSG'_\theta) &= size(MSG_\theta, fk) \\
&= size(MSG_\theta) - size(fk) \\
size(OBJ'_\tau) &= size(OBJ_\tau, fk) \\
&= size(OBJ_\tau) - size(fk)
\end{aligned} \tag{5.3}$$

Now, the size of these messages is scaled through parameter η , with $1 \leq \eta \leq 20$. For example, for $\eta = 20$ we generate messages of approximately 512 MB in size. In comparison, the ESB Performance benchmark [Adr13] specifies messages up to 100 kB. In addition to the generic scale factor η , there is another message size factor γ , which helps to increase the number of business objects of a scale level: Equation (5.4) brings all previous pieces together and shows the generic calculation of the message size of a scaled message MSG_{scale} for a particular scale level η .

$$MSG_{scale}^\eta = size(MSG'_\theta) + \eta \cdot \gamma \cdot size(OBJ'_\tau) \tag{5.4}$$

For instance, the concrete scale factor constant in EIPBench is $\gamma = 6$. For the nested messages $MSG_{Nest}(n)$, n is defined as $n := \gamma \cdot \eta$. Concrete values for MSG_{scale}^η , e.g., for simple customer messages in EIPBench with $\theta, \tau := Cust$ range between approximately 256 B (for $\eta = 0$) up to 256 MB and 512 MB.

Summary

Based on the scenario analysis and classification, we identified appropriate message formats and macroscale factors for EIPBench. Considering the focus on data-aware pattern processing the definitions allow for the specification of a comprehensive benchmark. Subsequently the tested patterns are introduced and defined by their microscale factors.

5.3.2 Pattern Design Choices

In this section, we define microscale factors for the patterns to be tested based on the categories of message routing, transformation patterns, and message delivery semantics from the integration scenario analysis. Each pattern represents an operation on one or multiple of the defined message models and defines its own microscale factors. The microscale factors describe and test the complete characteristics of the patterns. Subsequently, the scale levels and variations for the different benchmarks are enumerated alphabetically, where A usually denotes the normal or simple case and the cases B, C, \dots represent (scale) variants.

Message Routing Patterns

The message routing patterns decouple the message sender from its receiver(s). We focus on content-based routing capabilities (i.e., no header), which are mainly used in practice and especially relevant for the evaluation of data-aware processing. Table 5.3 lists the relevant routing patterns (RT), which are subsequently discussed.

Content-based Router, Message Filter ($RT-1, RT-2$) The Content-based Router (CBR) routes one incoming message to exactly one of the n outgoing channels according to the ordered evaluation of $n-1$ channel conditions that read the message's content. The first condition that evaluates to **true** decides on the outgoing channel, if no condition evaluates to **true** the message is routed to the n th channel (the default channel). The Message Filter (MF) is a special case of

Table 5.3: Message Routing (RT) patterns with microscale factors

Label	Patterns	Description	Scale
RT-1	Content-based Router (CBR), Message Filter (MF)	Channel cardinality $1:[1 n]$, $n \in \mathbb{N}$ outgoing channels, $m \in \mathbb{N}$ (dis-) conjunctive conditions w/ increasing complexity	A : normal, B : $n > 1$, C : $m > 2$
RT-2	Multi-format CBR, MF	same as <i>RT-1</i> on multi-format message with $k \in \mathbb{N}$ entries	D : $k > 1$
RT-3	Multicast (MC)	Channel cardinality $1:n$, $n \in \mathbb{N}$ outgoing channels, parallel processing, stop on exception	A : $n = 1$, B : $n > 1$, and variations
RT-4	Recipient List (RL)	same as <i>RT-3</i> with n receiver determinations	A : $n = 1$, B : $n > 1$, and variations
RT-5	Splitter (SP)	message cardinality $1:i$, $i \in \mathbb{N}$ outgoing messages, parallel processing, stop on exception	$i > 1$, split cardinality, variations
RT-6	Aggregator (AGG)	message cardinality $i:1$, $i \in \mathbb{N}$ incoming messages	completion sizes, aggr. strategies

the CBR with a channel cardinality of 1:1, resulting in a “pass or no-pass” decision.

Example: Using different processing for an order with higher **ORDERPRIORITY** and higher price **TOTALPRICE** with the CBR or filter out messages with **ORDERSTATUS** of “F”.

Scale / Variations: Scaling through number of complex conditions and number of branches.

Implementation: EIPBench evaluates the different micro- and macroscale factors, conditions on MSG_{Ord} for (A–C) and MSG_{Nest}^{CN} for (D).

Multicast, Recipient List (RT-3, RT-4) The Multicast (MC) describes the statically configured serial or parallel sending of n copies of the same message to n receivers, while the Recipient List (RL) dynamically computes the receivers from the original message through a receiver determination function. Technically, both patterns create message channels (i.e., threads) for each outgoing message.

Example: Copying one order message to several (parallel) channels statically for further processing with a MC, or selecting or calculating the message channel from the body of the message with the RL (e.g., orders with different priorities).

Scale / Variations: Scaling through branches (tests branching), parallel branching vs. sequential processing; on exception.

Implementation: EIPBench scales multiple outbound message channels for the MC and configures RL to use one outbound route per order priority on MSG_{Ord} , which tests the channel branching behavior (i.e., channel creation).

Splitter (RT-5) The Splitter (SP) splits an incoming message (with repeated elements) into i outgoing messages to the same receiver using a split condition.

Example: The creation of new messages for each part of a multi-format message MSG_{MF} (incl. foreign key creation) or the separation of a message collection $Col_i(MSG_{Ord})$ into single messages per entry.

Scale / Variations: Scaling through increasing split cardinalities i .

Implementation: EIPBench configures the splitter to (A) split collections of messages MSG_{Ord} into single messages (the reverse of an Aggregator for micro-batching), (B) split each entry or section of a message into a single message, (C) take first four fields of a message (message id, type, orderkey, custkey) as fixed parts, then split the next j elements into j messages and add last one element (comment) to the message as a footer.

Table 5.4: Message Transformation (MT) patterns with microscale factors

Label	Patterns	Description	Scale
MT-1	MT	program with $n:m$ distinct field mappings, each with a directed operator tree of size i	(A) $n, m \leq 10, i = 1$
MT-2	CF	filter o fields	$o \geq 1$
MT-3	CE	enrich message with j new fields or complete structures	$j > 0$; nesting and multi-format variants.

Aggregator (RT-6) The Aggregator (AGG) combines j incoming message into one outgoing message, sent to the same receiver using correlation and completion conditions (e.g., size, time) and an aggregation strategy.

Example: Constructing the union of two relations as one multi-format message MSG_{MF} or the stashing of messages as a message collection.

Scale / Variations: Scaling through increasing completion size or time.

Implementation: EIPBench configures the aggregator to merge different numbers j of order messages MSG_{Ord} .

5.3.3 Message Transformation Patterns

The message transformation patterns cover an important aspect of integration systems that contain the translation of one format into another one (Message Translator (MT) [HW04]), the enrichment of additional information in a message (Content Enricher (CE)) and the filtering of content (Content Filter (CF)). In more practical realizations, the Message Mapper is used to convert from the message’s format to a Canonical Data Model. In addition, new patterns were found for executing arbitrary scripts on the message (Script pattern) and for the more guided modification of the content using expression editors in form of the Content Modifier (CM) pattern in Chapter 2.

In this chapter, we focus on the standard MT, and (transient, internal) CE and CF patterns (MT-1-3) as shown in Table 5.4. For all of these patterns the channel- and message cardinalities are 1:1, i.e., they are non-message generating, and we consider the stateless cases here.

Message Translator (MT-1) Message translators (MT) transform the structure and values of an incoming message. The mapping program has $n:m$ distinct field mappings, where the incoming message has n and the outgoing message m fields. Each field mapping can be expressed with a directed operator tree of size i . For instance, $i = 1$ means that one operation is used to transform one field into another one. The single operations intersect with some of the information integration queries defined in [HST05] (e.g., Query-2 “Mathematical operations”, Query-3 “String contains”). According to the study of [SAP19a], MT for integration programs is more complex and can be summarized as arbitrary combinations of the following n -ary operations:

- value assignments / mappings: e.g., default values, constants, copy.
- type-specific operations: e.g., string concatenation, numeric subtraction, addition.
- conditions: e.g., equals, greater than, contains.
- external scripts/functions: e.g., value mapping lookups, user-defined functions, external service calls.

Example: Transforming the mandatory ORDERKEY field to one or many fields of the target structure. For instance, a mapping from source field A to target field B checks that the ORDERKEY

is not null, has a certain length and only then assigns its value: $\text{checkNotNull}(A) \rightarrow \text{checkLength}(A) \rightarrow \text{assign}(A, B)$.

Scale / Variations: Variations of increasing numbers of n , m and the size of i , as well as the complexity of the operations (e.g., iterative calculations).

Implementation: Since the operator-tree processing of the MT is similar to complex conditions that are already checked in the benchmark, only a simple mapping program MT-1 (A) is benchmarked in EIPBench.

Content Filter (MT-2) Content filters (CF) remove o fields and values from a message.

Example: The message receiver only requires `ORDERKEY`, `CUSTKEY` and `ORDERPRICE`, and all other fields and values are removed.

Scale / Variations: Scaling through an increasing number of filtered fields o ; and adding more complex filter conditions.

Implementation: EIPBench filters `ORDER` message fields.

Content Enricher (MT-3) Content enrichers (CE) add j new fields and values to a simple message, or build a nested or multi-format message structure.

Example: The message requires additional master data for the credit check of a customer, which is added to the current message.

Scale / Variations: Scaling through an increasing number of added fields j ; and building more complex structures.

Implementation: EIPBench focuses on the “in-memory” enrichment of message content (i.e., leaves out external calls).

5.3.4 Message Delivery Semantics

For reliable messaging, integration scenarios require different levels of message delivery semantics: Best Effort (BE), At-Least-Once (ALO), Exactly-Once (EO), and Exactly-Once-in-Order (EOIO) [RH15], which can be composed through the standard Idempotent Receiver (IR) and Resequencer (RS) patterns, and the Message Redelivery on Exception (MRoE) pattern from [RS14].

The discussed benchmarks in Section 5.5 assume no reliability, which either means message redelivery on exception by the applications, devices in case of synchronous messaging, or message-loss after an unforeseen event during asynchronous best effort delivery. If the message shall be delivered at-least-once, a message redelivery on exception pattern is required, which might lead to duplicate messages exchange. To avoid that exactly-once combines at-least-once with an idempotent receiver pattern, which filters out duplicate messages. When a special sequence of messages shall be preserved (e.g., create before update operation), then exactly-once is combined with a resequencer pattern, or a commutative receiver is used. The microscale value domains are configurable. However, for our experiments with EIPBench they are set to values, which show the general impact on message processing. Table 5.5 lists the relevant message delivery semantics, which are subsequently discussed.

Message Redelivery on Exception (MDS-1) Redeliver messages on exception (transient) to receiver o times.

Example: The creation of an order fails due to a temporary network outage and will be immediately re-delivered to make sure that the order will reach its destination as soon as the issue is solved.

Scale / Variations: increase number of redeliveries o ; send original or modified message.

Table 5.5: Message Delivery Semantics (MDS) with microscale factors

Label	Patterns	Description	Scale
MDS-1	MRoE	redeliver message on failure $o \in \mathbb{N}$ times, (non-) original message	A: $o = 1$, B-F: $32 \geq o > 1$; variants
MDS-2	RS	sequence of $n \in \mathbb{N}$ messages, sequence identifier	$n \geq 1$; A: $n = 10$
MDS-3	IP	filter duplicate messages $m \in \mathbb{N}$ “in-memory”	A: $m = 0$, B: $100,000 \geq m > 0$

Implementation: EIPBench configures the MRoE pattern with (A) no redelivery, (B-F) with $o := \{1, 2, 4, 8, 16, 32\}$ on MSG_{Ord} messages.

Resequencer (MDS-2) Receive sequence of messages (transitively), correlate using a sequence identifier and re-order, when the sequence is complete.

Example: The creation of a customer has to happen before the update of the same customer (i.e., an enforced sequence of operations) or before the creation of a referenced order (i.e., an enforced sequence of business object creation).

Scale / Variations: Scaling through number of entries per sequence

Implementation: EIPBench varies the number of sequence entries n , with $n \in \{10, 100, \dots, 10^5\}$; implemented for (A) $n = 10$ and resequencing messages according to their **TOTALPRICE** on MSG_{Ord} messages.

Idempotent Receiver (MDS-3) Filter duplicate messages using (transient) memory.

Example: The message source sends the same order twice for creation in another application (same **ORDERKEY**).

Scale / Variations: Scaling through increasing number of duplicates leads to more main memory consumption due to transient and more frequent lookups or scans.

Implementation: EIPBench configures the IR for (A) no duplicates, (B) duplicates after 100,000 checked for $msgId$ on MSG_{Ord} messages.

5.3.5 Benchmark Implementation

The EIPBench is executed close to the pattern implementations, potentially even within the same process. Our reference implementation uses JMH⁴, a Java harness for running benchmarks on the JVM, which factors out JVM side-effects (e.g., on stack replacement) through code generation and allows to configure warmups, iterations and the number of isolated JVM instances. As illustrated in Figure 5.8, the benchmark realization is divided into three phases: initialization (**pre**), execution (**work**), and verification (**post**). We provide a tool suite that contains:

Initializer: generates the data and creates the messages in the preparation (**pre**) phase.

Client: selects the benchmarks in the preparation (**pre**) phase and uses JMH to schedule the execution of message producers for the different integration scenarios in the **work** phase.

Monitor: collects the statistics, calculates performance metrics and plots the results in the post-processing (**post**) phase (not shown).

⁴JMH, visited 5/2019: <http://openjdk.java.net/projects/code-tools/jmh/>.

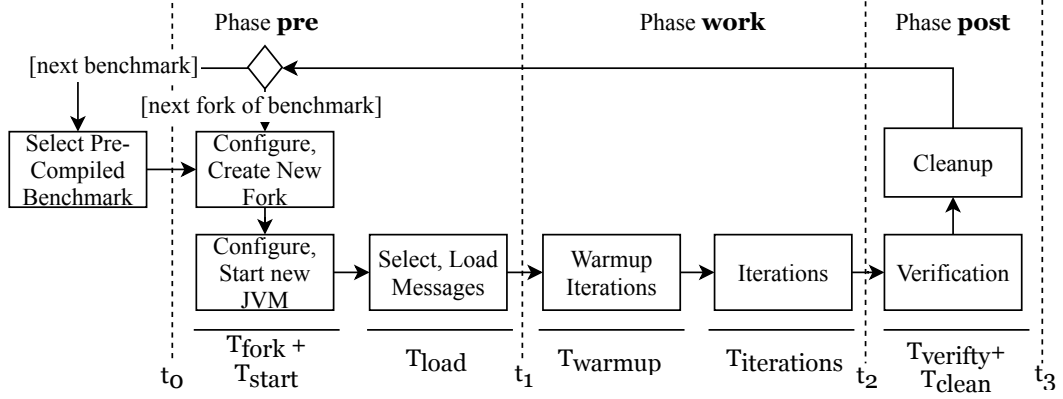


Figure 5.8: EIPBench execution phases

The time spent during the **pre** phase T_{pre} consists of the time required for the creation of a fork T_{fork} , the loading time of all messages required by the current benchmark T_{load} , and the preparation time for the start of the benchmark T_{start} :

$$\begin{aligned} T_{pre} &= T_{fork} + T_{start} + T_{load} \\ &= t_1 - t_0. \end{aligned} \quad (5.5)$$

During the **work** phase, the client executes the defined pattern benchmarks on a specified number ζ of isolated and freshly initialized JVM instances, called forks, for a configurable amount of m warmup and n main iterations. The execution time of this phase t_{work} mainly adds up the warmup T_{warmup} and the actual evaluation time T_{eval} :

$$\begin{aligned} T_{work} &= T_{warmup}(\varphi) + T_{eval}(\varphi) \\ &= m \cdot eval(\varphi) + n \cdot eval(\varphi) \\ &= t_2 - t_1. \end{aligned} \quad (5.6)$$

During the evaluation, the selected benchmark is executed, and the discrete throughput values φ are collected. Each fork accesses the created message files (T_{load}) and sends (collections of) messages to the message channel with the tested patterns. Hence the overall runtime of the whole benchmark is $T_{Bench} = \zeta \cdot (T_{pre} + T_{work} + T_{post})$. To measure T_{work} , the message scenarios are synchronous and have a **VOID** receiver adapter, which immediately returns to the sender. Then, cleanup and verification are performed:

$$\begin{aligned} T_{post} &= T_{clean} + T_{verify} \\ &= t_3 - t_2. \end{aligned} \quad (5.7)$$

When a complete scale factor run is finished, the results are serialized to disk in a **raw** format, containing all captured measurements. The monitor parses the data and creates plots for all tested patterns and scale factors.

The relevant metrics for EIPBench is the discrete throughput measures φ of a tested pattern (i.e., T_{eval}). More precisely, T_{eval} is the calculated mean of the individual evaluations $eval(\varphi_i)$,

with $i \in I$, for the number of iterations I within one fork:

$$T_{eval} = T_{mean/fork} = \frac{\sum_{i=1}^n eval(\varphi_i)}{n}. \quad (5.8)$$

For reproducible results the whole test instance will be cleared after one fork and initialized. The benchmark will be executed for the number of forks ζ . Equation (5.9) shows the calculation of the mean for multiple forks. While higher number of forks (i.e., $\gg 10$) leads to increasing overall execution times, the results become more reproducible.

$$T_{mean} = \frac{\sum_{j=1}^{\zeta} T_{mean/fork}(j)}{\zeta} \quad (5.9)$$

$$T_{\sigma} = \sqrt{\frac{\sum_{i=1}^{\zeta} (T_{eval}^i - T_{mean})^2}{\zeta}}.$$

In addition to the mean, EIPBench measures a confidence value for the result with a confidence level α of 99% (i.e., confidence interval ci). The confidence interval is calculated once for all forks based on the observed mean throughput values and the standard deviation. Equation (5.10) shows the upper and lower bound calculation of T_{ci} .

$$T_{ci} = \begin{cases} T_{mean} - \alpha \cdot \frac{T_{\sigma}}{\sqrt{\zeta}}, & \text{lower.} \\ T_{mean} + \alpha \cdot \frac{T_{\sigma}}{\sqrt{\zeta}}, & \text{upper.} \end{cases} \quad (5.10)$$

Subsequently, the T_{ci} values will be shown as error bars for macroscale plots.

5.4 Evaluation

Now, we briefly describe the implementation and setup of the benchmark and share the results of running the benchmark to answer our guiding questions and discuss lessons learned, e.g., including “deficits” found in the pattern implementations.

5.4.1 Benchmark Setup

All measurements are conducted on a HP Z600 workstation, equipped with two Intel X5650 processors clocked at 2.67GHz with a 12 cores, 24GB of main memory, running a 64-bit Windows 7 SP1 and a JDK version 1.7.0 with 2GB heap space.

For our experiments we used the test harness described in Section 5.3.5. As first system under test, we decided to use the open-source integration system Apache Camel v2.17 [IA10] implemented in Java, referred to as Java/AC, since it provides implementations for all discussed patterns and is used in SAP Cloud Platform Integration [SAP19a]. For comparison we have chosen a Java-based, data-aware integration pattern implementation introduced in Section 5.1.2, which simulates table operations on the message content, unmarshalled to `ONC`-iterators instead of JSON objects during T_{Load} . For the table operations we use the Datalog reasoner from [RW12] implemented in Java. Since the reasoner runs embedded in Apache Camel [Rit15b], we subsequently refer to it as TIP/AC and in some cases simply Datalog.

5.4.2 Benchmark Results

For the discussion of the benchmark results, we follow the research questions RQ2-3(a)–(c). We show representative results, instead of discussing each particular result. Subsequently all diagrams

Table 5.6: Message throughput (number of messages) of Content-based Routing (RT) and Message Transformation (MT) pattern benchmarks compared to the Baseline (BL)

Benchmarks	Scale	Java/AC (early-out)	Java/AC	TIP/AC
BL	–	n/a	300,837 +/-8,252	n/a
RT-1	A (simple)	174,795 +/-8,100	176,319 +/-4,704	179,528 +/-5,485
	B (branching)	158,838 +/-3,002	100,070 +/-2,635	163,672 +/-4,186
	C (complex)	115,599 +/-3,901	98,237 +/-2,261	115,859 +/-3,417
	D (join)	165,644 +/-3,132	–	176,926 +/-6,513
MT-1	A (medium)	n/a	172,545 +/-7,612	193,378 +/-4,407

Throughput denoted by T_{mean} (cf. Equation (5.9)) and T_{ci} (cf. Equation (5.10)).

show message throughput for different scale levels. Discrete points are calculated mean values T_{mean} (cf. Equation (5.9)) according to the metrics, and the error bars denote the precision of the values according to the 99.9% confidence interval T_{ci} (cf. Equation (5.10)); i.e., small intervals indicate low variance, thus a higher confidence).

Before benchmarking the different patterns, we conducted a “baseline” benchmark using Java/AC without any pattern configurations, which measures the pipeline processing without operations on the message (cf. *BL* in Table 5.6).

Microscaling

To answer the “microscale” question *RQ2 – 3(a)* about the impact of complex routing conditions and multiple branchings, we benchmarked the routing test description *RT-1* (i.e., content-based routing) together with streams of *MSG_{Ord}* messages for the Java/AC and TIP/AC implementations. Conceptually the routing conditions are similar to the examples for the patterns in Section 5.3.2.

On the impact of complex routing conditions and multiple route branchings (RQ3-2(a)): Table 5.6 shows the results of *RT-1* starting with the simple routing condition case *RT-1 (A)*, followed by increased route branchings *RT-1 (B)*, condition complexity *RT-1 (C)*, and complex conditions on multi-format messages. Not surprisingly, the materialization of messages for processing by a pattern implementation results in a significant decrease in the throughput compared to the baseline measurement (cf. *BL*). The number of route branchings in *RT-1 (B)* correlates with the number of evaluated conditions (worst case). In our experiments, all conditions are executed. The impact of an increasing branching factor on the throughput can be considerable. An even stronger impact on the throughput comes from more complex routing conditions in *RT-1 (C)*. Hence, as answer to question *RQ3-2(a)*, the results show a significant impact of multiple branchings and complex routing conditions suggests that selectivity estimations on the conditions and re-orderings similar to DB queries should be further investigated (cf. [DAF⁺03]). Particularly, for the TIP/AC implementations, the parallel evaluation of routing conditions could result in performance improvements, however, that would probably require a change of the pattern semantics (cf. [Rit15b]).

Further message routing impact factors: During the implementation of the benchmark, the “early-out” capability of implementations (i.e., filter can return halfway during the scanning (for row filter) [Geo11]) turned out to be another important factor of routing throughput. The Java/AC “early-out” implementations are comparable to the corresponding TIP/AC implementations. However, the non-“early-out” Java/AC implementation performs even worse apart from *RT-1 (A)*, which is conceptually equal to the “early-out” variant.

The microscale factor (D) for cross-relation operations requires a multi-format message

Table 5.7: Throughput benchmarks for message delivery semantics

Benchmarks	Scale	Java/AC
MDS-1	A (1 redelivery)	70,585 +/-2,323
	B (2 redeliveries)	31,649 +/-1,131
	C (4 redeliveries)	14,774 + / - 513
	D (8 redeliveries)	9,456 + / - 268
	E (16 redeliveries)	4,906 + / - 139
	F (32 redeliveries)	1,995 +/-85
MDS-2	A (sequence of 10 messages)	161,918 +/-4,883
MDS-3	A (duplicate after 100,000)	172,544 +/-6,156

Throughput denoted by T_{mean} (cf. Equation (5.9)) and T_{ci} (cf. Equation (5.10)).

MSG_{MF}^{CNR} . Therefore a cross-relation operation is used for TIP/AC, which is represented by a join over the CUSTOMER and NATION relations with several conditions. For the TIP/AC implementation these operations seem more natural than for the AC/Java implementations, thus show slightly better results.

On the impact of complex message transformations: The results for the benchmark of *MT-1* message transformation of simple (A) mapping programs are shown in Table 5.6. In both cases the TIP/AC implementation outperforms the Java/AC approach, which is designed for data-aware operations on messages. Again, message transformation operations seem more natural for a data-aware implementation. Hence, further investigations on an extension or refinement of the EIP semantics for data-aware processing could be preferable.

On the impact of message delivery semantics (RQ2-3(a)): The study of the impact of the message delivery semantics (cf. *RQ2-3(a)*) touches the inner workings of the integration pipeline system, thus are only executed for Java/AC. Table 5.7 shows the microscaling of *MDS-1* (A–F) for an increasing number of retries o starting with $1 \leq o \leq 32$. The variant “use-original message” (not shown) does not show a significantly different throughput behaviour. Since *MDS-1* MRoE is a “loop” pattern, this test allows insight in the loop-processing capabilities of the runtime system. The redelivery delay penalty (without exponential backoff) becomes notable in the results for an increasing amount of redeliveries. This raises questions for future work like “Could a more scalable implementation keep up the general message throughput of the system during message redelivery?”.

For the resequencer pattern, Table 5.7 shows case *MDS-2* (A), which measures the throughput of a resequencer with a sequence size of $n = 10$. That means, after receiving 10 unordered messages, the messages are ordered and resumed. The relatively low impact on the throughput is a result of transient sequences in an operational data store.

Conceptually, the (transient) idempotent receiver and the message filter patterns are comparable. This is supported by the similar message throughput as shown in Table 5.7 *MDS-3* (A) with a duplication factor of $m = 100,000$ messages.

Conclusions.: (1) Considerable impact on processing for complex routing conditions and multiple route branchings; (2) positive impact of (data-aware) evaluation strategies like early-out (for routing only) and micro-batching; (3) trade-off: more efficient processing through differing pattern semantics required; (4) the common case of error handling (e.g., message redelivery on exception) requires more scalable runtime implementations.

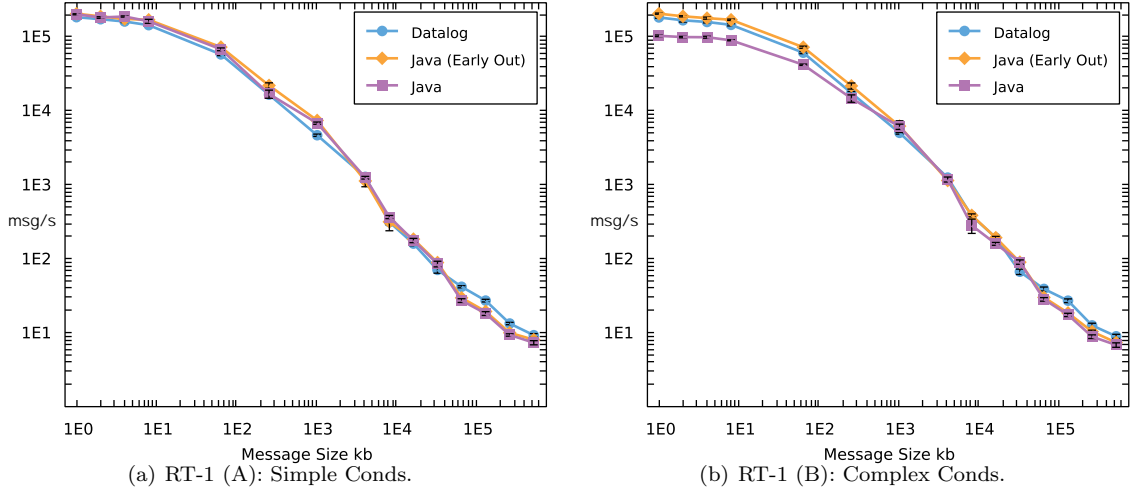


Figure 5.9: Content-based router (size scaling)

Macroscaling

To answer the “macroscale” questions RQ2-3(b) and RQ2-3(c) about the impact of message sizes, concurrent users and micro-batching, we benchmarked the routing test description *RT-1* (i.e., content-based routing) together with streams of MSG_{Ord} messages for the Java/AC and TIP/AC implementations. Conceptually the routing conditions are similar to the examples for the patterns in Section 5.3.2.

On the impact of increasing message sizes (RQ2-3(b)): The data-aware messaging question RQ2-3(b) about increasing message sizes for content-based routing leverages *RT-1* together with messages of type MSG_{scale}^{OC} . Figure 5.9 shows the immense impact of big messages for *RT-1 (A)* and *RT-1 (B)* (with *Java* denoting *Java/AC*). Notably, the data-aware implementation performs only slightly better for messages bigger than 64 MB. Especially for the TIP/AC approach, handling larger amounts of data-aware data similar to “in-memory” database table processing should be further studied.

On the impact of concurrent users (RQ2-3(b)): Especially for Machine2Cloud (cf. ST6 in Section 5.2) integration scenarios, “concurrent user” cases are common, which we formulated in question RQ2-3(b). Figure 5.10 shows the multi-threading scaling capabilities of Java/AC for the routing cases *RT-1 (A)* and *RT-1 (B)* showing an early saturation after batch $scale_{cu}(\omega)$ with $\omega = 3$. The results indicate a non-optimal usage of hardware resources through the Camel threading model [IA10], used by the EIP implementations. For instance, a thread pool can be configured for the Multicast [IA10], but not for the router pattern. However, even with a sufficiently configured threading, the multicast implementation does not reach a message throughput comparable to the router (cf. Section 5.4.2). This observation and further measurements indicate an impact on composed patterns like scatter-gather implementation (i.e., multicast and aggregator).

On the impact of micro-batching (RQ2-3(c)): For integration scenarios that trade the single message processing latency for message throughput and the overall latency (e.g., especially data movement and data processing ST5, ST6 as well as process invocation scenarios ST1–3), the processing of collections of messages RQ2-3(c), called vectorization or micro-batching, seems to be

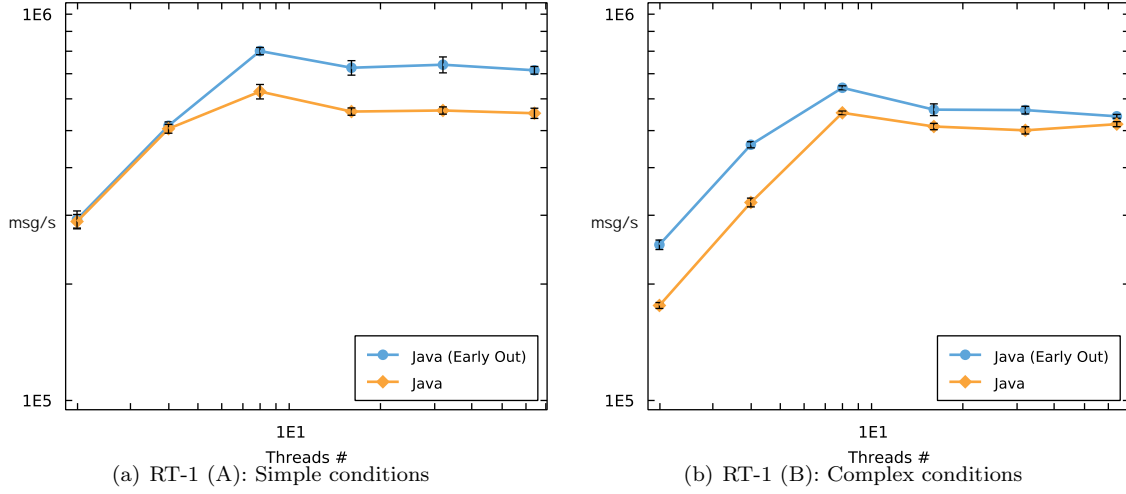


Figure 5.10: Message throughput of the content-based router (concurrent users as threads)

beneficial. Figure 5.11 shows a good scaling behavior of the data-aware TIP/AC implementation, which is able to process several messages in ONC-format with one operation. The scalability outperforms event multi-threading by factors. To fully leverage micro-batching within integration systems, the EIP semantics [HW04] have to be re-visited in future work.

Conclusions.: (5) Considerable negative impact of message size scaling; (6) non-optimal utilization of resources of (Java-based) software solutions (at least in Apache Camel); (7) trade-off: micro-batching beneficial for non-latency bound scenarios.

General Aspects and Deficits

The benchmark results show general integration system aspects, which are important for the message throughput. Besides the routing and transformation, the system is responsible for message and channel creation. For instance, the creation of messages is part of the *RT-5* and *RT-6* benchmarks, while channel creation is covered by *RT-3* and *RT-4* (not shown). The results indicate that the message creation involves time consuming operations (e.g., message ID generation, message model creation, format transformations), thus lower the throughput of those patterns. The creation of channels requires thread management (e.g., thread creation, pooling), which has an even bigger effect on the message throughput, thus making patterns like the “machine-local” load balancer [IA10] practically unusable in data-aware scenarios (cf. conclusion (6)).

Conclusions.: (8) Considerable impact of message creation (incl. format conversion).

5.5 Related Work

We survey related benchmarking approaches and analyze to what extent they satisfy core requirements for integration systems and thus help answering the questions RQ2-3(a)–(c). Based on the comparison we identify gaps in current benchmarks Table 5.8, which also influenced our design criteria for *EIPBench*.

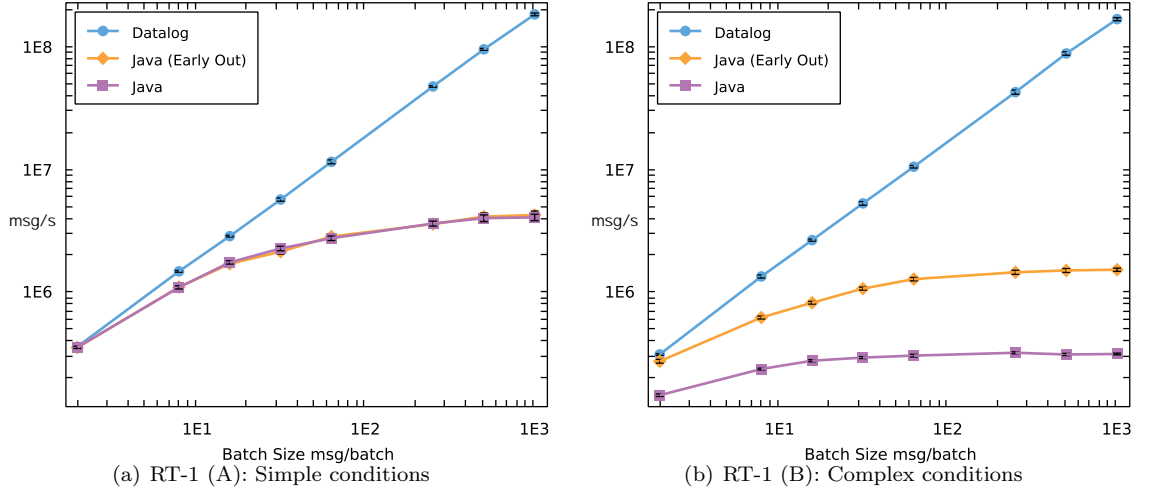


Figure 5.11: Message throughput of the content-based router (batch scaling)

For that, we categorize features of benchmarks in our field by their target system (e.g., Extract/Transform/Load (ETL), Messaging System (MS), Integration System (IS)) and scope (i.e., End-to-End (E2E) or Micro-benchmark (Micro); cf. [BBN⁺12]). We analyze the following benchmark dimensions along important IS tasks (cf. [Cha04, HW04]): (a) we conduct an evaluation of the message format definitions, leading to a differentiation between multi-format (MF), nested (NE) and simple messages. Then we check (b) how well the related work supports EIP operations on these messages (e.g., content-based routing (CBR), message transformation (MT)) and (c) message delivery semantics in general. Hereby, format conversions on a message protocol level (i.e., usually done by integration adapters [RH15]) are distinguished from those on the message content level. The scale factors for (d) concurrent user measurements (Concurrent Users) are specified as either configurable or **static** (i.e., cannot be changed), and (e) additional scale factors (SF) are shown separately. Since the EIPBench micro-benchmark, considers the operations in the integration process, integration adapter and transport protocol related topics are out of scope. These categories are discussed subsequently for each related field or target system and compared in Table 5.8 for their major representatives. To rate the maturity of a benchmark, the discussions contain hints on how recently the benchmarks were published and whether they are still actively maintained.

Integration System The only known, public integration system benchmark is the ESB Performance benchmark [Adr13], which was last executed in the year 2013. The benchmark defines E2E integration scenario performance measurements. The number of concurrent users ranges between 20 and 2,560 users, with a simple, flat XML-based payload embedded in a SOAP envelope. The test cases contain content-based routing on the SOAP header and the body with one simple string-equal routing condition using XPATH, and XSLT-based format conversions (e.g., XML to CSV). Besides concurrent users, the benchmark defines a static scale level for message sizes (i.e., from 512 B to 100 KB). In contrast, EIPBench exclusively focuses on the performance (i.e., throughput) of EIP implementations, which requires more complex message formats and more elaborate EIP operation definitions that target only the message payload (currently defined by example in JSON format). In addition EIPBench defines tests for message sizes up to 500 MB and

Table 5.8: EIPBench in the context of related work

Benchmark	Category	Transport protocol	Message format / conv.	EIPs (cf. Q1, Q2)	Message delivery semantics (cf. Q3)	Scale factors (cf. Q4, Q6)	Concurrent users (cf. Q5)
ESB Perf. [Adr13]	IS, (EIP) [E2E]	HTTP	simple (SOAP-XML)	CBR, MT (partially)	–	msg sizes, concurrent users	static
SPECjms2007 [SKBB09], jms2009-PS [SAKB10]	MS, JMS [E2E]	JMS, AMQP	n/a	–	reliable, transactional	#dest, #msgs	user sessions
TPC-DI [PRC14]	DI, ETL [E2E]	FILE, DB	simple (CSV, XML, TXT)	–	–	data (incr. load)	multiple sources
DIPBench [BHLW08a, BHLW08b]	DI, ETL, IS [E2E]	configurable	simple (XML)	–	–	data size, time, distribution	parallel streams
FINCoS [MBM08, MBM13]	CEP, Stream [E2E]	FILE, JMS	simple (CSV)	–	–	#msgs	–
EIPBench	IS, EIP [Micro]	n/a	complex MF , NE (JSON)	covered	reliable messaging	concurrent users, micro-batching, message sizes	configurable scale factor

Category: Integration System (IS), Messaging System (MS), Java Message Service (JMS), Extract / Transform / Load (ETL), Data Integration (DI), Complex Event Processing (CEP); Enterprise Integration Patterns (EIPs): Content-based Routing (CBR), Message Transform. (MT); Format: Multi-Format (MF), NEstED (NE).

reliable messaging (message retry, idempotency repository, resequencing). As transport protocol, the ESB benchmark [Adr13] uses HTTP only, while EIPBench measures the performance of EIPs in an integration process without protocol adapters (cf. Table 5.8).

Messaging System The complementary field of *Messaging System* (MS) benchmarks targets point-to-point message queuing and topic-based, publish-subscribe tests. The most prominent and still active representative is the SPECjms2007 benchmark [SKBB09], on which the jms2009-PS [SAKB10] publish-subscribe benchmark is based. Although it addresses JMS implementations only, it defines an E2E benchmark for concurrent users (i.e., connections, sessions), scale-levels in the numbers of destinations and messages, and reliable, durable and persistent message queuing (cf. Table 5.8). The latter feature is similar to the reliable messaging in integration systems, which uses messaging systems for that purpose. However, SPECjms2007 does not define an integration pattern benchmark.

Data Integration / ETL The work on data integration and ETL benchmarks can be considered conceptually related from a message transformation point of view. For instance, the recently

released TPC-DI benchmark [PRC14] defines an E2E data acquisition from multiple source systems with simple CSV, XML and TXT file data sets, and a data size scale factor for the import into multiple target systems (e.g., data warehouse). Similar to TPC-DI, the EIPBench uses the TPC-H data generator [RFSK10, PRFD11], however, EIPBench constructs more complex message formats (e.g., multi-format, nesting). The TPC-DI message transformations are format conversions as conducted by integration adapters [RH15] (e.g., XML or CSV to DB), which are different from the message transformations defined by the EIPs. The quality of service patterns in EIP are not in the focus of TPC-DI. On the other hand, the TPC-DI data quality checks are not in the EIPs, thus out of scope of the EIPBench (cf. Table 5.8).

The E2E *Data-Intensive Integration Processes* (DIPBench) benchmark [BHLW08a] is positioned as a hybrid, conceptual framework for ETL and integration system performance measurements. This discontinued benchmark targets the physical data integration within the context of ETL processes. Compared to EIPBench it does not specify EIP operations on the messages, works only with a simple XML-based message format, and neglects the message delivery semantics aspects of integration systems. Similar to our benchmark, DIPBench specifies several scale factors for data size, time and data distribution and allows to conduct concurrent user tests (i.e., parallel streams). The provided DIPBench tool suite [BHLW08b] focuses on the flexible configuration and pluggability of integration adapters (cf. Table 5.8).

Stream / Event Processing Benchmarks like FINCoS [MBM13, MBM08] target the identification of performance bottlenecks in event processing systems, by measuring event throughput and the scalability of engines when increasing the throughput of small event messages and continuous streams. Similar to EIPBench different load conditions can be configured, however, messages sizes and format complexities are static. Although the defined operators (e.g., join, select, project) are similar to the operations in integration systems, the definition does not target the EIPs.

EIPBench Summarizing our analysis in Table 5.8, none of the benchmarks covers all relevant aspects for the evaluation of integration processing in the context of data-aware scenarios. EIPBench fills this void and addresses the following aspects:

- the analysis and classification of common and new integration scenarios and the required patterns.
- representative message models used in these integration scenarios (cf. message format and conversion).
- the definition of a message throughput, micro-benchmark for EIPs that covers all benchmarking principles (e.g., relevance, scalability) and specifies microscale factors for each pattern (incl. message delivery semantics; cf. RQ2-3(a)).
- the specification of macroscale factors that address the aspects of large messages (i.e., concurrent user and micro-batching for RQ2-3(b)).

5.6 Conclusions

With EIPBench we specify the first benchmark for integration patterns, which play a crucial role for the message throughput and processing latency of integration scenarios (e.g., represented by IPTGs from Section 3.2). The benchmark definitions put emphasis on the identified micro- and macroscale factors, for which we provided a reference implementation. Based on that, we experimentally evaluated the benchmark definitions along the discussed research questions RQ2-3(a)–(c).

The experimental results bluntly show the deficits of current software-based solutions and answer our sub research question RQ2-3(a) “*What is the impact of complex message routing*

and transformation?” (relevance: cf. conclusions (1) and (4)) by notable performance drops, when increasing the complexity, the route branching or redeliveries. Even more notable for question RQ2-3(b) “*What is the impact of non-functional aspects such as message volume and concurrency?*”, the negative impact on the scalability for increasing message sizes and user scaling is significant (cf. conclusion (5)). When applying the benchmark the experimental table-centric integration processing (vectorization), featuring micro-batching, the question RQ2-3(b) “*What is the potential of new message processing approaches, and how can they be compared?*” can be clearly answered by stressing on the shown potential of such a design (cf. conclusion (2)). The evaluation also shows important deficits of current implementations for core integration capabilities (cf. conclusions (6) and (8), as well as interesting trade-offs (cf. conclusions (3) and (7)). In summary, these questions are answered by the EIPBench definition and a prototypical implementation in this chapter and denote contributions in form of DSR artifacts:

- A data set for textual message protocols,
- A benchmark definition (simple, relevant, scalable, portable) which is also extensible, e.g., for multimedia data (\rightarrow RQ3-2(a,b)),
- An instantiation of the benchmark in form of a prototype (\rightarrow RQ3-2(c)).

The benchmark is defined in an understandable way per pattern and is based on a data set consisting of textual message protocols matching the specification of the benchmark, similar to TPC datasets. Benchmarks are defined for relevant operations in application integration (cf. RQ3-2(a)) with built-in micro and macroscale levels according to the pattern characteristics (cf. RQ3-2(b)). While the benchmarks are applicable to subsequent pattern solutions, the instantiation for our two runtime systems showed the impact of the benchmarks on current solutions and on improved runtime (cf. RQ3-2(c)).

Together with the specified contributions, the identified deficits can now be studied further for contemporary technological trends such as improved pattern solutions.

Limitations Limitations of the approach concern the focus on benchmarking integration patterns in the form of a micro-benchmark. While the current EIPBench design allows for the evaluation of pattern compositions, it does not yet care for more sophisticated composition topics like benchmarking scenario partitioning (e.g., as in a multi-cloud environment, e.g., [IPE14]). This naturally leads to the question about the comprehensiveness of the benchmark. With a strong focus on patterns the benchmark might need extensions at least in the following cases: (i) for new patterns, (ii) for new scenarios with new message formats, (iii) and new processing techniques. While we have already shown the suitability for vectorization (addressing (iii)) and will discuss a benchmark extension for multimedia data in Section 6.3 (addressing (ii)), new patterns might be mostly covered by the general micro- and macroscale benchmark specifications.

We recall that EIPBench allows for the evaluation of IPTG and thus timed db-net-based integration scenarios, e.g., as Camel routes (Section 5.1). However, besides our timed db-net with boundary implementation in Section 3.2 there is currently no EAI system with corresponding modeling language that supports pattern contracts. While this would allow for correctness checking and not improve the efficiency of message processing, we leave an extension of current implementations with contract graphs and a translation of IPCGs to their executable runtime artifacts (e.g., Camel routes, processors) as future work.

Moreover, besides the benchmark results, the analysis brought up several areas for future research in the area of the benchmark (e.g., extend the benchmark for pattern composition and integration adapter processing) and more efficient message processing (e.g., routing selectivity and re-ordering, more efficient *in-memory* TIP/AC processing). To fully leverage micro-batching within integration systems, the integration pattern definitions should be extended. In this context, the system aspects message and channel creation have to be re-visited.

In general, an independent, online evaluation platform for benchmarking and comparing pattern solutions similar to TPC would be preferable.

Impact The impact of the benchmark is manifold. It not only allows for an evaluation of single pattern implementations and their compositions, but also for their comparison. With that, different variants and increments of pattern implementations can be compared and performance degradations identified immediately, and set into a broader context with alternative runtime environments. In a similar way, EIPBench is employed by SAP Cloud Platform Integration for its high frequent cloud deliveries.

Chapter 6

Pattern Solutions

Contents

6.1	Vectorization: Database-centric Pattern Solutions	205
6.2	Specialization: Hardware-Accelerated Pattern Solutions	221
6.3	Evolution: Multimedia Pattern Solutions	246
6.4	Related Work	269
6.5	Discussion	274

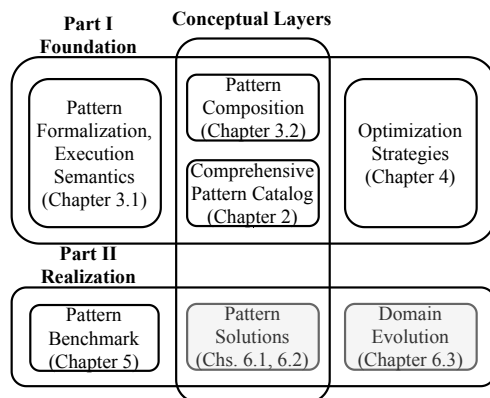
The human spirit must prevail over technology.

Albert Einstein

In this chapter we seek a more practical approach to integration patterns according to the pattern application step in [Section 2.1](#), which is about finding a suitable representation called

pattern solution (cf. [Section 2.1.4](#)) of a pattern in the context of a technological domain. We recall that [Chapters 2 to 4](#) (in Part I) feature several instantiations of the presented concepts in the form of prototypes (e.g., CPN Tools extension in [Section 3.1.3](#)). The prototypes were constructed for specific purposes such as the *responsible* development of integration scenarios and their improvements, but not for *efficient message processing* and not with the latest technological trends in mind. As discussed in [Chapter 5](#), the work on improvements of integration scenarios (cf. [Chapter 4](#)) is now complemented by more efficient pattern instantiations on the system level. We argue that the technology trends identified in [Chapters 1 and 2](#) are not only

drivers of (disruptive) change (e.g., digital transformation) accompanied by many new challenges for applications and their integration [[Rit17b](#)], but they also offer new design possibilities for existing and new pattern solutions. Hence, we explore new solutions in promising recent technological trends for a more efficient EAI processing (e.g., for message throughput, processing latency). We assess the new solutions for a productive usage and compare them to Apache Camel



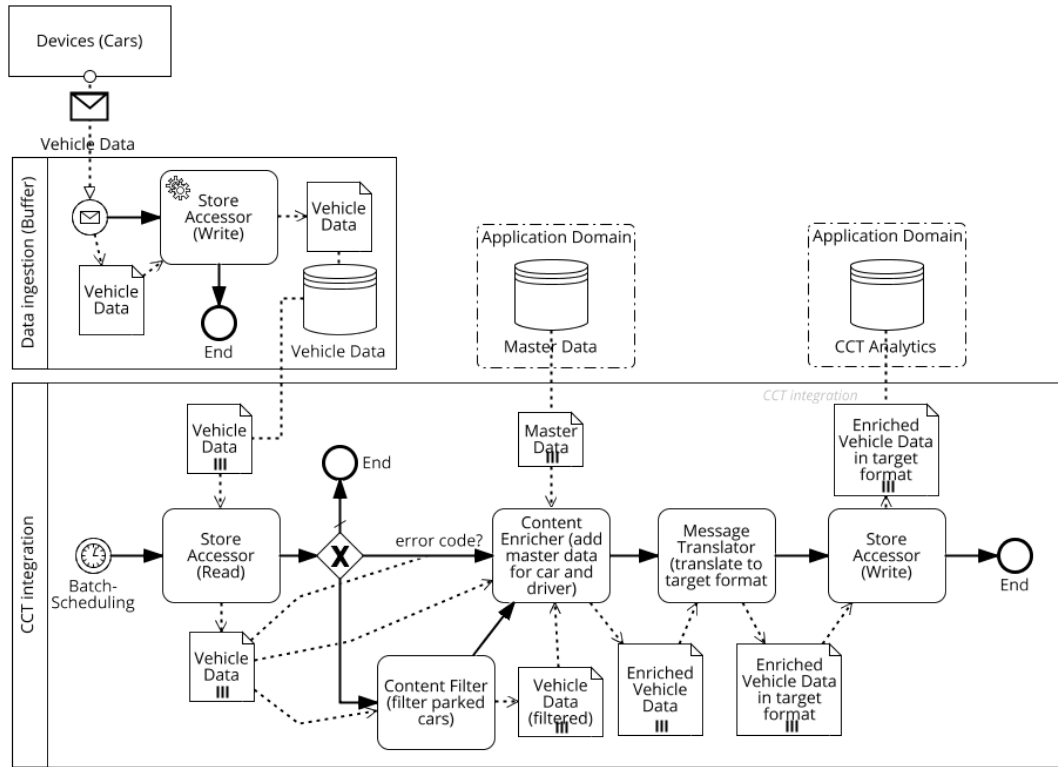


Figure 6.1: CCT scenario with focus on the integration processing

using our benchmark from Chapter 5. For efficient message processing, the following running example is considered.

Example 6.1. For scenarios like the *Connected Car Telemetry* (CCT) [SAP19c] device integration from the vehicle insights domain, integration systems receive large amounts of telemetry data. This is similar for scenarios like *contract accounts receivable and payable* (FI-CA: incl. convergent charging, invoicing and billing) [SAP19b] from the ERP financial domain (not shown), which require the systems to deal with large amounts of *billable item* information (e.g., such as call record details for song billing in media library). Consequently, EAI systems face challenges that are currently solved through special offline, “buffer-and-batch” solutions. Figure 6.1 shows a simplified realization of the CCT scenario, which will be used as motivating example in the two subsequent pattern solutions. CCT connects cars with (public or private cloud) business applications where each car (e.g., 260.3 million in the US¹) sends aggregated telemetry data and error code messages of several kB every few seconds for further processing in a data warehouse or analytic application. Briefly, the *Vehicle Data* messages represent telemetry data (e.g., vehicle speed) and error codes that are received at high frequency and volume by a message *Buffer* and stored during the data ingestion into the *Vehicle Data* table. We focus on the CCT processing that

¹Number of vehicles in the US, visited 5/2019: <http://www.statista.com/statistics/183505/number-of-vehicles-in-the-united-states-since-1990/>

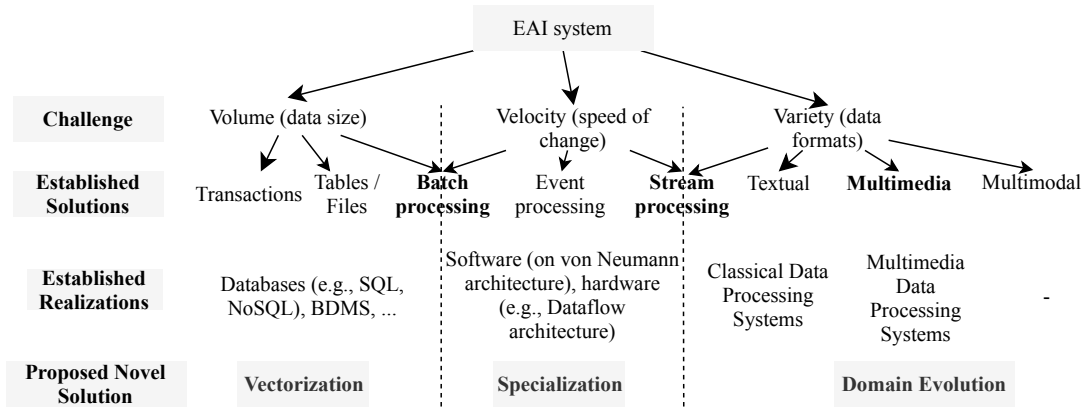


Figure 6.2: Solution proposals for the three Vs in related domains

includes integration semantics like content-based routing (i.e., to distinguish between the telemetry and error code data), content filtering (i.e., removing unnecessary data), content enrichment (i.e., adding master data), and translation into the format of the target application. Furthermore, the message senders (*Devices*) and other application domains (i.e., *Master Data*, *CCT Analytics*) are decoupled. Besides the de-coupling of a growing number of distinct CCT senders, they produce varying workloads that are currently difficult to process by applications. Current solutions show a tight coupling between senders and the CCT application and are implemented in a process-like but “ad-hoc” manner, however, do not formalize, improve or optimize the processing according to the integration semantics. The offline, batch-processing nature of these solutions prevents them from (near) real-time analysis of the data, which requires novel solutions. We observe that solutions for scenarios like CCT and FI-CA are currently built in the same way, and thus they have the same problems. ■

In the spirit of Einstein’s quote about technology becoming part of our everyday lives, we argue for a productive application of the identified trends (i.e., volume, velocity and variety) in favor of improved pattern solutions, according to our *efficiency* sub-question RQ3-1 “Which related concepts and technology trends can be used to improve integration processing and how can this be practically realized?” to derive novel solutions (e.g., for scenarios like CCT and FI-CA). Together, RQ3-1 and RQ3-2 (cf. Chapter 5) target an answer to the general research question RQ3 “Which related concepts and technology trends can be used to improve integration processing and how can the resulting integration solutions be practically realized and compared?”. We explore novel pattern solutions based on some of the research challenges that we identified in Chapters 1 and 2: data volume, velocity and variety. Figure 6.2 classifies the challenges in the context of their characteristics and current implementations in related domains like databases or data integration from Figure 1.1 (on page 3). The research *challenges* come with *established solutions* for these challenges in related domains. These solutions have common *established realizations* in the form of software systems or hardware.

The challenge of (big) data size or volume processing has existed for a long time and fostered the conception of (relational) databases for structured data (e.g., [Cod70]), and later NoSQL databases for unstructured data (e.g., [Cat11]). The pre-dominant data processing style is *batch processing* or *vectorization* (e.g., to build aggregates) [SAP19c] as discussed in Section 5.1.2. More recently, the ever growing amounts of data lead to the development of so called Big Data Management Systems (BDMS) like Asterix DB [AAA⁺14] or SAP HANA [MLH⁺15, FML⁺12, FCP⁺12]. The

BDMS allow for uniform access to structured and unstructured data across multiple heterogeneous stores (e.g., row, column, graph, object) [RPBL13]. In the spirit of active databases [WC96, PD99], BDMS emerged as data-centric (application) programming platforms with uniform processing capabilities from algebraic [KKL15] and statistical data processing [GLW⁺11] up to user-defined functions [BRFR12] and built-in machine learning capabilities [BBC⁺12]. We recall the promising message throughput results for the Apache Camel *vectorization* extension benchmarked in Chapter 5. However, we also noted a decline in performance for growing amounts of data, which is targeted by big data management systems. Consequently, our first pattern solution in Section 6.1 targets an implementation of integration pattern compositions within a BDMS. This is not an implementation of timed db-nets from Section 3.1, but we give practical hints on similarities and differences in terms of the transition system. A formal translation from timed db-nets to BDMS and proofs of their equivalence is considered as future work.

The speed of change, called velocity, is the second of the contemporary challenges in EAI. While current solutions support batch processing (e.g., see Example 6.1), other forms of processing gained more importance. For smaller, higher frequent data packets, the domain of (complex) event processing emerged. Since events are one type of EAI messages among many others according to [HW04], event processing denotes a subset of EAI as experimentally studied in [ES13] using Apache Camel [IA10]. A processing style that was developed for arbitrary data sizes is stream processing or streaming (e.g., [GÖ03, LG03]). Contemporary software systems include but are not limited to those by the Apache Foundation: Spark [ZCF⁺10], Flink [CKE⁺15], Storm [IS15] and Kafka [Gar13]. Most of them combine batch and stream processing similar to the table-centric processing system based on Camel used for our benchmark Section 5.1.2. However, they still denote software solutions based on the von Neumann hardware architecture, and thus provide instruction streams and not data streams according to [Bac78, AI83]. A better suitable alternative is denoted by a Dataflow architecture [Cul86], which is currently best implemented in the form of hardware specializations (cf. [Har97, Rit17b]). Reprogrammable hardware like *Field Programmable Gate Arrays* (FPGAs) represent the best balance in the trade-off between efficiency represented by ASICs and flexibility such as software [Rit17b]. Consequently, our second pattern solution targets an implementation on reprogrammable hardware in the form of a Dataflow streaming EAI system in Section 6.2. Again, we do not claim equivalence between the resulting solution and timed db-nets.

The emergence of new data formats during EAI processing concerns its variety (cf. Chapters 1 and 2). While EAI was built for textual message formats (e.g., tabular, hierarchical, graph) [Lin00, HW04], social (media) trends require binary and multimedia data processing (e.g., image, video), and eventually combined textual and multimedia (short multimodal) processing. The new message formats are required for new integration scenarios as part of the digital transformation of more and more aspects of our lives. Consequently, the final pattern solution addresses the new variety challenges in Section 6.3 by specifying and studying EAI processing with multimedia data, which leads over to more complex user interactions and configuration. After these solutions are set into context in Section 6.4, we conclude in Section 6.5 with a discussion on several important lessons learned (incl. the relationship between their execution semantics and timed db-net).

The three studies show the great potential of emerging technologies in the context of the integration patterns and pave the way for industrial solutions. Thereby the solutions do not only improve existing ones, but allow for new innovative business models (e.g., near real-time billing or car analytics) and more sustainable solutions (e.g., with respect to energy consumption).

Parts of this chapter have appeared in the proceedings of BICOD 2017 [Rit17a] (vectorization), DEBS 2017 [RDMRM17] and ACTIVE@Middleware 2017 [Rit17b] (specialization), and EDOC 2017 [RRM17] (variety, domain evolution).

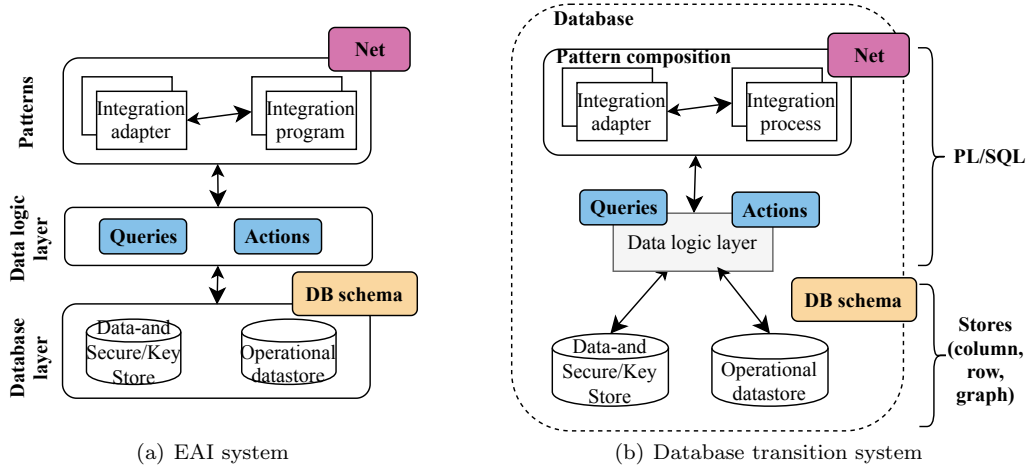


Figure 6.3: Timed db-net realizations

6.1 Vectorization: Database-centric Pattern Solutions

In this section we revisit the system design of our timed db-net from Section 3.1 in the context of the *vectorization* processing style (cf. Section 5.1.2) and modern *Big Data Management Systems* (BDMS). While the proposed solution is correct with respect to the execution semantics of the realized integration patterns, we do not formally show an equivalence with timed db-nets, nor do we compare the CPN Tools prototype message processing capabilities with our BDMS pattern solution. However, we give an intuition on similarities between the two solutions. In particular, the design of db-nets in Figure 3.4 (on page 69) denotes a control layer separate from the data logic and persistence layers. Figure 6.3(a) sets this design into the context of the EAI system architecture in Figure 2.4 (on page 26). The integration *patterns* (e.g., *integration adapter*, composed patterns denoted by *integration program*) are defined on the *pattern* layer (i.e., timed db-net control layer), requiring (mediated) access to the database layer (i.e., timed db-net persistence layer). While the differentiation of these layers allows for a formal analysis as discussed in Section 3.1.2, the design is not efficient from a system perspective: The data moves on the control layer and requires frequent updates of the lower layers (incl. data shipment).

Example 6.2. We recall the CCT scenario in Figure 6.1, where the data is shipped several times between the integration logic denoted by *CCT Integration* and the *Application Domains* (i.e., *Master Data*, *CCT Analytics*) with corresponding actions: read *Vehicle Data*, read *Master Data*, and write *CCT Analytics*. ■

For a more efficient interaction with the database layer, Vrhovnik et al. [VSS⁺07, VSES08] propose a solution for the related business process systems. In terms of timed db-nets this requires an adaptation of the data logic layer for more efficient mediation between the control and persistence layers (e.g., token population to view places). Alternatively, there are ideas inspired by the active database domain [WC96, PD99] to implement integration processing within a (federated) database management system [Böh11, Rit14c], and thus leverage their functional (e.g., structured, batch data processing) and non-functional capabilities (e.g., scalability, transactional guarantees). Essentially, this means to “push-down” the integration logic from the application

tier (incl. sender / receiver decoupling) into the database, by representing the transition system as well as the patterns with database means (e.g., PL/SQL, stores) as depicted in Figure 6.3(b). We call such a solution a *Database Transition System* (DTS). However, this has been difficult so far due to the limited expressiveness of the (active) database capabilities (e.g., no transactional decoupling between sender and receiver possible with database triggers).

In the context of the shift of traditional relational database management systems towards BDMS (i.e., essentially data-centric application programming platforms), we argue that not only could applications having their data in the same database system benefit from integration processing within the database system, but also that EAI systems could “push-down” their semantics to the database in order to improve their processing. More precisely, we found that there are three types of integration scenarios that would benefit from this [Rit14c]: (C1) two applications with data in the same database system with built-in integration logic (\mapsto internal / internal), (C2) one application with data on the database system (with integration logic) receives or sends out data (\mapsto internal / external), (C3) two applications that have no data on the database system with integration logic (\mapsto external / external). The table-centric integration processing described in Section 5.1.2, investigates (C3) on the application tier by “moving up” database processing techniques into an integration system. The results were especially promising for message throughput and scalability. While (C2) is partially covered by JMS-like message queuing extensions of database systems [GM03] (i.e., *Buffer to Vehicle Data* in Figure 6.1), integration processes in the database (cf. (C1)) were only partially addressed by evaluating nested views for engineering applications in [HMWMS87] so far, and thus proposed by [BLN86] as future research. Consequently, we address the internal / internal (C1) case and investigate the integration pattern semantics and the transition system in the context of relational database processing along the following research sub-questions of RQ3-1 with respect to databases (*DBx*):

- (DB1): “How can EAI semantics be represented in database systems?”
- (DB2): “How can EAI processing semantics be compatibly adapted to databases?”
- (DB3): “Can databases accelerate message processing?”

While question DB1 targets basic EIP semantics like the representation of messages and message channels, as well as message processors that require extended semantics (cf. Chapter 5), DB2 is about a transactional process model that is compliant with the standard integration pattern execution semantics. We measure the message throughput using the EIPBench benchmark for selected EIPs for answering question DB3.

We briefly introduce the required database foundations Section 6.1.1, before discussing database processes in Section 6.1.2 starting from basic integration principles (e.g., document message, datatype channel, canonical data model) to the transition system (cf. DB1, DB2), and discuss alternative pattern semantics, more efficiently represented on databases, and their representation in timed db-net. Note that we do not formally map the timed db-net integration patterns and their composition to database processes, however, briefly discuss similarities between the concepts. The evaluation includes throughput and latency studies according to EIPBench (cf. Chapter 5) and the motivating CCT example (cf. DB3).

Parts of this section have appeared in the proceedings of BICOD 2017 [Rit17a].

6.1.1 Transaction Processing and Big Data Management Systems

For a better understanding of our proposed database transition system solution, we briefly introduce their underlying (big data) database management and transaction processing concepts. We assume basic knowledge about relational databases and ACID transactions, referred to simply as transactions. If unfamiliar with these terms or concepts, see [UGMW01, EN10, KE11].

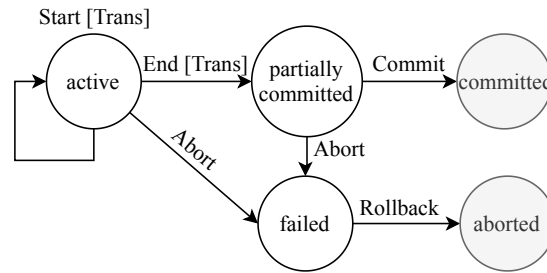


Figure 6.4: Database transaction states (adapted from [KE11])

Database Transaction Processing and Programming Model

The consistency of database system state changes (e.g., read, write) is preserved by transactions that denote a sequence of read and write operations as well as computation steps. Transactions might be executed concurrently, and failures might occur, which requires concurrency and failure transparency. An *active* ACID transaction always terminates as shown in Figure 6.4 either by a *commit*, after the last task or statement has been successfully executed, or *abort*, after the *rollback* of a transaction (e.g., system failure, unsatisfied condition, deadlock, crash), or explicit abort. In the latter case, the transaction performs a rollback, which means that the actions are undone, and the database returns into the state before the execution. When a transaction successfully commits, the results are permanently stored in the database (known as durability), and the results are visible to other transactions (known as consistency, isolation). During the execution of the last operation of a transaction it goes to a *partially committed* state, which gives the possibility to abort or commit the transaction. Within a database system, transactions are abstracted by a programming model. The subsequently introduced programming model concepts for transaction processing (transaction bracketing, chained transactions and boundary operations, and savepoints) are summarized from [GR92, BN09], if not stated otherwise.

Transaction Bracketing Database programming models implicitly or explicitly offer transaction bracketing, which allows for the specification of the commands along the states in Figure 6.4 to start, commit and abort a transaction. The commands “bracket” a transaction by identifying which operations execute in the scope of that transaction. The start command creates a new transaction, which means that all subsequent operations are part of that transaction until commit or abort are invoked. The application or program that invokes a transaction finds all its procedures executed within that transaction. The bracketing of several procedures into one transaction might lead to problems like the transaction composability problem, if one of these procedures start new transactions. Similarly, the abortion of a transaction in one of the procedures requires further specification of the semantics in the form of *nested transactions*. Since we make no use of nested transactions and enable composability by allowing only one start and one commit or abort in larger transactions, we refer the interested reader to [GR92, BN09].

Chained Transactions and Boundary Operations Usually, in a database system, tasks or procedures are executed within a transaction. Consequently, the transactions can be created by the system, and thus only require the specification of a “boundary” between the transactions. To ensure that the programs are always executed within a transaction, these so called “boundary operations” commit one transaction and immediately start another one (e.g., IBM’s syncpoint).

Since the sequence of transactions is executed in the form of a chain, this programming style

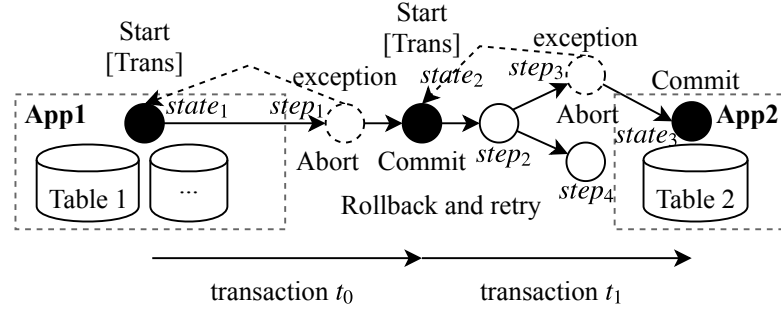


Figure 6.5: Transaction processing example

is called *chained transactions*. Alternatively, a program controls the creation of transactions, which does not need to happen right after the commit or abort of a previous one, and is thus called *unchained*. While this seems to be more flexible, it mostly makes sense if transactions have significant processing overhead, even if it does not access recoverable data.

Savepoints In the presence of (unsolicited) aborts and exceptions during the execution, transactions can periodically save its state for recovery (e.g., by the database exception handler). This abstraction that does not undo all of the transaction's effects is called a *savepoint*. These savepoints can be programmatically issued through a savepoint command, which tags certain points during the execution of a transaction. Even if a transaction must be aborted and neglects the possibility of a retry, the saved information allows for the generation of diagnostic information about the exceptional situation. With savepoints, nested transactions can be realized.

Example 6.3. The transaction processing in database systems is shown by example in Figure 6.5 as a directed graph. The persistent tables or savepoints are depicted as black and transient operations or data types as white nodes. The first transaction brackets one operation $step_1$ that reads from Table 1 (i.e., $state_1$) and writes into an intermediate persistent node $state_2$. If the operation finished successfully, transaction t_0 commits with *commit*, otherwise it aborts with *abort* and performs a rollback to the previous state $state_1$. The second transaction t_1 of the chained transactions brackets three operations $\{step_2, step_3, step_4\}$ and inserts the records into Table 2 of App2 (i.e., $state_3$). In case of an exception (e.g., in $step_3$), the transaction aborts and can be started from the intermediate persistent node $state_2$, instead of Table 1 (i.e., $state_1$).

Notably, during the execution of the first instance of t_0 , a concurrent, second instance t'_0 would work on the same records until t_0 removes the records from $state_1$ and commits them in $state_2$. Moreover, the chained transactions are executed as part of what is called *App1*. ■

Remark 6.4 (Batch Processing). In the context of transaction processing, batch processing systems execute each batch as a sequence of transactions [GR92, BN09]. To avoid serializability problems (e.g., through concurrent execution), the transactions are executed one transaction at a time. Moreover, batch processing can be configured with respect to their time of execution and the batch size (number of records). ■

Big Data Management Systems

In recent years, relational database management systems evolved into data-centric programming platforms, called *Big Data Management Systems* (BDMS) [AAA⁺14]. This trend started at

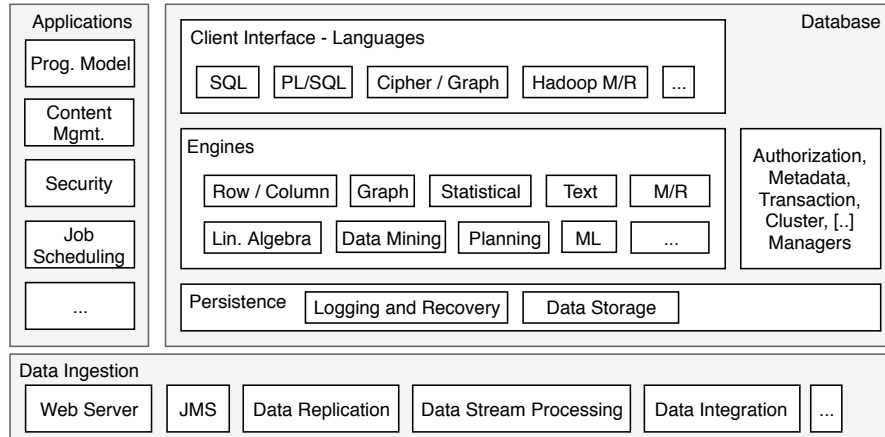


Figure 6.6: Big data management system architecture sketch (adapted from [FML⁺12, MLH⁺15])

the latest in 2008 with the fusion of OLTP and OLAP engines, which essentially combined the domains of transactional and analytical applications (e.g., SAP HANA database [FML⁺12, MLH⁺15], HyPer [KN10, KN11]). To support richer analytical applications and application services, more capabilities were added meanwhile, thus making the database a data-centric programming platform for applications. Figure 6.6 depicts a sketch of such a BDMS with a focus on relational processing that we compiled from the classical database architecture and the extensions (cf. [FML⁺12, MLH⁺15, AAA⁺14]). Notably, the core database architecture remains mostly the same. The data is stored in the *Data Storage*, the *Engines* access the data on behalf of the applications, which use predefined *Client Interfaces*. Thereby, client *Authorizations* are checked, the *Transaction Manager* ensures the consistency of the operations, *Metadata* is provided and the *Cluster* is managed.

The emergence of hybrid OLTP and OLAP systems leads to new *row / column* engines and stores [FML⁺12, MLH⁺15, KN10, KN11], which was extended by graph [RPBL13, GCKP13], algebraic [KKL15, LGG⁺18] and statistical data processing [GLW⁺11] up to user-defined functions (e.g., [KE11, BRFR12]) and built-in machine learning capabilities [BBC⁺12] as discussed before. However, the most notable extensions came with *Data Ingestion* and *Analytic Application*, which allowed for the development of data-centric applications within or at least close to the database. Therefore, the already existing data channels like *Data Replication* and database *JMS* broker [GM03], were complemented by general *Data Integration* and *Web Server* (e.g., [GC15]) as well as *Data Stream Processing* (e.g., [AAA⁺14]) capabilities. For the development of applications, data-centric application *Programming Models* have been proposed (e.g., Microsoft Azure Data Lake [RSD⁺17] with big data language [CJL⁺08]), and *Content Management* as well as *Security* capabilities were added. Moreover, (cron-like) *Job Scheduling*^{2,3} allows for timed application logic down to the database constructs.

Example 6.5. With the job scheduling capabilities, database client interface constructs (e.g., in SQL and PL/SQL) can be scheduled. Consequently, the transaction coupling (e.g., of active database constructs like triggers) can be overcome. In other words, the two transactions t_0 and

²Microsoft SQL Server — Schedule a Job, visited 5/2019: <https://docs.microsoft.com/en-us/sql/ssms/agent/schedule-a-job?view=sql-server-2017>

³SAP HANA database — XS Engine job scheduling, visited 5/2019: <https://blogs.sap.com/2015/03/19/step-by-step-procedure-to-schedule-job-in-sap-hana-to-execute-stored-procedure/>

Message (optional)		Body			
Message ID <int>	fault <int>	Message ID <int>	Data Col 1 <type 1>	...	Data Col N <type N>
1	0	1	'#1'	...	'invoice'
2	0	2	'#1'	...	'order'

Header (optional)			Attachments (optional)		
Message ID <int>	Key <string>	Value <string>	Message ID <int>	Key <string>	Value <blob>
1	sender	IBM	1	invoice	<doc>
2	sender	CISCO	2	order	<doc>

Figure 6.7: Multi-relational message collection

t_1 in Figure 6.5 could be scheduled in “system”-controlled transactions with read-privileges for *App1* tables and write-privileges for *App2* tables, and move data between different application domains within the database, without requiring *App1* to access tables of *App2* as well as having no data movement in a transaction of an application. This essentially allows for the decoupling of applications in the database. ■

This new type of data-centric application programming system allows for building rich applications (e.g., EAI system) close to the data that leverage the database transactions, scalability and big data processing capabilities.

6.1.2 Database Transition System

As motivated in Figure 6.3(b), the complete EAI processing logic is represented within a relational database close to the data of applications. Therefore, we map the integration semantics in the form of integration patterns and their compositions to database concepts in a *Database Transition System* (DTS), in which the complete pattern compositions are executed. In terms of timed db-net, the control and data logic layers are represented by (procedural) database constructs, and the persistence layer denotes different kinds of stores like row, column or graph stores.

Relational Pattern Definitions

First we introduce basic integration semantics and discuss the implications on the integration patterns and give realizations for those of the motivating CCT example: Message, Message Channel, Message Filter, Content-based Router, Message Translator.

Basic Principles: Multi-relational Message Collections Set-oriented processing changes the structure of messages, however, does not require any change at the message channel level — although the current document message or the datatype channel could be specialized to indicate the nature of the relational messages exchanged. According to Section 5.3, a message $m := (b, H_b)$ consists of a body b with arbitrary content and a set of name / value-pair meta-data entries describing b , called the header H_b . An extension for multimedia data is a set of name / binary value attachments A . For example, an e-Mail adapter is able to receive messages with a textual body and attachments.

For relational message processing, the body b has to be restricted to relational message contents. For example, Figure 6.7 denotes the body b_r , header H_{r,b_r} , and attachment A_r relations. In addition, a control record for each message is inserted into an optional message relation, which manages the mapping of messages to faults that happened during processing. As such, m_r denotes a *multi-relational* message, similar to a relational variant of a multi-format message from Section 5.3, since it has to be represented using one relation for each of the distinct formats of the message entries. A relational message $m_r := (b_r, H_{r,b_r}, A_r)$ consists of a relational message body b_r in normalized form, and sets of headers H_{r,b_r} and attachments A_r represented by ternary relations in the form of name / value character, and name / binary character-clob/blob fields. When considering b_r to be in BCNF, a relational message m_r consists of at least one and up to n relations, allowing for different, possibly joining body relations. In conventional message processing, the body table would have only one entry in each of the corresponding database tables, representing the content of one message. For more efficient processing in databases, we define a relational message collection MC_r of m_r . A (*relational*) *message collection* $MC_r := (B_r^{f_1}, B_r^{f_2}, \dots, B_r^{f_k}, H_{r,b_r}, A_r)$ consists of a collection of k message content sets B^f of the same message format f , H_{r,b_r} (cf. *data col* with $\langle type \rangle$) and A_r relations. While the header and attachment relations can be shared between messages, the body might require additional relations, if the message formats differ.

In practice, the constructed MC_r can be represented by a table instance for each relation. With this representation tree- and graph like message formats (e.g., XML, JSON) can be expressed. However, the header and attachment relations, which are added as input and output to each message processor, are accessible using relational algebra (e.g., selection, join) in contrast to standard XML processing (e.g., not considered in [BK11]).

Selected Relational Integration Patterns We recall that the pipeline model of integration scenarios consists of message channels that connect message processors by passing the outgoing message of the previous processor to the subsequent processor as input message (cf. Section 2.1). These processors are generally categorized as routing and transformation patterns. We focus on two of the patterns from the motivating example (cf. Figure 6.1), namely a Content-based Router (incl. Message Filter) and a Message Translator. For a general expressiveness discussion of relational integration patterns, we refer to Section 5.1.2 and [Rit14c, Rit15b].

Content-based Router. The semantics of the Content-based Router, given by the timed db-net in Figure 3.10(a) (on page 81), are algorithmically described in Listing 6.1 to put some more emphasis on the message generation aspects. With a message cardinality of 1:1 and channel cardinality of 1: n (actually 1:1, since only one channel is selected) the router checks the message content sequentially (cf. line 2). If a channel condition $ch_i.cond()$ matches the original message m , then m is routed to that channel, and no further condition is examined (cf. line 3). Otherwise, the other channels are checked in sequence until one matches. The message is routed to a default channel $ch_{default}$, if none of the conditions match (cf. line 6). Since the original message is forwarded, the router is read-only and non-message generating. While this definition preserves the semantics of the router according to Chapters 2 and 3, for a database system this essentially means to process a single record in a table at a time, which is not efficient.

On Database. In the context of (set-oriented) database systems, these semantics can be changed for a more efficient parallel message processing. The actual message and channel cardinalities are adapted for sending collections of messages to multiple receivers. Listing 6.2 shows the modified semantics with a focus on the message generation, in which all channel conditions are evaluated on a set of messages MC_r in parallel (cf. line 1). If a message content matches a channel condition, the message is added to a new message collection ($MC_r^{ch_i}$; cf. line 7) that is created per channel (cf. lines 4–6). Consequently, each channel condition results in a set of records (i.e., message

collection). The default channel can either be added to the list of channels C beforehand or executed for all messages $m_r \in MC_r$ that are not in one of the message collections of one of the channels (cf. lines 8–10). For each channel ch_i the message collections $MC_r^{ch_i}$ can be routed in parallel (cf. line 11).

Listing 6.1: Original CBR semantics

```

1 sequential for channel  $ch_i \in C$ ,  $m$ 
2   if  $match(ch_i.cond(), m)$  then
3     route( $m, ch_i$ ); return;
4   end if;
5 end for;
6 route( $m, ch_{default}$ );
```

Listing 6.2: Table CBR semantics

```

1 parallel for channel  $ch_i \in C$ 
2   create( $MC_r^{ch_i}$ )
3   for message  $m_{r,j} \in MC_r$ 
4     if  $match(ch_i.cond(), m_{r,j})$  then
5       add( $m_{r,j}, MC_r^{ch_i}$ )
6     end if;
7   end for;
8   if  $MC_r^{ch_i} = \emptyset$ 
9     add( $MC_r, MC_r^{ch_{default}}$ );
10  end if;
11  route( $MC_r, ch_i$ );
12 end for;
```

The new semantics are formally defined by the timed db-net in Figure 6.8. The message cardinality is $n:m$ (depicted by input $List(msg)$ and output $\{List(msg_1), List(msg_2), \dots, List(msg_k)\}$) and the channel cardinality changes to $1:k$ (depicted by output channels $\{ch_1, ch_2, \dots, ch_k\}$), which means that several messages can be processed in one transaction. Essentially, the resulting semantics allow for the routing of one message to several endpoints, and thus changes the ordered, only-one-receiver semantics, which can be validated and verified in timed db-net.

Content Enricher. The semantics of the Content Enricher pattern, given in Figure 3.10(b) (on page 81), mostly remains the same. The enricher adds content to an existing message, if the message originator does not provide all the required data items. The enrichment can be done (a) statically (i.e., constant), (b) from the message itself or (c) from an external source. We consider (c), while the external data is queried from another database table within the same system.

On Database. During the enrichment in the database, queries for the totality of enrichments for all messages in the input set have to be handled. Thereby, the message cardinality changes from $1:1$ to $n:n$, but the channel cardinality remains the same, and thus the timed db-net stays structurally the same (not shown).

Message Translator. The semantics of the translator, given in the message processor example Figure 3.44 (on page 122), is to translate one message content to another to make it understandable for its receiving processor or endpoint [HW04]. The translator has a channel and a message cardinality of $1:1$. It does not generate new messages, but modifies the current one.

On Database. A database Message Translator changes its message cardinality to $n:n$, allowing for a more efficient translation of several messages in one MC_r . Again, the translation could be executed in parallel (e.g., by partitioning MC_r), depending on the underlying database system. Note that the timed db-net stays structurally the same (not shown).

Relational Message Processing

For the execution of pattern compositions we specify a suitable transactional processing model based on the considerations from timed db-nets by defining an adapted, but compatible message processing semantics.

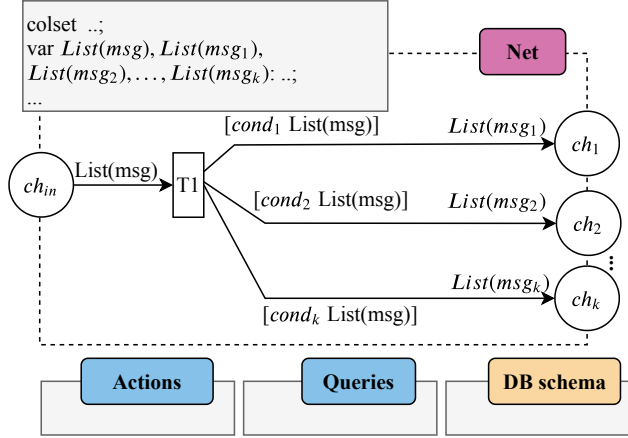


Figure 6.8: Timed db-net realization of a vectorizing content-based router

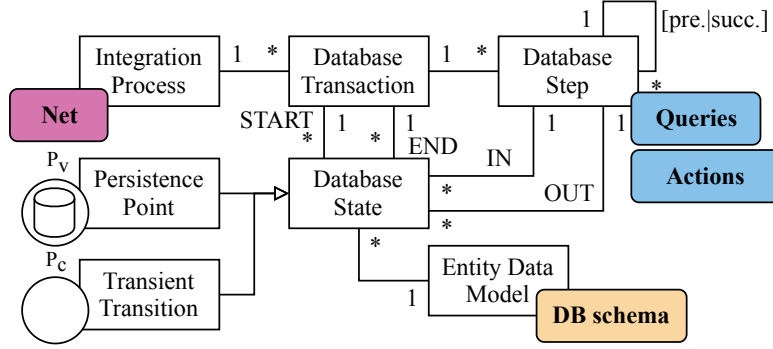


Figure 6.9: Transactional process model

Transactional Process Model For the processing semantics of the database transition system, we recall the timed db-net based processing model of integration compositions from [Chapter 3](#). In timed db-nets the control and dataflow in the *Net* does not require database transactions, if only control places and transitions without persistence layer operations are used. For the persistence operations, the view places continuously read data in read transactions, and transitions execute actions in one transaction. Although timed db-net transactions provide ACID guarantees, they are atomic or “unchained” [\[GR92, BN09\]](#). Through the unchained transactions, the data logic needs to start, abort and commit the transaction bracket for each operation.

In contrast, with a database transition system as depicted in [Figure 6.3\(b\)](#) (on page 205), the control, data logic and persistence layers are represented within the database, and thus fully transactional (similar to “chained transactions” [\[GR92, BN09\]](#)). In other words, a pattern composition is assumed to be always executing within a transaction, which can be realized by chained transactions and boundary operations. That means, when a boundary operation commits one transaction another transaction immediately starts. For exceptional cases persistent places similar to IBM syncpoint (similar to the concept of savepoints within transactions [\[GR92, BN09\]](#)) are required.

When assuming that a pattern composition is always executed within one or many database transactions (e.g., by chained transactions and boundary operations [GR92, BN09]) and the applications store their data in the same database without shared schema access (i.e., sender, receiver decoupling), the resulting model for a database transition system is shown in Figure 6.9. We have associated timed db-net concepts to the concepts of our transactional process model to attempt a mapping for a better understanding of their relation. A formal relationship is left for future work. Now, the pattern composition or processing instance similar to a timed db-net *Net* is granted read-only access to the sender data and write access to the receiver data and consists of at least one database transaction, where the control and data flow are mapped to database constructs. Notably, we do not consider nested transactions [GR92, BN09]. Each transaction is composed of database steps (similar to timed db-net transitions) and states (similar to timed db-net places). The database steps behave similar to timed db-net transitions by moving the data from a source to a target state, while executing *Queries* and *Actions*. Database states can be persistent on disk through committing the transaction (similar to timed db-net view places) or transient as data type definitions within one transaction (similar to timed db-net control places). Since every database step is part of a transaction, a transaction requires at least a persistent start and end state and arbitrarily many intermediate persistent (e.g., table) or transient (e.g., table type) states. A step processes the data from a preceding state and moves it to a successor. The control flow and dataflow map to the database states that are bound to entity data models as *DB schema* (i.e., message formats). Each state has a step “writing” to or “reading” from it. Transient states are parameter types within PL/SQL procedure calls.

Example 6.6. Figures 6.10 and 6.11 illustrate the described processing model in terms of the three most common structural pattern categories (cf. Section 3.2.1) as directed graph with steps denoting white nodes and states denoting black nodes: message processor (cf. Figure 6.10), fork (cf. Figure 6.11(a)) and join (cf. Figure 6.11(b)). The sending and receiving adapters or stateful processors represent persistent states (savepoints marked by black nodes) and message processors without state are denoted as steps with transient state (white nodes). In one transaction, messages are read by $step_1$ from the source (i.e., state $state_1$ for message processor and fork, and states $state_1, state_2$ for join) to the target state(s) (i.e., state $state_2$ for message processor, states $state_2, state_3$ for fork, and state $state_3$ for join), while grouping sets of updates into a single transaction (as mentioned in [GR92, BN09]). ■

Database Message Processing For database message processing, let us assume an instantiation of the transactional process model from Figure 6.9 on a database. There, the message is processed in one transaction from one persistent state to another (i.e., store and forward [Lin00]). This would become a challenge for integration scenarios represented by timed db-nets, since it might result in several cross pattern sub-processes. Note that in future work, this could be conceptually approached by considering the pattern boundaries to be persistent *states*, which would give each pattern at least one dedicated transaction and avoid cross-pattern transactions. However, additional transactions with persistent states might lead to performance degradations.

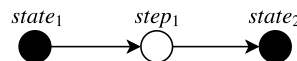


Figure 6.10: Structural pattern category: message processor

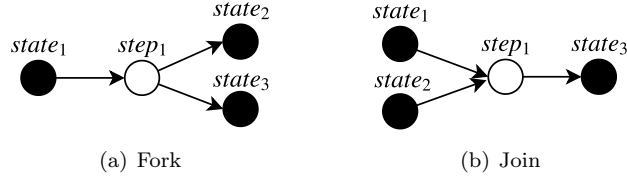


Figure 6.11: Structural pattern categories: fork and join

Back to the database processing, in case of an error, the original message is redelivered from the last persistent state. However, on current database systems the transaction processing does not support setting a message or a collection MC of messages into “in-flight” (i.e., marked as already taken up by another transaction), but they are still visible to other transactions. Figure 6.12 illustrates the processing on a database denoting source $table_s$ and target $table_r$ tables as persistent states, and several transient message processors (i.e., no persistent state in-between). At the start of the processing, the messages msg_1 and msg_2 in the current message collection MC_{tx_1} are read in transaction tx_1 , and subsequently processed by transient message processors, before the messages are inserted in the target table. Since database triggers are executed within the sender’s transaction context, we assume the transactions to be started by a controller or *scheduler* component (cf. BDMS job scheduling capabilities in Section 6.1.1; similar to the operator scheduling in streaming systems [CCR⁺03]). The scheduler processes a new transaction either when an event is triggered, clocked or like a window-operator (e.g., cf. [CKE⁺15]) on a certain message collection size, illustrated by scheduler instances s_1, s_2, \dots, s_n in Figure 6.12. Within one sub-process transaction, the data is read and deleted (i.e., marked for deletion) from the source table, processed and inserted into the target table, and then committed. The messages in the source table $table_s$ remain visible for other consumers and become visible only in the receiver table $table_r$ after commit. In case of an exception, the messages will be redelivered in a new transaction (not shown). The difference compared to the conventional EAI processing is that the messages in the source table remain visible, which can result in duplicate messages. To avoid duplicates, a stateful filter can be added that removes already processed messages (e.g., leveraging unique message identifiers or through logging the processed messages for each step). Similarly, the BDMS scheduler could ensure the “in-flight” transaction behavior by filtering the message collections. While both mechanisms require additional, complex state management — possibly across distributed transactions — we decide on a third option. First, we separate integration processes by namespaces, and second we limit one scheduler to one transaction that starts only after a successful delivery (i.e., only one scheduler per process). While this prevents parallel processing, it ensures that messages are processed consistently. Parallel processing can still be achieved by increased collection sizes through partitioning. For example, when new messages msg_i, msg_{i+1} arrive in $table_s$, they will be picked up by the next scheduled transaction s_2, \dots, s_n . Within the database, the transactional processing is mandatory, making an automatic identification of the transactional boundaries important for our design.

Remark 6.7. While a formal mapping from timed db-net to database message processing is left for future work, we note that the time aspect of timed db-nets could be managed by the scheduler, which periodically checks the ages of the messages currently processed and manages the transactions accordingly.

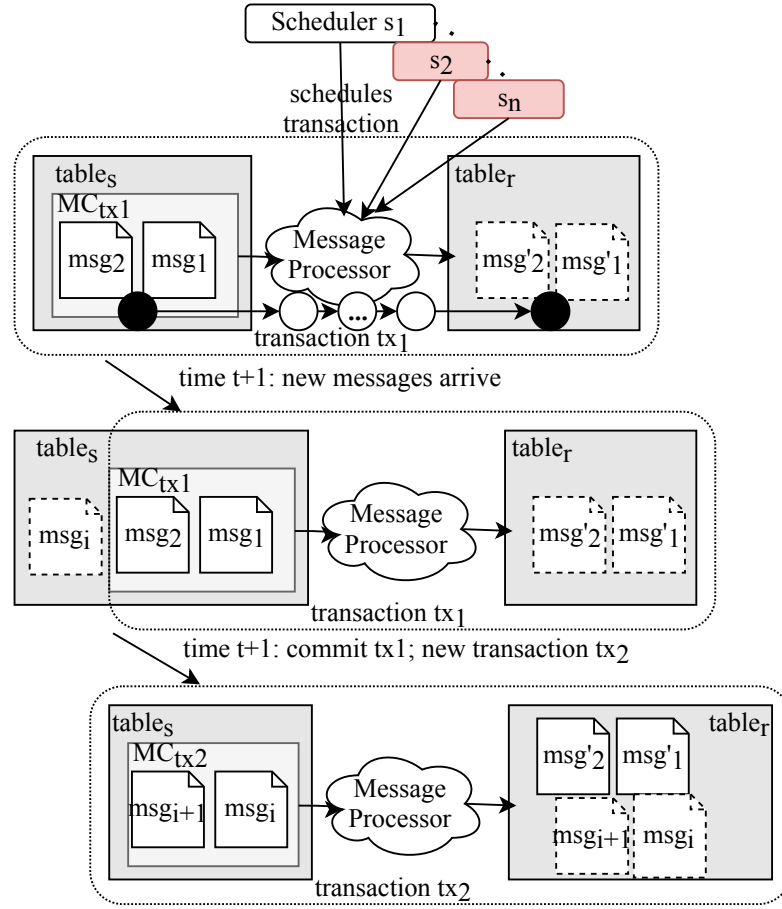


Figure 6.12: Message processing model

Database Process Compilation with Transaction Identification The message processing on a database might require more transactions than the conventional queue-based approach. For the evaluation of our approach, we built an integration to database process compiler that automatically determines transactions from a given pattern composition, leveraging meta-data about the persistent states (e.g., Aggregator, Resequencer). The corresponding *persistent-set* (p-set) algorithm is shown in Listing 6.3.

Listing 6.3: pset Algorithm

```

1 transactions =  $\emptyset$ 
2 For all nodes  $node \in PG$ 
3     if pset-match( $node$ ) then
4         transactions.add(pset-execute( $node$ ));
5     end if;
6 end for;
7 return transactions;
```

The algorithm takes the process graph PG from Section 6.1.2 with adapters and message processors as nodes N and message channels as edges. We apply ECA rules [Act96] to all nodes in PG (cf.

Listing 6.3). Thereby a rule r is defined as condition / action pair $r : \text{pset-match}(\text{IN} : \text{node} \in N, \text{OUT} : \text{Boolean})$, shown in Listing 6.4, and $r : \text{pset-execute}(\text{IN} : \text{node} \in N, \text{OUT} : \text{transactions})$ in Listing 6.5.

Listing 6.4: pset match

```

1 pset-match(node):
2     return node.state=='persistent'
3         && node.inDegree > 0;

```

Listing 6.5: pset execute

```

1 pset-execute(node):
2     t = new Transaction();
3     t.IN = ∅
4     t.OUT = {node};
5     s = new Stack<Node>();
6     for all inNode ∈ node.inNodes
7         s.push(inNode);
8     end for;
9     while (!s.empty)
10        currentNode = s.pop();
11        if currentNode.state=='persistent'
12            t.IN.add(currentNode);
13        else
14            t.IN.add(currentNode);
15            for all inNode' ∈ currentNode.inNodes
16                s.push(inNode');
17            end for;
18        end if;
19    end while;
20 return t;

```

The pset-match is applied to all nodes in PG, while pset-execute is only applied to nodes for which pset-match evaluates to true. This is the case if the processor has a state — adapters are assumed to be persistent by default, and receives data through a channel. A transaction t is defined as structure with sets $\text{IN} \subseteq N$ for inbound states and $\text{OUT} \subseteq N$ for outbound states. The pset-execute creates a transaction for each identified state and adds all states to its IN and OUT sets, in a backward direction, to cover all structural pattern categories (cf. Figures 6.10 and 6.11).

Example 6.8. The structural pattern categories from Figures 6.10 and 6.11 result in one transaction each. Furthermore, when applying the algorithm to a more complex process model in Figure 6.13(a), three transactions are identified. The scheduler executes these transactions ordered by their indices: $t_1 := (\text{IN}:se:1, s:1; \text{OUT}:s:2)$, $t_2 := (\text{IN}:s:2, s:3; \text{OUT}:ee:1, s:4)$, $t_3 := (\text{IN}:s:4, se:2, s:5; \text{OUT}:ee:2)$.

The integration process in Figure 6.13(b) denotes a case that cannot be executed on a relational database with parallel transaction scheduling, since it would result in overlapping transactions identified by the pset algorithm. Consequently, this process would be rejected and the overlap highlighted: transactions t_1 and t_3 overlap in adapter $se:2$, and t_2 and t_3 in the stateful message processor $s:2$. In a subsequent manual or semi-automatic step, the situation could be resolved by adding persistent states or converting transient states into persistent states such that the transactions do not overlap any more (e.g., making $s:4$ and $s:5$ persistent). ■

Similar to the conventional message queuing, in our approach — with ordered execution of transactions by only one scheduler — the transactions can be executed in a consistent state.

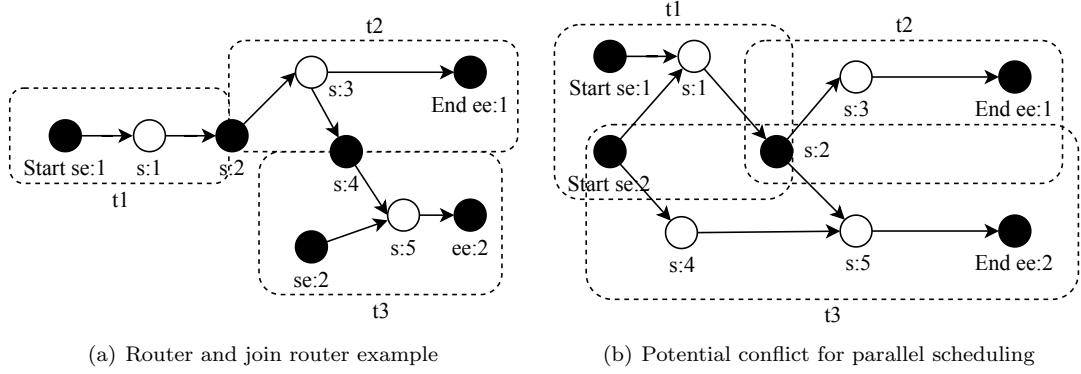


Figure 6.13: Automatic detection of transactions for more complex integration scenarios

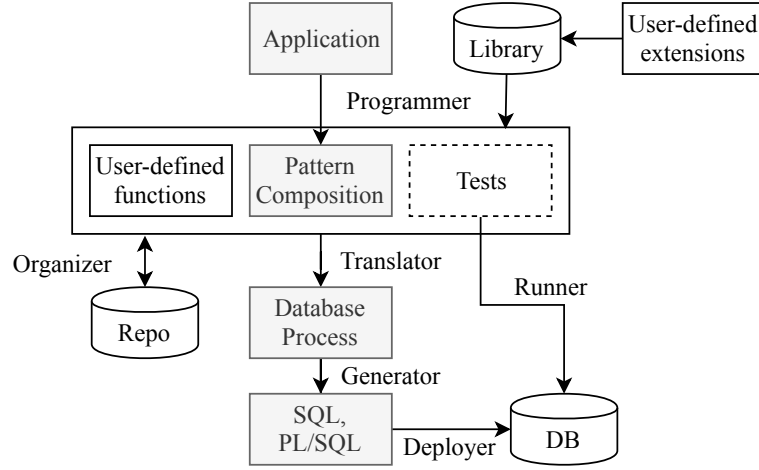


Figure 6.14: Database transition system design flow

The transactions are identified in the *translator* step in Figure 6.14, which shows the system setup and design flow from an *application* to the database, as part of the *database process* (according to the model in Figure 6.9). Then the *generator* translates the database process into *SQL*, *PL/SQL* code, using a templating mechanism (i.e., for each pattern there exists a *SQL/procedural database template*). The conditions and expressions for the specific benchmarks are provided as *user-defined functions* (from a *repository*) that are part of the translation process. If a pattern requires a conditions or an expression, its *SQL template* implements the respective interface and gets bound to the user-defined function during the generation step. The user-defined functions have to be specified in *SQL* or database procedures themselves.

6.1.3 Evaluation

We evaluate the message throughput of the defined database integration processes for distinct routing and transformation patterns and their latency in the context of the motivating scenario. The results implicitly illustrate the semantic correctness of the redefined message processing for

Table 6.1: EIPBench pattern benchmarks

Benchmark	Description
CBR-A	simple cond.: OTOTALPRICE < 100.000
CBR-B	multiple conds.: OTOTALPRICE < 100.000, ORDERPRIORITY = “3-MEDIUM”, OORDERDATE < 1970, OORDERSTATUS = “P”
MT-A	map names and filter entries according to a map. program

Content-based router (CBR), message translator (MT).

databases for the motivating scenario. The patterns are only parameterizable in terms of routing conditions and a mapping program, while the execution semantics remain unchanged. Hence, we argue that the evaluation sufficiently shows the general correctness of the Content-based Router and Message Translator on the database.

Benchmark Setup

We compare the message throughput of our approach deployed on a relational, column-store database system (referred to as *SystemX*) with the open-source integration system Apache Camel [IA10] (*Camel*) and our “table-centric” extensions of Camel with Datalog processing (*Camel-D*), as introduced in Section 5.1.2. The integration systems are running on an HP Z600 workstation, equipped with two Intel X5650 processors clocked at 2.67GHz with a 12 cores, 24GB of main memory, running a 64-bit Windows 7 SP1 and a JDK version 1.7.0 with 2GB heap space.

The relational database system is running on the same machine. The benchmark definitions are taken from EIPBench in Chapter 5, which specifies benchmark configurations derived from “real-world” integration scenarios. The message data sets are generated from the TPC-H order to customer processing, but we only generate messages based on orders. Table 6.1 summarizes the benchmark configurations from Chapter 5 for the benchmark definitions that are relevant for our evaluation (i.e., CBR-x and MT-x).

In the design flow in Figure 6.14, the benchmark data is provided by the *runner* and the benchmarks, defined as *tests*, run after the successful deployment.

Pattern Throughput

For measuring the message throughput, we selected the CBR-A router and the MT-A transformation benchmarks, due to their similarity to the operations required for the CCT scenario. We added the CBR-B benchmark to further study the impact of more routing conditions. Figure 6.15(a) shows the benchmark results for our scheduled, store and forward processing (cf. Section 6.1.2). The database integration process reads the data from a persistent state with MC_r size of 100,000 (incl. all optional tables from Figure 6.7). After successful processing, stores it into another persistent state in one transaction. We refer to Section 5.1.2 for the Camel and Camel-D “micro-batching” extension, to which the SystemX results are compared. The baseline throughput for Camel routes and database integration processes (i.e., no intermediate processors) is added for comparison. For Camel and Camel-D, the baseline is the same, due to sharing the same pipeline engine.

The results show a clear edge for the more data-centric, relational Camel-D processing for more complex routing and transformation patterns over the conventional Camel processing, while having similar results for the simpler CBR-A case. However, database processing outperforms both in all benchmarks. This is mainly due to the more efficient set-oriented data processing and the parallelization of read-only operations (e.g., used in the router). More routing conditions (cf.

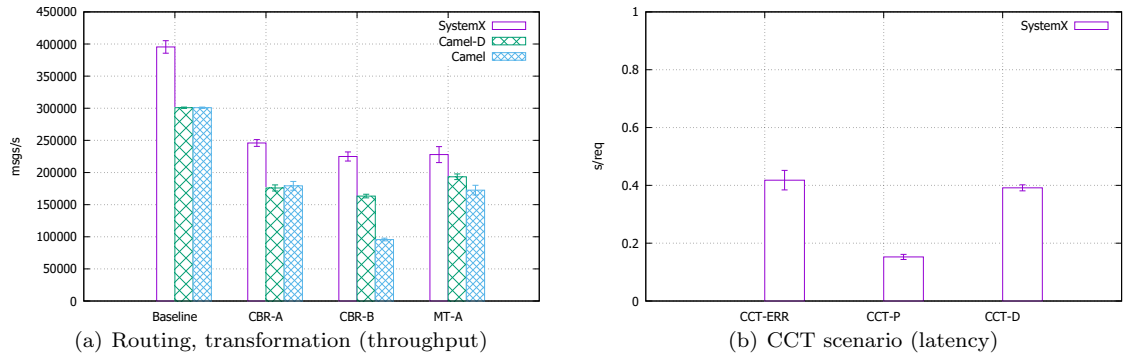


Figure 6.15: Pattern throughput benchmark and process latencies for CCT scenario

CBR-B) reduce the throughput for all three systems, however, have a lower impact on the more data-centric approaches (i.e., database processes, Camel-D).

Conclusions. (1) The batched, parallel processing of the database transition system results in high message throughput; (2) higher micro-batch sizes increase the message throughput, but decrease the latency, and thus lead to a latency vs. throughput trade-off.

Pattern Composition: Connected Car Telemetry (CCT)

The data sent from the vehicles in the CCT example in Figure 6.1 consists of approximately 304B error code JSON messages (stored normalized in the column store) with fields like "Trouble.Codes":"MIL is OFF0" and approximately 762B car telemetry data with fields like "Vehicle.Speed":"0km/h" and "Engine.Load":"18,8%". For the evaluation of the CCT scenario, we assume that all the data is in one (federated) database transition system. Figure 6.15(b) shows that the latency depends on the path the messages take through the integration process. The routing of parking car data (*CCT-P*) shows the lowest latency, since parking cars are filtered early in the process. However, the telemetry data of driving cars (*CCT-D*) pass the filter and are then further processed in the same way as the error codes (*CCT-ERR*). Consequently, the filter has no significant impact on the latency. Subsequently, the CCT-D and CCT-ERR data is enriched by the car owner's master data by lookup of the `Vehicle.ID.Number` and transformed into the receiver's format, showing a more significant reduction of the latency compared to the filter. This is due to the increasing amount of data and the more data-intensive operations.

While the performance gains are notable and the realized patterns are correct, the current systems are lacking expressiveness with respect to non-functional aspects like exception handling and security, which would require extensions for integration scenarios in general. For example, *try-catch* extensions⁴ allow for the representation of the Local Catch pattern (see Chapter 2), however, do not allow to preserve the error context as well as already processed intermediate results in case of an error. Moreover, dealing with security relevant content (e.g., encrypted content, signed content) is currently only possible for encrypting data on transport and row or column level (e.g., cf. SAP HANA database [FCP⁺12, FML⁺12]), but for example not for verifying message signatures or signing data in messages.

⁴Microsoft SQL-Server – TRY...CATCH (Transact-SQL), visited 5/2019: <https://docs.microsoft.com/en-us/sql/t-sql/language-elements/try-catch-transact-sql?view=sql-server-2017>

Conclusions. (3) The processing latency of the procedural constructs is high, but sufficient for our scenario; (4) the expressiveness is not sufficient for all non-functional EAI aspects.

6.1.4 Conclusions

To explore the potential of pattern composition vectorization, we follow the approach in Figure 6.3(b). We define a representation (cf. research question *DB1*: “How can EAI semantics be represented in database systems?”) and processing semantics for application integration on relational databases (cf. *DB2*: “How can EAI processing semantics be compatibly adapted to databases?”). We redefine message processing patterns that are relevant for the motivational CCT scenario (e.g., implicitly: Event Message; Point-to-Point Channel, Channel Adapter; explicitly: Document Message, Datatype Channel, Content-based Router, Message Filter, Message Translator, Message Store, Canonical Data Model), discuss new pattern semantics for more efficient high volume message processing and evaluate the approach for message throughput of distinct patterns as well as composition on our connected car telemetry scenario. The integration pattern compositions are compiled to the DTS by automatically identifying transactional contexts.

While the evaluation of the DTS showed promising results for acceleration of the message throughput of more data-centric approaches (cf *DB3*: “Can databases accelerate message processing?”; cf. conclusions (1,3)), new patterns for non-functional aspects like exception handling and message privacy are currently only partially supported by database systems (cf. conclusion (4)). Our approach allows for the compilation of pattern compositions to other database systems, but makes their configuration (e.g., through user-defined conditions or expressions) a task for database experts (i.e., not suitable for an integration developer or even business user), due to differences in SQL and procedural constructs. Open research questions target the optimal message collection size for scenario workloads and parallel transaction processing that preserves the integration semantics (cf. conclusion (2)). Moreover, the applicability to NoSQL databases is of interest.

We conclude that a DTS allows for scalable, high-throughput message processing for big (data) volume scenarios. While there are still some conceptual gaps (e.g., see conclusion (4)), we conjecture that data-centric applications can be implemented on (big) data processing platforms like BDMS.

6.2 Specialization: Hardware-Accelerated Pattern Solutions

Today, challenges like the increasing gap between memory access time and processor speed (the memory wall; it takes several hundred cycles to access off-chip memory) and the fact that the bus between main memory and CPU is shared between program instructions and workload data (the Von Neumann bottleneck) decrease the processing velocity. While this can be partially solved by memory locality, caches as well as out of order execution, branch prediction, pipelining, and cache hierarchies [Har97], the latter requires many of the available transistors to implement all sorts of acceleration techniques to nonetheless improve performance. Furthermore, Hill et al. [HM08] observed that Amdahl’s law applies to homogeneous multi-core systems and Esmailzadeh et al. [EBA⁺11] showed that underutilization of transistors can be due to power consumption constraints and/or inefficient parallel hardware architectures conflicting with Amdahl’s law, called dark silicon, which can be up to 50-80% of the transistors.

A move towards heterogeneous architectures, e.g., where tasks are off-loaded to customized hardware, can improve the velocity and save energy [BC11]. That means mapping customized hardware (e.g., ASICs) to a given task instead of mapping the task at hand to a fixed general-purpose hardware architecture. Since customized hardware has orders of magnitude lower power consumption (compared to general purpose CPUs and GPUs) and avoids the von Neumann

bottleneck, lower clock frequencies are sufficient to solve the task (fulfill performance requirements). Between extremes general purpose processors and ASICs, reprogrammable hardware combines performance and lower consumption of ASICs with the flexibility of general-purpose hardware. Currently, the most advanced reprogrammable hardware are *Field Programmable Gate Arrays* (FPGAs), whose designs can be implemented as ASICs. In the past years FPGAs are increasingly used for data processing (e.g., databases [MT10], complex event processing [WTA10], stream processing [MTA09b], and deep learning [ORK⁺15]). Whereas FPGAs are still considered exotic today, cloud computing could make FPGAs a mainstream data processing technology [MS11]. A major cost factor for cloud providers is the power consumption of their data centers. Thus, any technology that can deliver the same performance and flexibility with a lower energy footprint is very attractive for cloud computing. In addition, projects like Microsoft’s project catapult show that the big cloud providers make progress on the challenges (e.g., FPGA virtualization [PCC⁺14, CCP⁺16]) and provide new solutions (e.g., accelerated networking [FPM⁺18]).

In the presence of integration scenarios like the CCT scenario in Figure 6.1, the near real-time processing of complex events or patterns is considered a challenging requirement [CDN11]. For such scenarios, FPGAs [SG08] promise lower latency, higher throughput and lower energy consumption than comparable solutions in software and on general purpose CPUs. With multi-chip processors delivered with FPGAs on the chip (e.g., by Intel), FPGAs might become far more widely used than today, and thus allows them to be included in cloud-scale deployments [CSZ⁺14]. However, when considering the message throughput results from Chapter 5, it becomes obvious that a single integration system instance would not suffice to process the load of messages generated in the connected car scenario using typical integration patterns. Common software solutions include optimization strategies as discussed in Chapter 4, which however lead to an even higher energy consumption and still suffer from the challenges on current von Neumann architectures [Rit17b]. Moreover, the integration patterns are more complex than message queuing for reliable queues and topics [EHI10, Sol16] or event and stream processing for continuous queries and alerts [MTA09a, TW13], for which FPGAs were employed before. At the same time the throughput demands are beyond the ones for query processing using FPGAs [MT10] which are bound by disk access. Figure 6.16 shows how database query processing puts programmable hardware in the form of FPGAs into the data path of the systems to evolve them toward heterogeneous many-core systems.

We envision application integration processing logic on the FPGA, which we put on the network path between applications, devices and databases (e.g., using TCP/IP stack on FPGAs [SAB⁺15]). The extensive body of research and industrial work on FPGA-based hardware event stream and data processing reports on competitive results (e.g., reconfigurable logic, low-latency) due to parallel streaming [GNVV04] through hardware characteristics like parallel stream evaluation and asynchronous circuits, as well as reduced power consumption compared to modern general-purpose CPUs or GPUs.

Example 6.9. With dataflow processing capabilities, the CCT scenario could be significantly changed as shown in Figure 6.17. Through fast processing, close to the network, devices could directly send the *Vehicle Data* to the network attached *CCT Integration*, for which we assume all required master data accessible on-chip for fast access. Instead of batch processing, the data is streamed through the pattern composition synthesized to the hardware. Then the resulting data is sent to the *CCT Analytics* application.

However, hardware is not the “silver bullet” [MT10], and the efficient usage of FPGAs involves non-trivial aspects and difficult challenges such as making the right design decisions for the computation model, dealing with low-frequency clocks, balancing the trade-off in usage between synchronous and asynchronous circuits, and resource limitation (cf. [Rit17b]). In addition, it

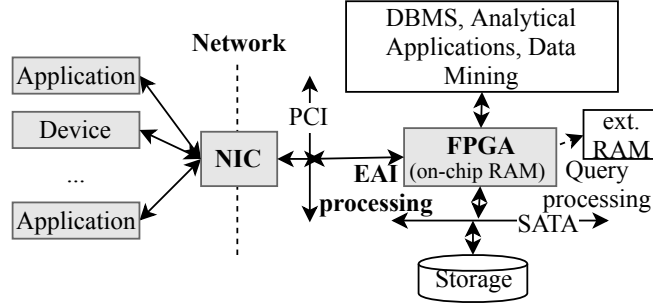


Figure 6.16: FPGA hardware for EAI processing

remains unclear, how the potential of FPGAs can be efficiently exploited for EAI. Important questions about the EAI building blocks and semantics, represented by the EIP, and their usage on FPGA hardware have to be answered. For instance, the integration patterns are defined for asynchronous, non-streaming cases, while FPGA hardware supports streaming very well. Hence, we answer the following research sub-questions of RQ3-1 (*HW_x*):

- (HW1): “How can the EAI building blocks and semantics be (efficiently) represented on reprogrammable hardware?”
- (HW2): “How can textual message protocols and user conditions be represented on hardware (i.e., predicates and expressions)?”
- (HW3): “How does the application of integration patterns on FPGA influence their semantics and implementations?”
- (HW4): “Could FPGAs accelerate the message routing even more than transformation?”

While question HW1 targets the feasibility and an efficient representation of the integration patterns on hardware (i.e., including message, pipeline, routing, transformation), HW2 is about the variety in message protocols, which requires support for non-trivial message formats and operations (e.g., hierarchical formats like JSON with JSONPath predicates). We consider the hypothesis that a hardware approach would favour message routing even more. Since the integration patterns are not defined with a focus on stream processing, we consider question HW3. The work on database and event processing (e.g., [MT10, MTA09b, WTA10]), where hardware was successfully applied for velocity (cf. challenge C5 in Section 1.2), discusses some data aspects relevant to application integration, which leads to question HW4.

To answer these questions we introduce FPGA fundamentals in Section 6.2.1 for a better understanding of the subsequently defined dataflow integration system on reprogrammable hardware. Therefore — taking the existing results and conceptual extensions from related domains into account — we study the feasibility, advantages and limitations of (composed) integration patterns on hardware. In particular, we define the streaming semantics of selected integration patterns (i.e., including message, pipeline), because our work fully focuses on streaming (i.e., no message off-loading into RAM), and analyze how they can be deployed on FPGAs in Section 6.2.2 (cf. questions HW1, HW3). This is not trivial because of the illustrated trade-offs between computation model, throughput, resource limitations, parallelization and diverse integration semantics. We categorize the streaming semantics of the EIP into three template classes based on their interaction with user-defined conditions and expressions: *Expression Template* (ET), *Predicate Template* (PT) and *No User Template* (NUT). Consequently only these classes have to be synthesized to hardware. For the message protocol variety and user

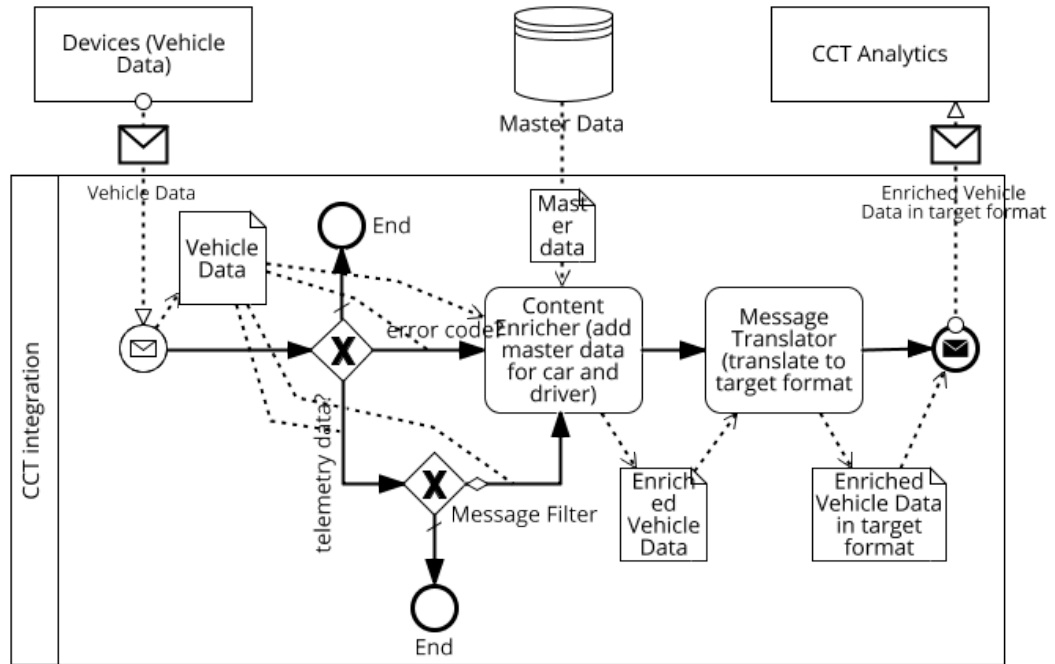


Figure 6.17: CCT scenario with focus on the integration processing

conditions (i.e., predicate, expression), we define a state machine parsing and matching approach for hierarchical message protocols in Section 6.2.3 (cf. question HW2), and we evaluate the solution and discuss optimizations in Section 6.2.4 (cf. question HW4).

The resulting application integration system on a dataflow architecture does not only allow to show the immense potential for velocity of message processing with reprogrammable hardware close to the network, but it also proposes a novel “responsible”, sustainable EAI system design with an improved message per watt energy ratio.

Parts of this section have appeared in the proceedings of DEBS 2017 [RDMRM17] and ACTIVE@Middleware 2017 [Rit17b].

6.2.1 Field-Programmable Gate Arrays

Following *programmable logic devices* (PLDs), developed by companies like Motorola, Texas Instruments and IGM in the 1970s, *Field Programmable Gate Arrays* (FPGAs) were developed at Xilinx in the 1980s. They implement a dataflow architecture [Cul86] that basically consists of wires, gates and registers. The key concepts of (re)programmable hardware are briefly introduced in a bottom-up approach based on Teubner and Woods [TW13] (if not stated otherwise) by explaining the basic building blocks of FPGAs, and then we gradually zoom out and show how the various components are combined and interconnected. For the programming of FPGAs we refer to the literature (e.g., [Sha98, TW13]).

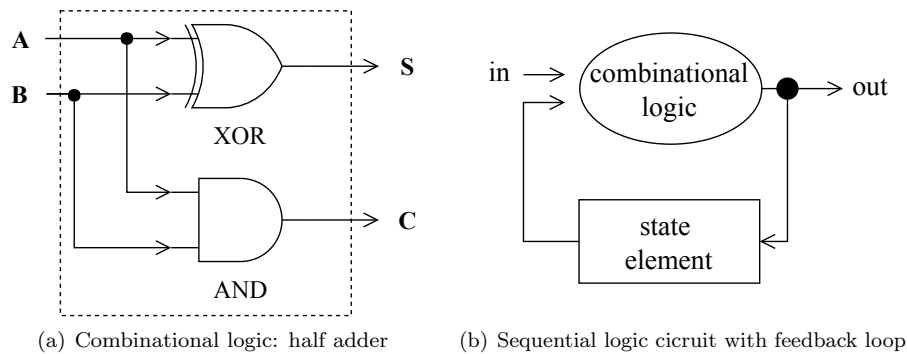


Figure 6.18: Combinational and asynchronous sequential logic (e.g., similar to [TW13])

Combinational logic

Any hardware circuit consists of basic *logic gates* (e.g., AND, OR; binary states: low, high depending on the voltage level), which can be combined as a form of combinational logic elements by wiring their input and output ports.

Example 6.10. Figure 6.18(a) shows a half adder for the addition of two one-bit numbers, whose output is written to port *S*. With *A* and *B* both set to one an overflow is produced that is captured by the AND gate and reported to output port *C* (carry bit). ■

The *combination logic* is purely driven by the input data (i.e., no clock). Due to physical effects (e.g., signal propagation times), a logic gate has a fixed propagation delay (from input to output). When composing the logic gates, the speed of a sequential circuit is the sum of the single propagation delays along the longest path.

Synchronous Sequential Logic

While combinational logic can be extended to *sequential logic* by adding state (memory), it still is entirely driven by the input, and no form of synchronization exists. Figure 6.18(b) denotes a function in sequential logic, which depends on its present and past inputs. The resulting type of circuitry is called asynchronous sequential logic, whose speed is only limited by the propagation delays. However, the lack of synchronization results in effects like race conditions, which are difficult to deal with.

Therefore, *synchronous sequential logic* adds synchronized clocks, which synchronizes all memory elements by a clock signal (i.e., a clock *clk* that is an electronic oscillator: logic high, low). The clock frequency determines the length of the clock period for all combinational logic elements, which have to be finished at the end of the period. The result is a predictable and reliable behavior, but the clock frequency is determined by the critical path in the circuit (the longest combinational path between states).

Basic State An already more sophisticated memory element for representing the state is a flip-flop (FF), which only stores the input from a dedicated port. The FF's data and clock ports can usually be bypassed for set or reset logic.

Logic Gate Representation The logic gates in FPGAs are “simulated” using a generic element called a look-up table (LUT). In contrast to ASICs, LUTs can be (re)programmed after manufacturing, which ensures the (re)programmability property of FPGAs. An n -input LUT implements an arbitrary Boolean-valued function with up to n Boolean arguments (e.g., two-input AND or OR gate).

These functions are stored in SRAM, which means that programming a LUT can be facilitated by updating the SRAM cells of a given LUT. An n -input LUT requires 2^n bits of SRAM to store the lookup table and a $2^n : 1$ multiplexer to read out a given configuration bit. The inputs determine which SRAM bit is forwarded to the output of the LUT. While a LUT can be usually read in one clock-cycle, the write requires 2^n cycles. This is due to a design decision under the trade-off between write performance and simpler design (chip space consumption). LUTs cannot only be used to simulate gates, but also be used as distributed RAM (e.g., to implement a FIFO queue), however, we will not use this in our design.

FPGA Architecture

The grouping and embedding of LUTs into a programmable logic component denotes an *elementary logic unit* (vendor-specific, e.g., called slice in Xilinx and ALMs in Intel FPGAs). However, common elementary logic units include LUTs, a (proportional) number of one-bit FF registers, arithmetic / carry logic and multiplexers. To store the result of the table look-up, the LUTs are paired with FFs (classical for input LUTs and two FFs). The arithmetic / carry logic are fast wires between LUTs and circuitry, e.g., carry chains (combining LUTs to implement logic). This facilitates a pipelined circuit design, where signals propagate through large parts of the FPGA while the high clock frequency is maintained. The multiplexers determine whether a flip-flop is used or by-passed, which can be (re)programmed through SRAM.

Routing Architecture

The wiring of neighboring elementary logic units is facilitated through direct wires (e.g., carry chains). In this way, modern FPGAs provide configurable resources that are sufficient to host an entire system on a chip. The required flexible communication mechanism is known as interconnect.

Logic Islands (LIs) A number of grouped elementary logic units, called logic islands (LIs), corresponds to often used term configurable logic blocks (CLBs). More generally, the communication between logic islands is facilitated by a *switch matrix* connected to the interconnect. The arrangement of logic islands is shown as two-dimensional array on the FPGA in Figure 6.19(b). The flexible interconnects allow for arbitrary communication between the logic islands as well with the periphery through special I/O blocks (IOBs).

Interconnect For the communication via a switch matrix (arbitrary communication patterns) between logic islands, the interconnect denotes a configurable routing architecture. This way the LIs access special, peripheral I/O blocks to communicate with the outside world through I/O standards like single-ended PCI and differential PCI-express, SATA, Ethernet. Since the interconnections on a circuit can be complex, the “place and route” part of the FPGA’s flow design (part of the development process [TW13]) is time-consuming. Furthermore, the circuit performance is limited by connections between internal components rather than their speed, which is known as the *interconnect bottleneck* (cf. Abadi [AFG⁺05]), which has to be considered during the design.

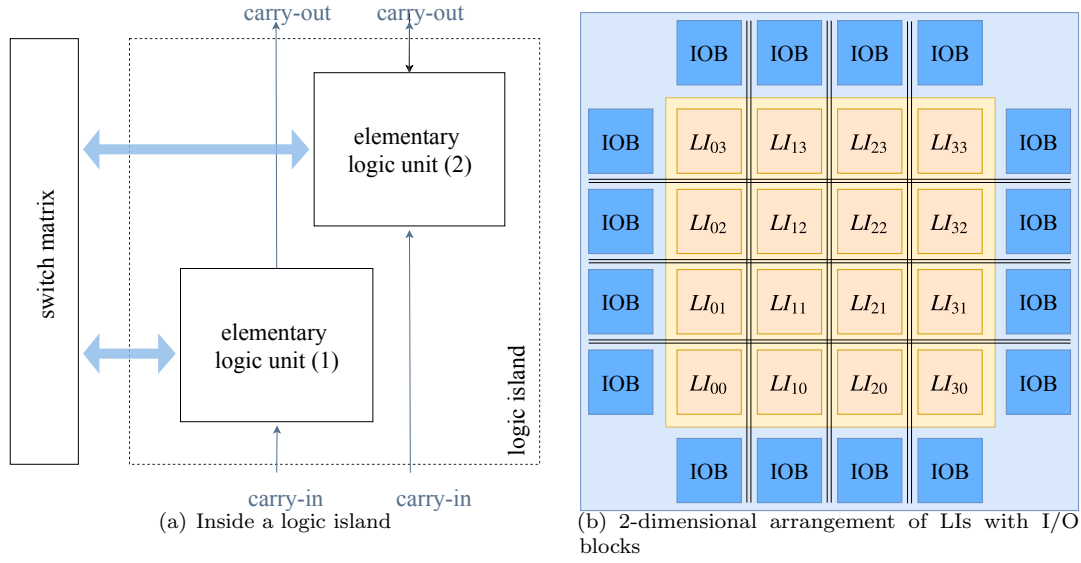


Figure 6.19: Logic islands and with I/O blocks (e.g., similar to [TW13])

System on a Chip For the design of a more complex logic towards a *System on a Chip* (SoC) further components are required. This includes reusable logic either as so called *soft intellectual property* (soft IP), which uses the existing resources to build composed building blocks (which we will do on a pattern level), or even dedicated silicon on the chip, called hard IP. The soft IP is usually provided by the FPGA vendor or contributors like partners and recently started to be provided in market places mostly organized by the vendors.

Example 6.11. Serial-to-parallel converters and 8b / 10b encoders / decoders (i.e., conversion of eight-bit data byte to a 10-bit transmission character and vice versa) are available, which are specifically useful for the implementation of communication protocols (the physical layer) based on available 10Gbit ethernet soft IP. ■

The hard IPs usually come from the FPGA vendors and comprise for example dedicated on-chip block RAM (BRAM), larger than distributed LUT memory, or digital signal processing (DSP) units, i.e., dedicated, customizable hardware multipliers and adders (e.g., for digital filtering, Fourier analysis). Combined, these components denote a simple FPGA architecture, as shown in Figure 6.20. The BRAM is distributed as stripes over the chip to allow the logic islands for a fast access. A small number of DSP units is added in a similar way. With the growing number of components, already provided in online market places organized by the vendors, FPGAs evolve from a “bag of gates” to a “bag of computer parts” [HGV+08] or SoCs.

6.2.2 Dataflow Integration System on FPGAs: Patterns to Circuits

As motivated in Figure 6.16, the complete EAI processing logic shall be put onto the FPGA on the network path between applications and devices. Therefore, we map the integration semantics in the form of integration patterns to hardware concepts by re-defining them for synchronous streaming with flow control (similar to [Cas05]) and classifying the patterns according to their characteristics to three templates that are then synthesized to the hardware.

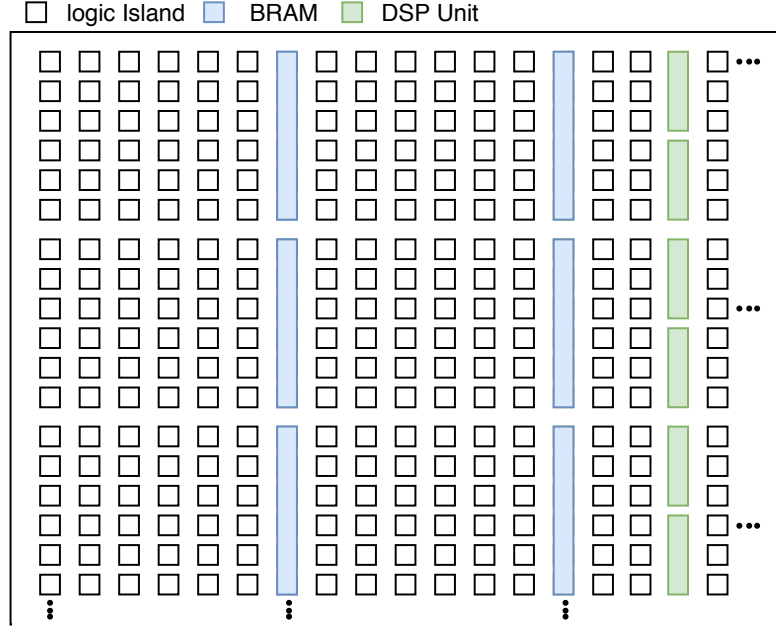


Figure 6.20: FPGA layout with interspersed BRAM blocks and DSP units (e.g., similar to [TW13])

Basic Integration Semantics

The basic integration semantics are described by a message, a message channel, at least one protocol adapter, and an ordered set of transformation or routing patterns that process the message as summarized in Figure 6.21 and subsequently discussed.

Messages on Hardware A message consists of a unique identifier, for identifying the message and enabling provenance along the different processing steps, a set of name / value pair header entries containing meta-data about the message and the message body, i.e., denoting the actual data transferred.

For the processing on an FPGA, a message is defined as a stream of bytes, which gets meta-data assigned on entering the FPGA via the network interface. In particular, we assign a unique message identifier and the length of the message.

Message Channels on Hardware The message channels decouple sending and receiving endpoints or processors and denote the communication between them. Thereby, the sending endpoint writes data to the channel, while the receiving endpoint reads the data for further processing. Our message channel definition on hardware is depicted in Figure 6.21. We use hardware signals and data lines to represent the control and data flow through a message channel. The channels contain a unique identifier as `id`, the message length as `length`, and the body as `data` of 8 bit chunks from the previously defined message over the data line (`data(0..7)`). To indicate that a message is sent over the channel, we added a message signal as `message`, which is set to one (i.e., `high`), when one message is sent — even if there is currently no valid data on the data line. The `message` signal is zero (i.e., `low`) only between messages (i.e., when the channel is ready to receive another message). For the transport of the data to the subsequent processor we

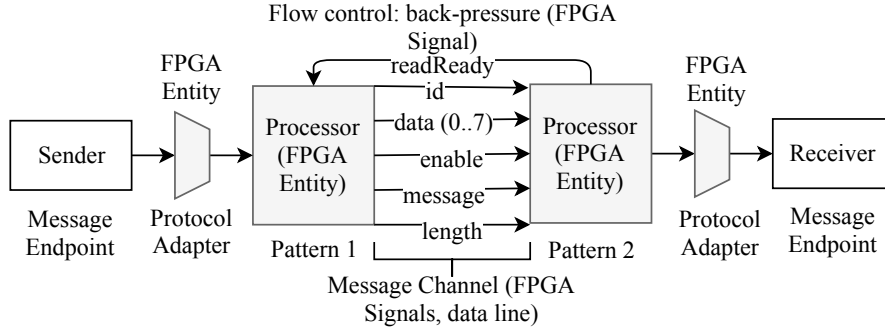


Figure 6.21: Basic integration aspects on hardware

define an enable signal as **enable**, which is **high**, when valid data is on the data line and **low**, when there is no valid data on the data line. The **id** and **length** are separate lines, which are constant, when the message line is high.

The FPGA hardware is able to stream massively parallel using pipeline processing. However, for efficient processing, the hardware is limited to the resources on-chip (e.g., BRAM) and on-board (e.g., RAM). Our basic integration aspects are represented with on-chip resources to avoid latency and throughput penalties for the on-board resource access. However, we expect that in the future FPGAs can interact with less overhead with off-chip resources like DRAM or general-purpose CPUs. This will offer more freedom to use these off-chip resources, e.g. to buffer large messages.

Flow Control on Hardware Message Channels The basic capability of an integration system to protect overload situations and data corruption is flow control. One technique used in connection-oriented wire protocols like TCP is *back-pressure*. Back-pressure allows the message processors (e.g., routing and transformation patterns) under high load to notify the sending operation or remote endpoint (e.g., via TCP back-pressure) about its situation. For instance, for TCP remote endpoints this could lead to the rejection of new connections.

On the FPGA, we define flow control similar to [Cas05], which is exclusively used there for the synchronous communication between remote endpoints. For the back-pressure between message processors (i.e., no TCP support), we cannot reject messages atomically, because the stream might already be processed partially. Therefore, we decided on an approach with small FIFO queues in each processor that are used to buffer message data that cannot be immediately processed by the subsequent processor and thus ensure that no message data is lost. The receiving processor signals this by setting its **readReady** to low (cf. Figure 6.21). The FIFO queues can be represented on hardware using flip-flops (FF), Block RAM (BRAM) or built-in FIFOs. Since FFs can only store one bit at a time and are very important for the logic of message processors, we chose BRAM. Although BRAM is a limited resources as well, it can be more easily extended by on-board DRAM to buffer larger messages. If the queue limit is exceeded, and the successor processor is not ready yet (i.e., **readReady** low), the current message processor notifies its sender by setting its **readReady** to low.

Streaming Integration Patterns

Since the EIPs are not defined for stream processing (cf. Chapter 2), we define streaming semantics for the practically most relevant routing and transformation patterns and map them to circuits.

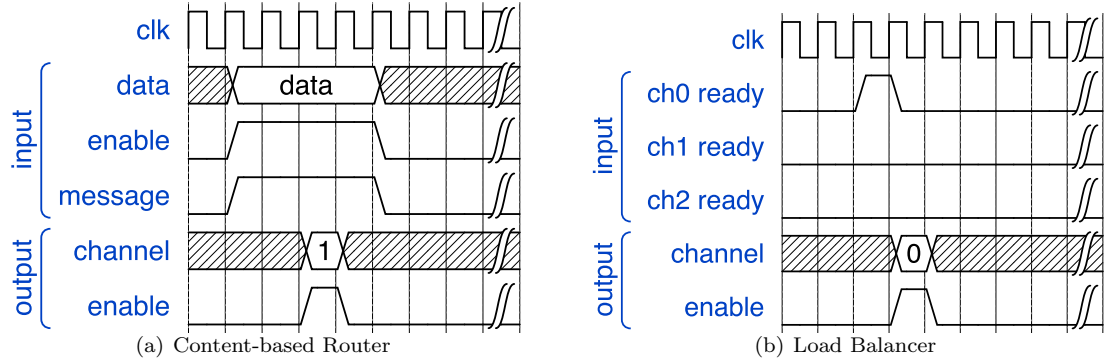


Figure 6.22: Router and load balancer patterns

Routing Patterns

The routing patterns are used to decouple individual processing steps so that messages can be passed to different filters depending on a set of conditions [HW04]. We selected the most relevant routing patterns from Chapter 5: Content-based Router, Message Filter and Splitter, and add the Aggregator, Load Balancer and Join Router used in the example scenario and in our evaluation.

Content-based Router The Content-based Router and the Message Filter are semantically similar. The filter is a special case of the router due to its channel cardinality of 1:1, while the router has 1:n. Both have a message cardinality of 1:1, are read-only (i.e., non-altering message access) and non-message generating (i.e., passing the original message).

Streaming Semantics. The router selects a message channel based on a condition that in worst case might have to fully read the message (i.e., requires buffering). In our approach the message corresponds to a data-based window similar to [GMM⁺16]. Alternatively, the message could be passed further into all leaving channels in parallel and filtered out later at a synchronization point. While the latter claims non-buffered streaming semantics the synchronization points cannot be set arbitrarily, which could lead to the same semantics as the message-window semantics in worst case. Hence, we use data-based windows as the basis for our work.

On Circuit. Since, the leaving channel is selected based on a condition evaluated on the stream, it specifies a mapping from a message to a channel. Figure 6.22(a) illustrates the semantics of our router design as a waveform diagram, which shows high and low circuit settings for the different signals and data lines required for the pattern. The input denotes the message from the previous pattern. The output is the response from the user code. The clock cycles are denoted by `clk`. The `channel` is represented by an integer identifier. When a message enters the router (i.e., `message`, `data` high) and the data is valid (i.e., `enabled` high), then the condition (i.e., user code) is evaluated and the identifier of the selected channel is set together with the data valid signal `enabled`. According to the channel identifier the message will be routed.

Load Balancer The Load Balancer pattern – not in the original EIPs – delegates a message to exactly one of several message channels using a load balancing strategy (e.g., uniform distribution). As the Content-based Router, it is read-only and non-message generating.

Streaming Semantics. For the purpose of this work, the Load Balancer is already suitable for

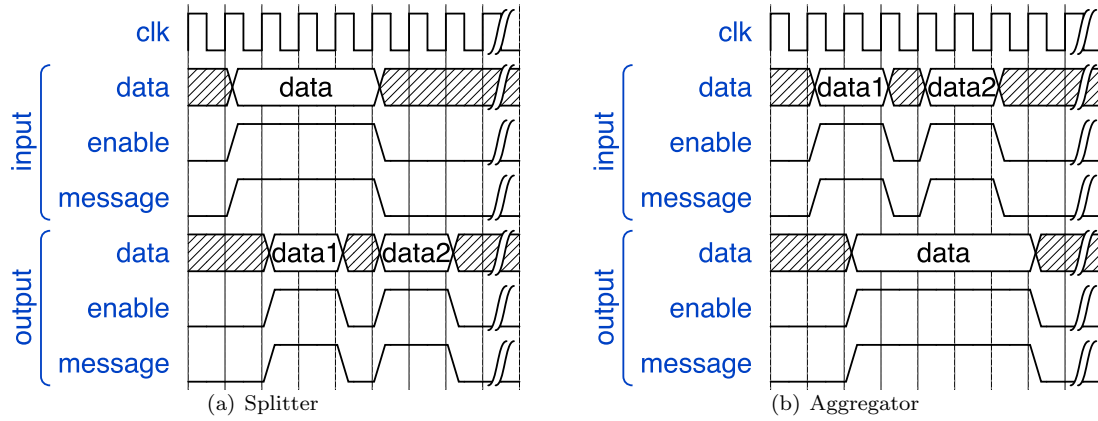


Figure 6.23: Splitter and aggregator patterns

streaming since it does not define any conditions on the message.

On Circuit. Similar to the Content-based Router, the Load Balancer maps from a message to a message channel. This time, the channel identifier is determined by a load balancing logic, configured through user code. Instead of a load distribution, we use the `readReady` signal used for back-pressure to determine which channel is currently available. The first channel available is selected as receiving channel. This semantics is shown in Figure 6.22(b) as `ch0 ready`. Consequently the output channel is written and the data valid signal is set. The message will be routed accordingly.

Splitter The Splitter pattern allows processing messages containing multiple elements, each of which may have to be processed in a different way. Therefore the inbound message is split into a number of smaller messages according to a user-defined expression (i.e., message cardinality $1:n$) with a channel cardinality of $1:1$. While the Splitter is message generating, it does not add new data to the n outbound messages.

Streaming Semantics. The Splitter splits parts of a message into smaller parts similar to single elements of an iterable. There are Splitter configurations with data structures of the form: head, iterable, tail. For each entry in an iterable a new message is created by starting with the common head entries, one entry from the iterable and the common tail entries. In these cases, the head has to be remembered and added before each element from the iterable. However, in case there is an element after the iterable, called tail, that has to be added, the streaming is limited. The tail is unknown to the Splitter until the end of the message, which means that the first new message would have to be buffered until the tail arrives. Similar to the router, a synchronization point could be used to add the tail part to each of the smaller messages. For the same reason as for the router, we use the buffered streaming option.

On Circuit. The Splitter maps one message to multiple messages. Figure 6.23(a) illustrates the execution semantics for input and output. When the messages arrive and the data is valid, the user-defined split expression is executed, which leads to several messages. Thereby the Splitter inserts a clock cycle, where the message signal is set to low and no data is sent inbetween the split messages.

Aggregator The Aggregator pattern combines a number of messages to one based on a time or number of messages based completion condition. Hence it has a message cardinality of $n:1$, a channel cardinality of $1:1$, and is message generating, however, only combines data from the inbound messages to the one outbound message.

Streaming Semantics. By definition, an Aggregator has to wait until the last message arrives. Only then can the new message be processed. Hence, the Aggregator shows buffered streaming semantics on a streaming window, which is defined by the completion condition of the Aggregator.

On Circuit. The Aggregator maps from multiple messages to one message. Depending on the messages, the Aggregator might only close the gaps between multiple messages or combine them in a new way (e.g., forwarding the common header of the messages and then the body of each one). In Figure 6.23(b) data arriving in separate messages (i.e., **message** high) are combined to one message by setting the message signal to high as long as there are messages arriving that shall be combined to the outbound message.

Join Router The Join Router is a new structural pattern that is needed to combine several control flows to one. This leads to a channel cardinality of $n:1$ and a message cardinality of $1:1$. Note that data flows are combined using an Aggregator.

Streaming Semantics. The router is already suitable for streaming, since it does not define any conditions or expressions on the message, but combines several streams to one.

On Circuit. The Join Router maps from channel to channel, however, without any additional logic (e.g., as in the Load Balancer case). It simply checks, whether there are messages on the inbound channels and whether the outbound channel is free (i.e., **readReady** high).

Message Transformation Patterns

The transformation patterns are used to translate the message content to make it understandable for its receiving message processor or endpoint. We selected the most relevant transformation patterns: Content Enricher and Message Translator, identified by a study on integration scenarios in Chapter 5. All transformation patterns have a channel cardinality and a message cardinality of $1:1$. They do not generate new messages, but modify the current one.

Content Enricher The Content Enricher pattern adds content to an existing message, if the message originator does not provide all the required data items. The enrichment can be done (a) statically, (b) from the message itself or (c) from an external source. In this work, we consider (a) and (b), however, the external data (c) could be provided on the on-board RAM.

Streaming Semantics. The current enricher semantics for (a) and (b) allow to fully stream this operation.

On Circuit. The enricher maps one message to another, while inserting data into the inbound message. Figure 6.24(a) shows the processing semantics for one message with **data1**, to which additional data is added by a user-defined expression as **data2**. Thereby, the message and the data valid signals are set to high.

Message Translator The Message Translator pattern converts the structure of the inbound message into one understood by the receiver. This includes filtering content, which covers the content filter pattern.

Streaming Semantics. The current Message Translator definition covers streaming for simple cases (e.g., one to one assignments, data type operations). In addition, for many to one field translations, parts of the data have to be buffered for later lookup and assignment.

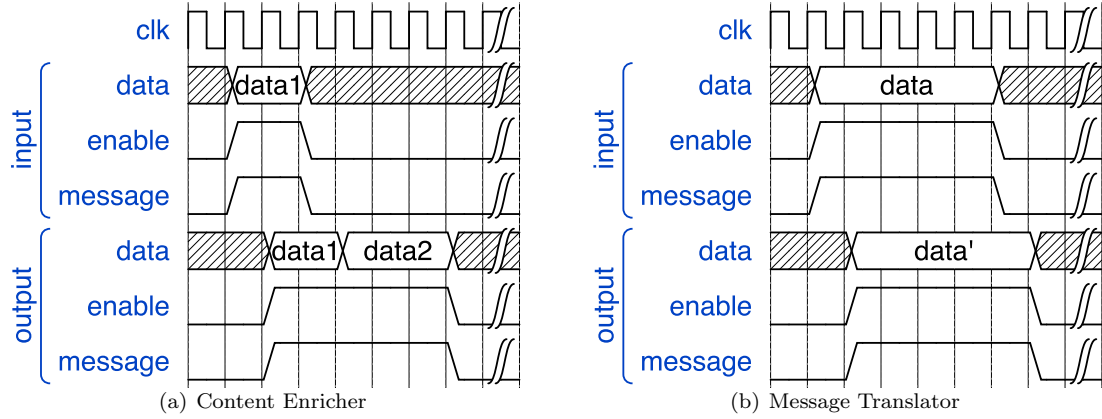


Figure 6.24: Translator and content enricher patterns

On Circuit. The translator maps one message to another one, while reorganizing the data in the inbound message using user-defined expressions. Figure 6.24(b) shows the behavior for the interaction with the user code, which returns the modified message content as `data'`. The message and the data valid signals are set to high.

Pattern Templates

Related approaches show that the hardware design is crucial for the performance of the resulting hardware scenarios [MT10]. To achieve good utilization of the FPGA hardware and high throughput we exploit commonalities between the patterns discussed in Section 6.2.2. Therefore we arrange them into three classes of behavior, which we call templates (similar to SQL-Query constructs like project and join in [MTA09b]). Unlike the original routing and transformation categories from [HW04], this classification is based on implementation criteria (i.e., when mapping to FPGAs).

From Patterns to Hardware Templates We build the categories for the classification based on the interaction with the user-defined conditions: predicates or expressions.

Expression Template (ET). The first template contains all patterns that conduct a “message to message” mapping. They mostly execute more complex expressions, which are provided by the user, while working directly with the data line. This applies to the Splitter, Aggregator, Content Enricher and Message Translator patterns.

Predicate Template (PT). The patterns that conduct a “message to channel” mapping, mostly execute simpler conditions like predicates. They set the `message` and `channel` signals. Candidates are the Content-based Router, the Message Filter, and the Load Balancer patterns.

No User Template (NUT). There is only one pattern in our selection that does not fit into the previous templates (and maybe not the only one). The Join Router conducts a “channel to channel” mapping and does not evaluate any user-defined conditions.

Putting it all together With the basic integration semantics (incl. flow control) and the patterns categorized into templates, we give a conceptual view on how an integration pipeline and message processors can be synthesized to hardware. Figure 6.25 gives a conceptual overview

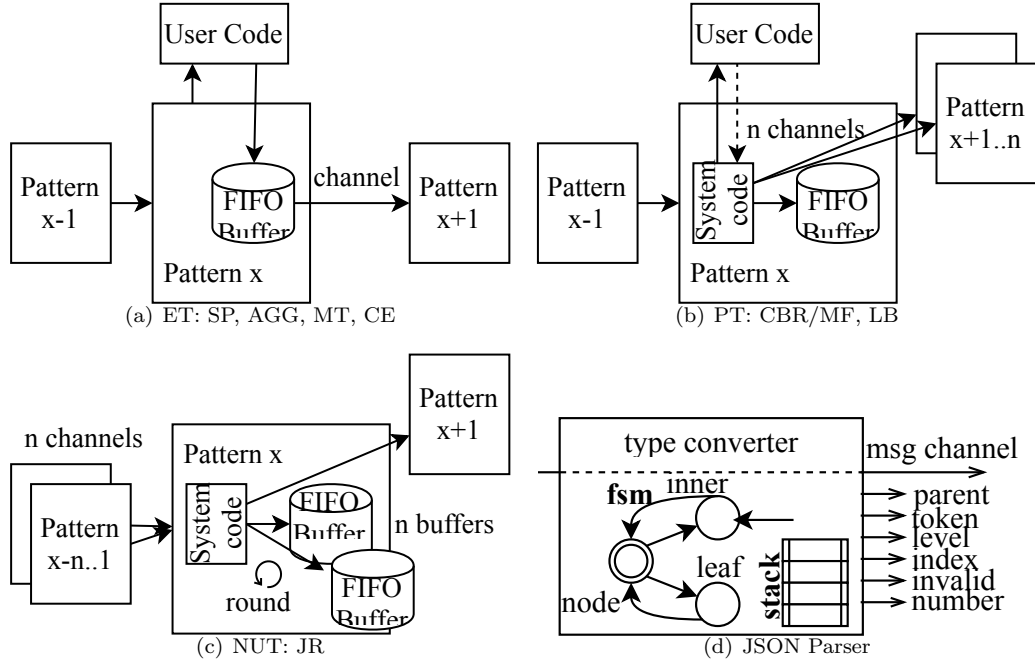


Figure 6.25: Conceptual view on pattern templates and hierarchical format processing

for the three templates. The rectangles denote patterns or user code, the cylinders buffers and all straight directed edges denote message channels. The dashed edge returns **channel** and **enable** (cf. Figure 6.22). The type converter Figure 6.25(d) will be discussed in Section 6.2.3.

The ET patterns wire the data to the user code, where it is evaluated. The user code can be in any *Hardware Definition Language* (HDL) or the compilation of higher level languages to HDL like OpenCL. The result is buffered in a FIFO queue to support buffered streaming patterns and deal with back-pressure.

The PT patterns require more system logic (cf. Figure 6.25(b)). Hence, the input data is wired to the system code that executes the user code and also stores the messages in a FIFO queue for buffered streaming patterns and back-pressure handling. In addition, the user code does not return the modified message, but **channel** and **message** signals. Consequently, the data has to be forwarded from the buffer. The buffers are reset after the message was sent. For n outbound channels, the system code creates n buffers for wiring subsequent patterns.

The Join Router NUT receives messages from n message channels at the same time, which are put into n FIFO buffers correspondingly. Figure 6.25(c) shows the NUT running a round robin fetch from the FIFO queues. Then the messages are pushed further, and the buffers are emptied. The back-pressure technique is used to avoid several messages arriving at the same channel, while the buffer of this channel is not yet empty (cf. **readReady** signal).

6.2.3 Message Processing

The message processing is defined by predicates and expressions as user code. While we decided to transfer data as sequences of bytes, the data can have arbitrary message types (e.g., a simple type like integer, or a more complex type like JSON). For instance, JSON messages can be evaluated

by a JSONPath predicate. This in return can be implemented as an automaton [FU80, WTA10]. For predicates, we define one automaton for parsing the message (i.e., similar to a type converter [IA10]) and one for matching a set of conditions. In contrast, the data might be changed based on complex expressions, for which one type converter might be used either together with EIP-typed automata (e.g., a Message Translator state machine) or user-defined hardware code that constructs the output message.

Message Protocol Handling

In this paper, we focus on hierarchical message formats like JSON, and thus implemented a streaming type converter that parses the data stream (Figure 6.25(d)). To explore the general question in integration systems on “where to do the type conversions?”, we placed the type parser in every user code that accesses the message. For instance, we placed the type converters between a pattern’s system code and user code in ET and PT (cf. Figure 6.25(a), Figure 6.25(b)). The NUT has no user code, hence, no dependency on message types.

An alternative approach is having one type converter at the beginning of each flow, which would require less instances of the type converter (i.e., less LUTs, FFs). However, this converter requires that all message channels are of the same type, which reduces its flexibility in usage and consumes more resources due to bigger amounts of fully materialized data in the FIFO buffers and during processing. In our approach, we work with generic message channels that are not bound to one data type.

When parsing JSON messages, we assume an automaton with a JSON-specific alphabet (i.e., for tokens and nodes). For handling hierarchical messages, we define a deterministic automaton with start state, an inner node (i.e., object, array) with transitions to inner and leaf nodes, denoting simple typed values or complex types like object for arrays and name/value pairs for objects. Figure 6.25(d) depicts the outbound interface of the general type converter for hierarchical data structures. The `token` signal indicates the current token (e.g., string, quote, comma), which depends on the current state of the automaton. The `parent` signal gives the type of the parent node (i.e., object or array). It is set when a new inner node is encountered and the old value has to be pushed to a stack, which is popped when the new inner node ends. The `level` denotes a pointer in the hierarchical data structure and the `index` signal denotes the index of the node in the current level of the hierarchy. The level is increased and decreased as values are pushed and popped from the stack and the index is increased when a new node is encountered, and propagated to the stack together with `parent`. The `invalid` signal indicates a malformed structure. Only the `number` signal is specific to the JSON parser and passes a value for convenience, if the current token is a number. This design parses arbitrary hierarchical structures, while consuming less resources, e.g., compared to [MLC08].

Predicates and Expressions

After the type conversion a predicate or expression can be executed. Although we mainly show the user code directly in VHDL for our evaluations, a general purpose JSONPath to VHDL translator — at least for predicates — is available. As illustration, Listing 6.6 shows how a JSONPath predicate, matching `OTPRICE` lower than 100,000, is generated to VHDL as a condition for a Content-based Router. While the code generated from the JSONPath serves as illustration, FPGA vendors provide alternative languages (e.g., Intels I++, former A++, to VHDL compiler) that could be used for formulating more complex expressions.

Table 6.2: Test hardware comparison

Characteristics	Xilinx XC7A35T	Z600
lookup tables (LUT)	20,800	-
flip-flops (FF)	41,600	-
block RAM (BRAM) / on-board RAM	1,800 kB / 256 MB	- / 24 GB
clock rate (CR)	100 MHz	2.67GHz
cores	-	12

On-chip BRAM (divided in 50*36 and 100*18 kB units) is accessible in one and on-board RAM requires several cycles (depends on fabrics).

Listing 6.6: Match field

```

1  if en='1' and token=NAME
2  then
3    if data=otprice(index)
4    then
5      index<=index+1;
6    if index=otpriceLen-1
7    then
8      field<='1'; index<=0;
9  end if;end if;end if;
```

Listing 6.7: Eval. Cond.

```

1  if field='1' and en='1'
2  and data=COMMA then
3    channel<=0;
4    -- $[?(@.OTPRICE<100000)]
5    if number<100000 then
6      channel<=1;
7    end if; channelEn<='1';
8  end if;
```

We assume a converted JSON message, which means that the signals `enable` (in the code shortened to `en`) and `data` are those of the message channel and the signals `token` (line 1) and `number` come from the type converter. The code in Listing 6.6 is located inside a VHDL process, triggered by the clock that also drives the `data` and `enable` signals. The incoming data is compared to the string `otprice` (line 3) and if the whole string matched (line 6), the `field` signal is set high (line 8). In the clock cycle after the whole total price amount was read (i.e., the data is a comma that is the end of every line) the signal `number` is compared to 100,000 (in Listing 6.7, line 4). The default channel is 0 and if the number is smaller, the message is routed to channel 1.

6.2.4 Evaluation

We evaluate the FPGA stream processing for application integration — represented by the three template variants (i.e., ET, PT and NUT) — assuming the FPGA can be used as a network-attached integration system (cf. Figure 6.16), e.g., as part of a company network or a cloud setup [CSZ⁺14]. We selected the Arty educational board shown in Table 6.2 with a Xilinx XC7A35T FPGA for the hardware tests. We compare the results with the open-source, software integration system *Apache Camel* [IA10] on CPU that was introduced in Chapter 5. For the CCT scenario benchmark, Intel provided us with the more “product-ready” Arria 10 SoC FPGA with 500 MHz clocks, 42,620 kB on-chip RAM, 1,006,720 registers and 251,680 adaptive logic modules, which we used in some of the other experiments as well. With this more powerful FPGA, we expect a linear increase of message throughput by the factor five higher clock speed. Camel runs on a HP Z600 work station, specified in Table 6.2. Besides verifying the feasibility and correctness of our approach, the main goal of the experiments is to perform throughput measurements, to study instance parallelization and resource consumption. Therefore we use the EIPBench pattern benchmark from Chapter 5, which specifies benchmark configurations derived from “real-world”

Table 6.3: FPGA pattern benchmarks

Benchmark	EIPBench (cf. Chapter 5)	Description
CBR-A	CBR-A	simple cond.: OTOTALPRICE < 100.000
CBR-B	CBR-B	multiple conds.: OTOTALPRICE < 100.000, ORDERPRIORITY = "3-MEDIUM", OORDERDATE < 1970, OORDERSTATUS = "P"
CBR-C	CBR-C	conds. on same fields as CBR-B, but multiple branches with different values
LB-x	LB-A	distributes messages over x routes.
SP-A	SP-B	split fields (iterable) into msgs.
SP-B	SP-C	split order fields (iterable) into separate msgs. while always adding head and tail
AGG-A	AGG-B	aggregate fields into msg. (SP-B reverse)
AGG-B	-	aggregate order entries into msg. (SP-C reverse)
JR-x	-	Join Router which joins x routes.
MT-A	MT-B	map names and filter entries according to a mapping program
CE-A	-	copy each entry, concatenate with a constant

Content-based router (CBR), Load Balancer (LB), Splitter (SP), Aggregator (AGG), Join Router (JR), Message Translator (MT), Content Enricher (CE).

integration scenarios.

To keep this section self-contained, Table 6.3 summarizes the benchmark configurations from Chapter 5 for the benchmark definitions that are relevant for our evaluation and maps them to our benchmark identifiers (e.g., the EIPBench SP-B \mapsto SP-A). We used the existing EIPBench definitions, however, added new benchmarks required for our analysis (cf. Table 6.3 without EIPBench representation). The hardware throughput for all benchmarks is measured with a simulator provided by Xilinx that uses post implementation simulation and element timing data of the FPGA as in [MTA09a]. First we study the message throughput on a single data stream with the same message size, and later we consider parallelism and message size, before we showcase our motivating scenario.

Pattern Throughput in Perspective

In this section we study the message throughput of our FPGA-based integration patterns. This will show better results than the software implementation. We use the configurations from EIPBench for all considered patterns in this work and subsequently identify them by their abbreviations (cf. Table 6.3).

We measure the empty pipeline as a baseline (i.e., without message processors) for all three FPGA templates and for the Camel using EIPBench order messages. The results are collected in Figure 6.26, which shows that some of the FPGA patterns perform close to the baseline (i.e., near optimal for most benchmarks).

Message and Content Generation Although the Splitter and Aggregator are classified as routing patterns according to [HW04], they reside in the ET template with the Message Translator and the Content Enricher. Figure 6.26(a) shows that the Splitter SP-A performs close to the baseline, since it is emitting the same amount of data that it consumes. In the second, SP-B case, the Splitter has to wait for the end of the message to be able to create the messages correctly and then emits the head with one entry from the iterable and the tail multiple times. The results for this data generating pattern are better than for Camel. However, the increasing amount of data reduces the throughput.

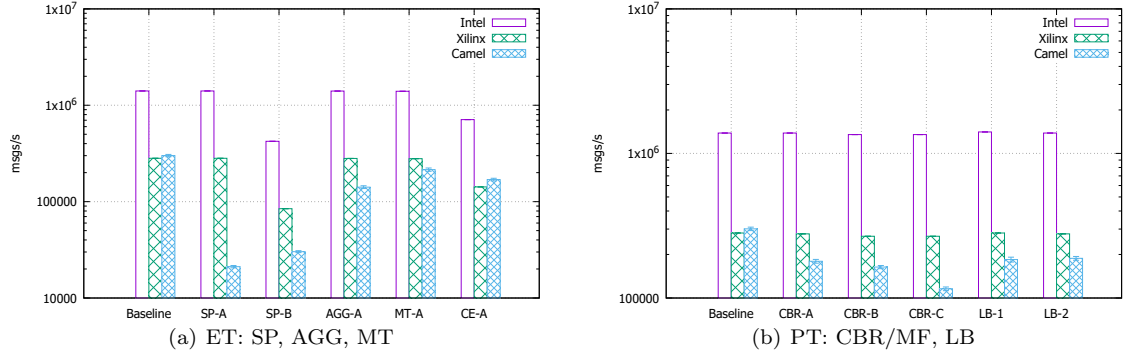


Figure 6.26: Template throughput for predicate and expression template patterns

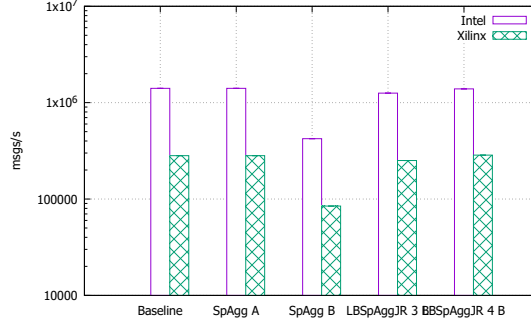


Figure 6.27: Template throughput for composed patterns

While the Aggregators AGG-A (reverse SP-A) and AGG-B (reverse SP-B; similar to AGG-A, thus not shown) as well as the Message Translator MT-A on the FPGA perform close to the baseline, the content-generating CE-A Content Enricher case shows a similar effect as for SP-B. That is due to the CE-A enricher case essentially duplicating the amount of data processed. While the Camel enricher is not limited by physical on-chip memory, the hardware enricher throughput reduces to half of the possible capacity (cf. baseline). The hardware throughput would be higher on bigger FPGAs. The high throughput of the Aggregators are measured on the order data set, not the much smaller, split order messages, which are produced by SP-A (in average 11.73 B) and SP-B (in average 197.67 B). When testing these cases AGG-A performs at 8,396,946 msgs/s and AGG-B at 518,134 msgs/s. This indicates that our approach saturates well up to the physical capacity limits.

Conclusions: (1) The message throughput of transformations (i.e., MT-A) and routings (i.e., SP, AGG) is much higher for all of the benchmarks, except for the CE-A case. (2) Content generating patterns lead to degrading throughput on the FPGA due to the increasing messages sizes and saturation up to the resource limits (e.g., BRAM). (3) The throughput scales linearly with more hardware resources.

Multiple routing conditions and branchings Let us start with the question from EIPBench about the “impact of multiple conditions and route branchings” for the Content-based Router.

The software implementation shows a decrease in throughput, especially when increasing the number of conditions and branchings in routing cases CBR-B and CBR-C (cf. Figure 6.26(b)). On the hardware, all router implementations score close to the baseline due to the parallel processing capabilities of our approach and the underlying hardware support. Hence, as long as the conditions can be executed in parallel, neither multiple conditions nor branchings significantly reduce the throughput.

The same results were observed for the Load Balancer, which is based on the same PT hardware implementation. Independent of the number of branches, the Load Balancer shows results close to the baseline. These observations indicate a major benefit of hardware over current software designs for routing patterns (e.g., due to thread handling).

The third template, which is implemented by the Join Router pattern, again scores close to the baseline (not shown). It is independent of the number of branches it accepts, too.

Conclusions: (4) The hardware throughput is invariant to the number of multiple, parallelizable conditions and route branchings. (5) The Load Balancer and Join Router perform near baseline (i.e., without message processors).

Pattern composition The previous results indicate that some variants of the Splitter, Aggregator Load Balancer, and Join Router can be combined to a composite message processing pattern without much throughput penalty (i.e., the composition would perform close to the baseline). However, data generating patterns should be avoided. For example the Scatter-Gather pattern uses a Multicast pattern (cf. Chapter 2), which copies the messages, thus increasing the amount of data. However, fork and join patterns that introduce no performance penalty like the Load Balancer and the Join Router are usually used to support optimizations such as “rewrite operator to parallel operator” or “rewrite sequence to parallel” (cf. Chapter 4). Especially patterns with less message throughput (e.g., Splitter, Content Enricher) might benefit from a parallel instantiation. To shed some light into this hypothesis we conducted the following experiments on the FPGA only: composite message processing for the simpler SP-A and AGG-A case (i.e., SpAgg A) and for the more complex B case (i.e., SpAgg B), as well as the “rewrite sequence to parallel” optimization with three (i.e., LBSpAggJR 3) and four (i.e., LBSpAggJR 4) parallel SP-AGG sub-sequences for case B.

Figure 6.27 denotes the results for these experiments, showing a near baseline throughput for SpAgg A case. The SpAgg B is dominated by the Splitter, which reduces the throughput to the individual SP-B result. When adding a Load Balancer that distributes the messages to three instances of a Splitter, Aggregator pair, then afterwards joining their output messages using a Join Router (cf. LBSpAggJR 3B), the throughput can be increased significantly. With four Splitter, Aggregator pairs, the throughput is near baseline again.

Conclusions: (6) The patterns that duplicate data like the multicast reduce the throughput. (7) The “sequence to parallel” optimization works well on sub-processes that only temporarily work with more data.

Parallelism: Space Management

The inherent support for parallelism is an advantage of FPGAs. When instantiating multiple integration scenarios in FPGA hardware, multiple message streams can be processed truly in parallel. The number of deployable scenario instances is determined both by the size of the FPGA, i.e., its resource capacity (incl. LUTs, FFs and BRAM), and by the capacity of the FPGA interconnect fabric.

Table 6.4 shows the resource occupation of the system code building blocks we explained earlier. One important limiting factor of these building blocks is the BRAM. Each FIFO queue

Table 6.4: Resource occupation

Building Block	LUTs	%	FFs	%	BRAM	%
Translation	82	0.39%	187	0.45%	1	1%
Routing(1→2)	246	1.18%	523	1.26%	4	4%
Join Router(2→1)	193	0.93%	361	0.87%	2	2%
JSON Parser	752	3.62%	897	2.16%	0	0%
UDP Receiver	649	3.12%	437	1.05%	1	1%
UDP Sender	457	2.20%	234	0.56%	0	0%

This is the maximum space occupation of each building block with optimization turned off. This can be much less, when only a few features of the building block are used. BRAM is in 18 kB units.

Table 6.5: Resource occupation CBR-A|SP-B

Building Block	LUTs	FFs	BRAM
user code (w/o parser)	101 1907	67 2977	0 0
JSON parser	504 396	193 159	0 0
messageRouter	246 -	523 -	4 -
messageTranslator	- 82	- 187	- 1
total	851 2385	783 3323	4 1
percentages	4.09% 11.47%	1.8% 7.99%	4% 1%

in the building block uses one 18 kB BRAM block. For example a maximum of 25 routers can be placed on the Artix-7 chip we used for our hardware tests. Code that has a lot of state, like the JSON Parser, has a high occupation on LUTs and FFs. The resource occupation numbers in the table are obtained via the resource occupation analysis tool of the Vivado IDE, that can be run on a synthesized and implemented design.

We placed one fully configured instance of the ET and PT templates on the Artix-7 chip, the SP-B, and CBR-A. Table 6.5 shows the resource usage differentiating the JSON parser, the user and template code. We also give the usage in percent of the total number of available resources. Note that there is a significant difference in size between the space required by the user code including the JSON parser (101+504, 1907+396 LUTs, respectively) and the space required by the system template code (246, 396 LUTs). This overhead indicates that the space consumption and the pattern performance hugely depends on the specific user code. Another interesting effect is the implicit optimization during synthesis to the FPGA (e.g., the reduction of the JSON parser to the features that are used).

The usage of parallelism brings forth another design trade-off characteristic of FPGAs. Due to their space occupation, the CBR-A can be instantiated 24 times and the SP-B 8 times on the Xilinx chip. To accommodate these instances, the VHDL compiler has to trade latency for space by possibly placing unrelated logic together into the same slice, resulting in longer signal paths and thus longer delays. This effect can also be seen in Figure 6.28, where we illustrate the space occupied by three of the 24 CBR-A configurations (cf. instance 1, 2, 24). Occupied space regions are not contiguous, which increases signal path lengths. This effect has also been identified for predominantly asynchronous designs [Cas05, MT10], while our experiments did not show any negative impact on the message throughput for our mostly synchronous designs. In summary, with more on-chip resources (e.g., FFs, BRAM) a higher degree of scenario instance parallelism, and thus more overall throughput could be reached.

Conclusions: (8) The message protocol handling and the complexity of user code impact the space consumption on the hardware. (9) The parallel processing through multiple instances is limited by the FPGA’s hardware resources. (10) The on-chip signal path length does not have

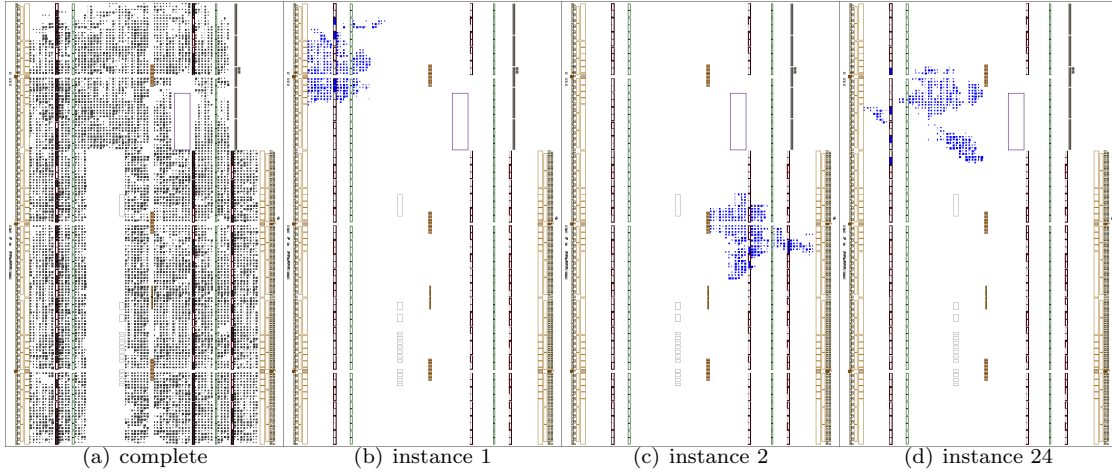


Figure 6.28: Resource usage on the FPGA chip (floorplan) for the predicate template (i.e., CBR-A) and the remaining system components

a significant impact on message throughput for our design.

Parallelism: Performance

One of the important questions to answer for pattern solutions is “what is the impact of concurrent users?” (cf. [Chapter 5](#)). To answer this question we use the SP-B and CBR-A configurations mentioned above to run up to 24 independent data streams in parallel. We call them processing units to allow comparison with the results of the software system from [Chapter 5](#), where multiple threads were measured. [Figure 6.29](#) shows the message throughput per second for an increasing amount of parallel processing units compared to the corresponding software implementation.

An important observation is that running additional process instances has no impact on the other instances, which let the processes be executed concurrently. Thereby the signal path length does not decrease the measured message throughput. Consequently, the throughput of the two hardware templates scales linearly with the number of process instances until the space limit is reached (i.e., for 8, 24 units, respectively). The multi-threaded software implementations, executed in a single JVM-process, cannot provide the same level of parallelism as an FPGA. This could be achieved with more JVM-processes on more CPUs, however, at a considerable expense (e.g., management of JVM or even VM instances, and increased power consumption).

Conclusion: (11) The throughput scales linearly with the number of instances until resource saturation on the hardware.

Message Size

The EIPBench specifies a scale factor for data size benchmarks to target the question on “what is the impact of message sizes?” (cf. [Chapter 5](#)). Therefore we used the TPC-H order-based messages approximately up to 8 MB per message from EIPBench and evaluated them for all defined templates. [Figure 6.30](#) depicts only one result, because all baseline measurement of the different templates performed identical.

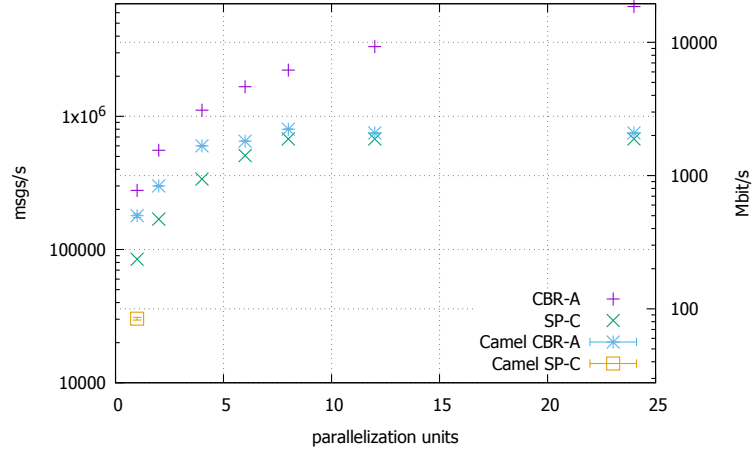


Figure 6.29: Concurrent user measurements

The immediate observation is that for increasing message sizes the memory bandwidth is saturated. In contrast to designs that use higher level memory (e.g., DRAM) and in-memory object materialization [MTA09a], which scales linearly with the input data size, our approach uses byte-streams using the local BRAM only for flow control (e.g., back-pressure). This design decision makes the approach directly limited by the practical upper boundary of resources on the board, which allows a throughput of approximately 800 MBit/s. For illustration, we added a secondary axis for MBit/s in Figure 6.30, which shows an almost stable data volume saturation. We conjecture that for most of the IoT scenarios the high number of smaller messages should fit into the on-chip memory.

As an evolution of our design, similar to [MTA09a], we could aim to load the messages in RAM temporarily and only pass the message pointers and a message structure, e.g., containing a map of names in a JSON to pointers to their data, from pattern to pattern (cf. *Claim Check*). This would allow for bigger messages, if the patterns only perform operations on small parts of the message (e.g., object lookup). For those cases pointers to larger messages could be transported, while keeping the processing fast. In case of many write and/or read operations, data has to be passed back and forth to the on-board RAM (compared to on-chip BRAM), which would decrease message throughput.

Conclusions: (12) The throughput is physically limited by the capacity of the hardware. (13) The throughput can be increased by using secondary memory, however, trading for more selective data operations.

Patterns to Scenarios: Connected Cars Telemetry

Let us get back to the motivating connected car example from Figure 6.17. The data sent from the vehicles separates into approximately 304 B error code JSON messages with fields like "Diagnostic.Trouble.Codes": "MIL is OFF0 codes" and approximately 762 B telemetry data with fields like "Vehicle.Speed": "0km/h" and "Engine.Load": "18,8%" from the car's OBD device. The error codes are enriched with master data of the owner by lookup of the obd2.Vehicle.Identification.Number_(VIN) and translated into the format understood by the receiver. For the telemetry data processing, the latter two steps are performed as well, while an additional Message Filter is added to consider driving cars only. The differentiation between

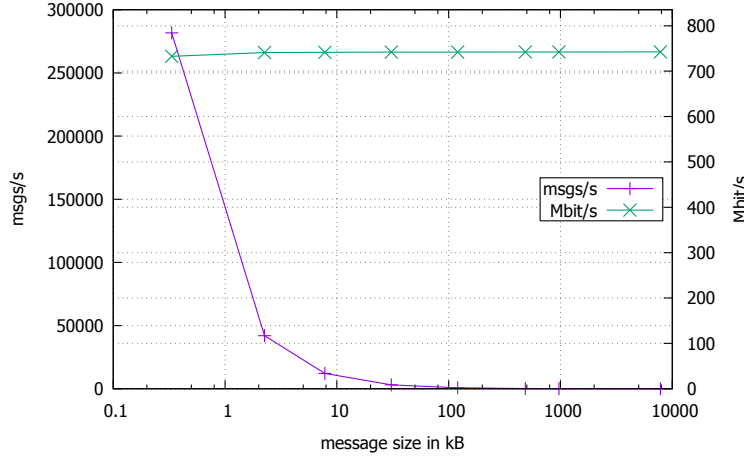


Figure 6.30: Message size baseline measurements

error code and telemetry data is done by an initial Content-based Router and the two control sequences are combined by a Join Router, before enriching and translating.

Figure 6.31 depicts the message throughput for the sketched simplified implementation of the sample scenario divided into the paths taken through the scenario: error code, telemetry and filtered telemetry. While the measured throughput for the unoptimized processing (i.e., **normal**) is as expected, considering the single pattern results, some control- and data flow optimizations from Chapter 4 can be considered. We exclusively focus on techniques that are especially beneficial for our hardware-based approach and the particular scenario. For instance, on FPGAs the flows are executed instantly and in parallel. Hence, optimizations like “Reschedule start of parallel flows” [RFRM19] (i.e., “start the slowest flow first”) and “Merge parallel flows” (i.e., “avoid forking costs”; no performance improvement, but space reduction possible on FPGAs), are not applicable or desirable.

The first control-flow optimization that looked promising was “sub-process parallelization” (i.e., **sub-parallel**), which aims to exploit the FPGAs parallel processing, of the sequence: Content Enricher and Message Translator. This worked well for the error codes, however, not for the telemetry, due to the size of the data.

Since the data and not the control-flow seems to be the limiting factor in this scenario, message-flow optimizations are more promising. Merging the neighbor patterns (i.e., **neighbor merge**) requires less resources due to the removal of one channel. However, the freed space is not enough for another instance and the performance penalty of a channel is low, thus no significant throughput increase is measured.

The throughput of the filtering can be increased in all cases, when stopping the evaluation immediately, when the condition matches (i.e., **early select early-out**). This optimization worked well because significantly less data has to be processed.

The optimizations that work well for query processing are early-selection and early-projection (i.e., similar to [MFPR90]). The selection optimization (not shown) has no positive effect (i.e., throughput increase less than 30 msgs/s), because the Message Filter is not able to cancel a message transmission when the condition does not match. The early-projection places a content filter pattern (not discussed) to the beginning of the process that filters empty fields. The results in Figure 6.31 for **early project** show an increase in throughput, since less data is moved through the hardware pipeline.

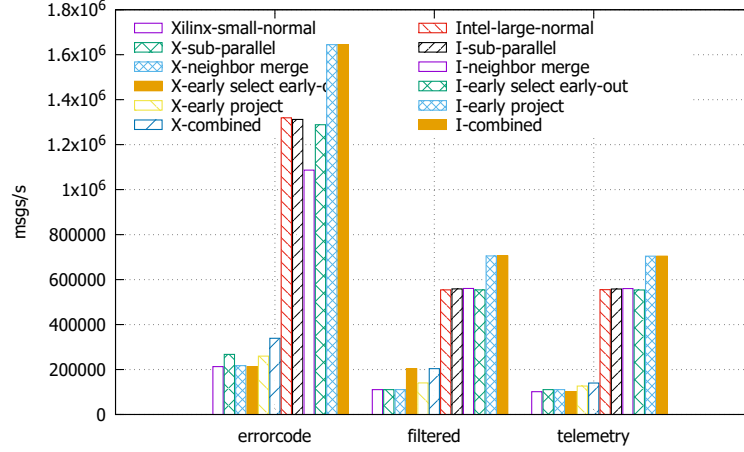


Figure 6.31: Connected cars scenario performance

Lastly, we combined several of the promising optimisations like sub-parallel, early-out and early project into one scenario and measured the performance (i.e., **combined**). For the error codes and the telemetry the sub-parallel and early project optimizations increase the throughput, because the early project significantly reduced the amount of data and the sub-parallel parallelized the slowest part of the flow. For the filtered messages the early-out optimization caps the throughput. Seven parallel scenario instances with combined optimizations fit onto the FPGA.

Power Consumption. For this setup the FPGA can handle 153,061.22 msgs/watt, while the CPU on the Z600 processes 11,052.32 msgs/watt (i.e., measured CPU consumption only). The measurement follows the technique in [MTA09a], which measures the power consumption of CPU and FPGA. The power consumption P of a logic circuit depends linearly on the frequency at which it operates: $P \propto U^2 \times f$, with voltage U , frequency f . For the consumption of the FPGA, a power analyzer provided by Xilinx reports an estimated consumption of 1.0 W, and for the CPU the consumption lies between the Extended HALT Power and the Thermal Design Power around 95 Watt. Hence, the scenario can be handled by an FPGA with much less energy consumption compared to a CPU. If all US cars sent one message every second, this would be a saving of approximately 92.78% of energy, which becomes a critical factor for most data centers⁵.

Product-ready Hardware. Despite the promising results on Xilinx, we studied the CCT scenario on the Intel Arria 10 SoC FPGA. Figure 6.31 shows the message throughput for one scenario instance (cf. **Intel-large-normal**) as for Xilinx. The results are as expected due to the factor five higher clocking (i.e., 100 MHz Xilinx vs 500 MHz Intel), thus our design scales linearly and could cover CCT for the car traffic in most countries with one FPGA.

Conclusions: (14) Especially data flow optimizations that reduce the message sizes increase the throughput. (15) Besides the optimizations suitable for CCT, a more systematic study on hardware process optimizations and their combination is required that collects all control and data flow approaches, analyzes their applicability to hardware and their impact on the message throughput and space reduction.

⁵NRDC, Anthensis. Data Center Efficiency Assessment, visited 5/2019: <https://www.nrdc.org/sites/default/files/data-center-efficiency-assessment-IP.pdf>

Discussion

From these observations we conclude that FPGAs allow for effective message processing based on the EIP streaming semantics. Despite the slow clock rate of our Xilinx FPGA (100 MHz), it achieves even higher throughput than powerful general-purpose CPUs, because FPGAs can implement fully concurrent data streaming pipelines. At the same time it consumes less power than a general-purpose CPU.

Additionally, we would like to discuss further non-functional topics including security aspects, exception handling, and monitoring of the hardware processing as they are required in advanced scenarios. While some security aspects can be handled on the transport protocol level, characteristics like message privacy either require additional on-chip IP cores or an on-board CPU (e.g., for decryption). The exception handling in integration scenarios can be implemented with a “stop process” and “message retry” on exception processing (cf. [Chapter 2](#)). As part of future work, additional logic for the cancellation of a message in the hardware pipeline and a timed resend have to be investigated. Eventually, monitoring capabilities on message and channel level are required for a more productive setting. Therefore, the statistics could be written to the on-board RAM and asynchronously fetched and processed by an on-board CPU or re-routed using a Multicast or Wire Tap pattern [[HW04](#)].

6.2.5 Conclusions

To address the velocity challenges of current software application integration solutions, we assessed the potential of FPGAs as message processors for data-intensive operations in the context of application integration. Therefore we specified an *integration system on a chip* (short ISoC, cf. question HW1) and define compatible EIP message streaming semantics, along which we categorized the patterns into three templates that can be efficiently synthesized to hardware (cf. question HW3). In addition, we specify a lightweight component for hierarchical message format processing (cf. question HW2).

Our experiments illustrate how FPGAs help to improve message routing and transformation throughput on hardware compared to a comparable software setup (cf. question HW4), e.g., due to the invariance to the number of branches and conditions. Our analysis also revealed some limitations for further research (e.g., dealing with data-generating patterns, message sizes). Furthermore, pattern instance parallelism can be used to scale the throughput to cover whole real-world integration scenarios on one chip by synthesizing multiple scenario instances. More generally, the analysis showed the suitability and soundness of our ISoC design for efficient message processing, e.g., for challenges like multiple, parallelizable conditions and route branchings and threading (cf. conclusions (1), (3)–(5), (10), (11)), identified interesting throughput vs. state trade-offs (conclusions (2), (6)) as well as limiting factors like UDF code quality and resource constraints (cf. conclusions (8), (9), (12)–(13)), discussed the suitability of some optimizations from [Chapter 4](#) (cf. conclusions (7), (14)), and resulted in future work in the form of a more systematic study of optimizations required (cf. conclusion (14)). For example, we analyzed and discussed the applicability of existing optimization techniques from the application integration domain to an integration process synthesized on hardware. But we leave a more systematic analysis as future work.

We conclude that a dataflow ISoC on the network path allows for a competitive and sustainable alternative to software-based solutions. While there are still many technological and operational challenges to solve (e.g., throughput vs. state trade-off), we conjecture that solutions on re-programmable hardware denote the future of fast, data-intensive processing on wires.

Table 6.6: Multimedia data induced shift of core EAI characteristics

EAI Concept [Lin00, HW04]	Foundations [Lin00, HW04]	Emerging: Media [LSDJ06], Chapter 2
Message (definition)	header, body	header, body, attachments
Message Protocol (format)	structured / textual (e.g., XML, JSON)	multimodal: textual, binary / media (e.g., image, video)
Message size	small to medium (B, kB)	medium to large (kB, MB)
Message Endpoint (sender, receiver)	few, static (e.g., on-premise applications)	many, dynamic / volatile (e.g., mobile / IoT devices, cloud applications)
Message channel (transport, style)	asynchronous	synchronous / streaming, asynchronous
Adapter, processor (interaction, processing)	relational	relational, semantic, confidence / probability

6.3 Evolution: Multimedia Pattern Solutions

Through the interest of (business) applications in social media, multimedia, personal and affective computing [RMRM17], socio-technical interactions and communications are introduced into applications. Thus, enterprise application integration (EAI) is now required to process unstructured multimedia data, e.g., in agricultural [Til91, YPL⁺00, BH15, M⁺15] and medical applications [AGWC14], and from social sentiment analysis [SAP19a, SSS07, TYRW14]. Following the idea in [LSO⁺08], we argue that the sequence of operations of many multimedia applications actually denote integration scenario, thus leading to new EAI characteristics with respect to the representation and variety of the exchanged messages (i.e., multimodal: textual and multimedia), the growing number of communication partners (i.e., message endpoints), as well as the velocity (i.e., message processing styles), and volume (i.e., message sizes) of the data [WB97].

However, the EAI foundations from 2004 in the form of the original EIPs [HW04] and system architectures [LÖON01, IA10] do not address the multimedia characteristics. Table 6.6 sets the current characteristics of the basic EAI concepts from [Lin00, HW04] into context to those of emerging applications [LSDJ06], see also Chapter 2. These characteristics lead to challenges that were re-emphasized in a recent seminars on the research directions for principles on data management (e.g., [AAB⁺17]). While their definition of multi-modal further includes temporal and spatial data, which plays a lesser role in EAI so far, we summarize the research challenges matching those of EAI. Subsequently, we introduce and discuss important EAI challenges (*CHx*) that are not met by current EAI concepts or system implementations:

CH1 *User interaction and interoperability (interaction with endpoints)*: The growing variety of message protocols with combined textual and media messages (e.g., seamless integration relational and media processing) [AAB⁺17] constitutes the first sub-challenge (a) on how to represent or model multimodal messages (i.e., relational and multimedia), e.g., in the form of message format extensions like attachments (cf. Table 6.6). The second sub-challenge is about (b) a uniform message processing, user interaction and data access changes from relational to multimodal (e.g., conditions, expressions) [AAB⁺17], which third (c) requires to deal with semantics in multimedia data (i.e., understanding data [AAB⁺17]), while over-coming the “semantic gap” [LSDJ06, TMM⁺16] as in the current MPEG-7⁶ standard. Although this was addressed by several initiatives, they targeted low-level media features that are inadequate for representing the actual semantics for business applications like emotions [SP15].

CH2 *Architectural challenges*: Current EAI architectures are challenged by the interaction

⁶ISO/IEC 15938-1:2002, visited 5/2019: <https://www.iso.org/standard/34228.html>

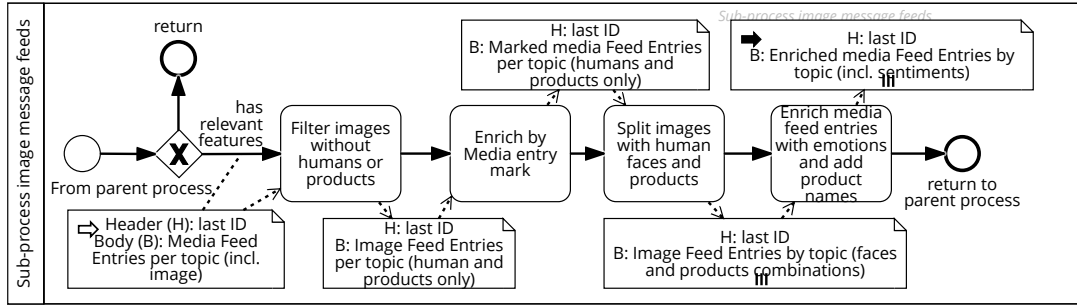


Figure 6.32: Multimedia sub-process for social media emotion harvesting (excerpt)

with a growing number of dynamic endpoints and the co-existence of processing styles: asynchronous and synchronous streaming (cf. Table 6.6; a solution was presented for textual EIP processing in Section 6.2), including additionally required components compared to the current EAI systems.

CH3 *Multimodal processing*: The challenges of combining processing styles (streaming, multimodal), dealing with distributed processing units (device, EAI system), guaranteeing efficient data processing [AAB⁺17] and optimizations [AAB⁺17] (e.g., data compression) are further complicated by increasing message sizes of multimodal content.

CH4 *Verification of multimodal EAI processes*: The challenge of verifying processes and data [AAB⁺17], addressed in this thesis for EAI in Chapter 3, is challenging for checking the correctness and compliance of multimodal processes.

Note that CH4 could be achieved through an extension of timed db-net for multimedia data, however, due to our focus on the evolution of the EAI domain, this is out of scope for this thesis and will be considered as future work (cf. Section 6.5).

Example 6.12. We consider a social media analytics scenario that illustrates the multimedia data variety challenge in EAI. The current implementations of social media sentiment analysis scenarios (e.g., [SAP19a]) are either focused on textual information, or they process multimedia data in an ad-hoc way (cf. CH1). As sketched in Figure 6.32, they usually collect and filter social feeds from sources like Twitter and Facebook according to configurable keyword lists that are organized as topics. The textual information within the resulting feeds is analyzed with respect to sentiments toward the specified topic. However, many sentiments are expressed by images in the form of facial expressions. Therefore, the received feeds would require an multimedia Message Filter, e.g., removing all images not showing a human, an Enricher for marking the feature, a Splitter for splitting images with multiple faces to one-face messages, and an Enricher, which determines the emotional state of the human and adds the information to the image or textual message, while preserving the image. The interaction with the multimodal messages by formulating user conditions and the required multimedia processing (cf. CH3) are currently done by a large variety of custom functions, thus denote ad-hoc solutions. Therefore, existing EAI systems are extended by — as it seems — arbitrary multimedia processing components in custom projects (cf. CH2) that destabilize these systems and make the validation of the multimodal EAI processes difficult. ■

These challenges are set into context of the current EIP processing in Figure 6.33, showing the new problem areas of user interaction (incl. semantic message representation and custom

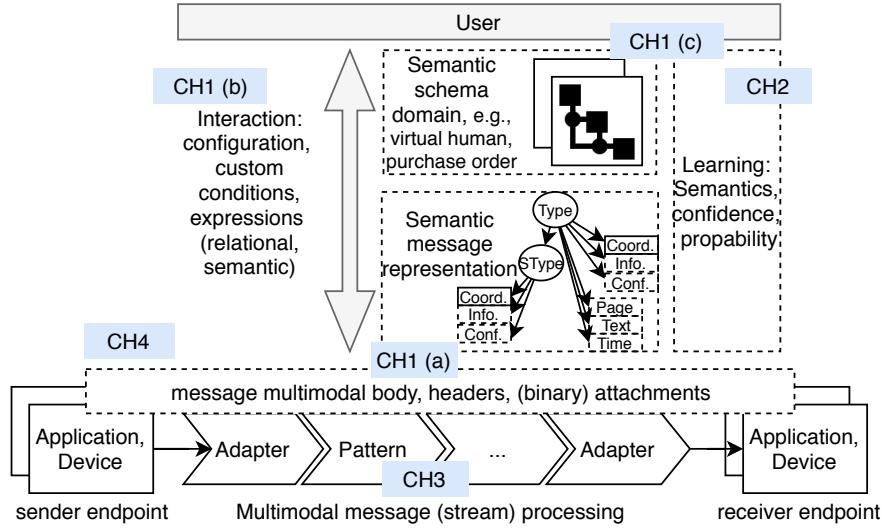


Figure 6.33: Challenges of user-centric interaction on semantic message contents

conditions, expressions), new architecture components for learning and detecting the semantics in the multimodal messages, and the multimodal message processing.

We seek answers to the following research questions (MMx) as sub-questions to RQ3-1 “Which related concepts and technology trends can be used to improve integration processing and how can this be practically realized?”, derived from the introduced challenges.

- (MM1) “Which industrial and mobile applications require multimedia application integration?”
- (MM2) “Which integration patterns are relevant in the context of multimedia data?”
- (MM3) “How could these patterns be realized and uniformly configured for multimedia data integration?”
- (MM4) “Do the current EAI architectures (e.g., [Lin00, IA10]) need extensions to support multimedia data integration?”
- (MM5) “Which additional EAI architecture components are required for multimedia message processing?”
- MM6 “How can the process-oriented multimedia integration processing be realized and improved?”

MM1–MM3 address the user interaction challenge CH1 by considering a broader view of the requirements of current multimedia solutions (cf. MM1), identifying relevant integration semantics in the form of patterns (cf. MM2), and the definition of a uniform, semantic multimedia data access (cf. MM3). The challenge CH2 regarding the architecture evolution of current systems is addressed in MM4 and MM5 that target an identification of missing architectural components compared to the classic EAI architecture described in Section 2.1.3. Finally, the challenge CH3 about an efficient multimodal processing is covered by MM6. Note that we do not focus on the areas of content-based media retrieval [LSDJ06, CDPV07], nor strive to improve existing algorithmic or hardware multimedia processing aspects (e.g., on GPU [SSGH15]), but seek a complementary mapping of the multimedia domain to EAI concepts.

To answer these questions, we introduce preliminaries on semantic knowledge representation in Section 6.3.1, before we conduct a scenario analysis in the form of a literature and a system

review of industrial and mobile applications in the context of the multimedia integration processes in [Section 6.3.2](#) (targeting MM1). The analysis results in a mapping of the integration requirements to integration patterns for existing and new patterns, i.e., not in EIP [\[HW04\]](#) (for MM2). In [Section 6.3.3](#), these patterns are set into context of the integration operations required by the scenarios, resulting in a logical representation toward a uniform user interaction (for MM3). We discuss their realization for multimedia integration scenarios (for MM6) and discuss required EAI system extensions in [Section 6.3.3](#) (for MM4+5). The proposed approach is evaluated in [Section 6.3.4](#) for its comprehensiveness and message processing throughput for the motivating social media example in a case study (for MM6).

The resulting multimedia integration system not only covers all aspects from multimedia patterns, processing and optimizations to data access and user access, but shows sufficient architectural extensions for an evolved EAI system architecture for multimedia data. The benchmark extensions allow for comparability of future extensions and alternative solutions and the grounding on a semantic knowledge representation can be used to show applicability to a representation in timed db-net as well as a formal analysis of multimedia pattern compositions.

Parts of this section have appeared in the proceedings of EDOC 2017 [\[RRM17\]](#).

6.3.1 Image Processing and Semantic Knowledge Representation

For a better understanding of our proposed solution, we briefly introduce image processing and semantic knowledge representation basics. Due to the large body of work for both topics, this can only be a brief introduction of some terminology and key concepts used in our work. For a more comprehensive introduction and further details, we refer to the literature (e.g., [\[Ros69, Jai89, Eas10\]](#)).

Digital Image Processing

The first techniques of *digital image processing* or *digital picture processing*, essentially a sub-category of digital signal processing, stem from the 1960s [\[Ros69\]](#), already with applications like medical image processing, *Optical Character Recognition* (OCR) and satellite imagery. Digital image processing subsumes all tasks required to numerically represent and process digital images. Subsequently, we briefly introduce the terms used in this work.

Digital Image A *digital image* $a[m, n]$ is the projection of an analog image $a(x, y)$ from a continuous to a discrete area [\[Jai89, Eas10\]](#). During the digitalization process, the analog image is scanned, in which the continuous image $a(x, y)$ is split into N rows and M columns, called pixels. Each pixel of the digital image has coordinates $[m, n]$ with $0 \leq m \leq M - 1$ and $0 \leq n \leq N - 1$. Besides coordinates, pixels have additional properties like depth, color, and time (for videos).

Example 6.13. The digitalization of a continuous image into eight rows and columns is depicted in [Figure 6.34](#). The original image is shown on the left followed by an image with 8×8 grid in the middle. For the calculation of the resulting image on the right, the average of all color values measured by the sensor is calculated and rounded to the next integer with a minimal value 0 (black) and a maximal value 255 (white). ■

Image Processing Operations The image processing on digital images describes the alteration of an image as well as addition of new information in the image [\[Wal92\]](#). The basic operations are subsequently introduced.

Geometric Operations. The *geometric operations* are those operations, which assign the color of a pixel at position $[n, m]$ to that of position $[k, l]$ with k, l functions on n, m [\[Jai89, Eas10\]](#).



Figure 6.34: Digitalization of an analog example image

Geometric operations can be used to create new images and common geometric operations are the translation, mirroring (change content position), rotation (change content orientation), cutting (remove content), and scaling (change content size).

Point Operations. The *point operations* usually require histograms of an image, which represents the distribution of properties like colors or brightness [Jai89]. Properties of a pixel can be read or changed (e.g., read or write pixel color), which denote a pixel's base operations. Common point operations are thresholding (select pixels with certain value to produce a binary image), segmentation (image partitioning), and arithmetic operations (e.g., logarithm operation for contrast reduction of brighter regions).

Neighborhood Operations. The *neighborhood operators* considers the pixel and its neighborhood [Jai89, Eas10] for the operation (e.g., colors of pixels in the neighborhood). The neighborhood is usually adapted to the filter applied (e.g., quadratic neighborhood for a convolution filter). The operations are commonly used for pattern recognition and filters like edge smoothing, and edge detection. Especially the recognition of patterns based on Haar-Wavelets by Papageorgiou et al. [POP98] grounded the work on Haar-classifiers used in pattern and object detection using cascading features [VJ01].

Global Operators. The *global operators* rely on point and neighborhood operators, but operate on the complete image [Jai89, Eas10]. Most common use cases are read-only operations like edge detection and object or feature detection. Write-operations denote, e.g., the marking of detected objects in images with frames of different shapes and colors.

Semantic Knowledge Representation

While the term *knowledge representation* was coined in the field of artificial intelligence and denotes the representation of real-world information in a computer-readable form for tasks like reasoning, *semantic knowledge representation* has its roots in the semantic web domain [AVH04] that is grounded on the concepts of hypertext and addressable resources via *Uniform Resource Identifiers* (URIs) [BLFM05]. The development in this domain is mainly driven by the World Wide Web Consortium (W3C), which coined several standards for the representation of data as resources, a vocabulary for the semantic data representation, and query languages for semantic reasoning that we subsequently introduce.

Resource Definition Framework The *Resource Definition Framework* (RDF) is a framework for representing, sharing and processing information in a machine readable form [KCM04]. Its abstract data model is based on triples, which consist of a *subject*, *predicate*, and an *object*. The

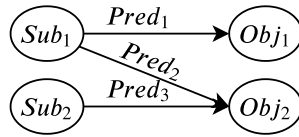


Figure 6.35: Example RDF graph

predicate represents a property that is a relationship between two things (subject and object). The object denote (literal) values, which can be empty (i.e., without global URI), although its usage for linked data is discouraged, since they cannot be linked from external documents [HB11]. The triples can be referred to as statements or assertions of relationships. A set of RDF triples is called a RDF graph, an example of which is shown in Figure 6.35, resulting from merging common subjects or objects into one node.

To add meaning, the data is assigned explicit semantics based on agreed-upon terms. Therefore RDF uses URIs as identifiers, since it has a clear authority (i.e., the URI belongs to a registered owner [BLFM05]) and the identifier is unambiguous due to their specified format. With that, RDF asserts that one resource identified by URI has a URI relationship (e.g., type, isA) to another URI resource.

Example 6.14. The resource `<http://dbpedia.org/resource/Vienna>`, as defined by the dbpedia.org organization, represents the Austrian capital city of Vienna, with namespace `<http://dbpedia.org/resource/>`. These namespaces can be abbreviated in RDF, e.g., by prefixes in the compact URI (CURIE) [BM10] scheme. To express that Vienna is a capital in Europe, we write in CURIE notation:

dbp:Vienna rdf:type yago:CapitalsInEurope,

with the following prefix definitions: **@prefix dbp:**`<http://dbpedia.org/resource/>`,
@prefix rdf:`<http://www.w3.org/1999/02/22-rdf-syntax-ns#>`, and
@prefix yago:`<http://dbpedia.org/class/yago/>`. ■

Ontologies The process of casting information about real-world entities into RDF triples requires modeling these entities according to a meta-model. This model can be a basic data model like the RDF triple format together with specific conceptualizations in the form of vocabularies (or ontologies). An ontology formally defines the terms (called classes or concepts) and relationships that can be used to model domain-specific vocabularies that are interpretable by machines. For example, vocabularies to express conceptualizations in RDF are, e.g., RDF itself, RDF Schema [Bri04] and the Web Ontology Language [MVH04], which are the result of a W3C standardization process. All provide formal semantics. The terms defined according to their semantics (in the form of triples) can be deduced using inference rules expressed as SPARQL queries [HSP13].

Example 6.15 (Virtual Human Ontology). A well-known vocabulary is the virtual human ontology that describes humans and their emotions through facial expressions. The depiction of part of the ontology is shown in Figure 6.36 and denotes a formal specification of the animation for a desired expression of emotion based on [RTKK02, GRVT⁺06]. The *VirtualHuman* class has *Face* and *Body* subclasses. The “virtual” face has animations that can have *facial animation parameters* (FAPs) like a spatial reference of facial shapes, and a facial expression *FacialExpression*,

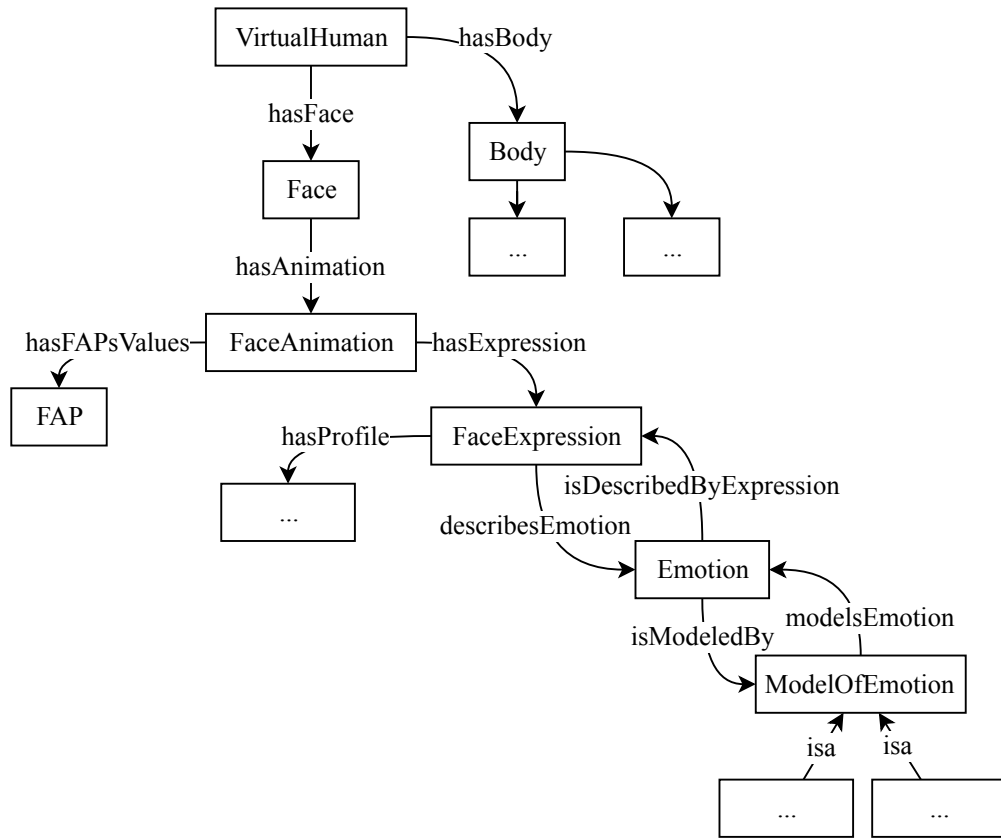


Figure 6.36: Excerpt of archetypal and intermediate face expression profiles and emotion representation of a *VirtualHuman* based on [GRVT⁺06]

which describes an *Emotion* subclass. The emotion is modeled by an abstract *ModelOfEmotion* subclass that can be one of many emotion models (e.g., Whissel’s Wheel of activation-evaluation space [Whi09] or Eckman’s model of emotion [Ekm93]).

The information that a human subject s has a face o can be expressed as

vh:s vh:hasFace vh:o

with prefix vh in RDF’s CURIE notation. ■

With that, knowledge can be represented and expressed understandable to humans and computers [GRVT⁺06].

SPARQL Protocol and RDF Query Language Similar the *Structured Query Language* (SQL) for queries on relational databases, the *SPARQL Protocol and RDF Query Language* (SPARQL) [HSP13] is used to query RDF data (e.g., stored in triple stores or remote SPARQL-endpoints), however, the difference lies in the structure of the data accessed. While the data in relational databases is organized in relational tables with primary and foreign key constraints, the RDF data denotes a graph. Hence, SPARQL allows for the specification of graph patterns to select data matching the pattern.

SPARQL version 1.1 [HSP13] denotes a rich query language over RDF by specifying several artifacts like term constraints (e.g., restricting String values), aggregators (e.g., group by), graph patterns (e.g., for empty nodes), expressions (e.g., String functions) and query forms (i.e., **ASK**, **CONSTRUCT**, **DESCRIBE**, and **SELECT**). We subsequently introduce those artifacts that are explicitly used in this work.

Query Forms. The query forms use pattern matching to form result sets or RDF graphs. The **ASK** query returns a Boolean indicating whether a query pattern matches or not. Listing 6.8 depicts its syntax with prefix definitions and conditions.

Listing 6.8: **ASK** query schematic

```

1PREFIX <PREFIX declarations ... >
2ASK {
3  <conditions>
4}
```

Listing 6.9: **SELECT** query schematic (incomplete)

```

1PREFIX <PREFIX declarations ... >
2SELECT <variables ... >
3WHERE {
4  <graph patterns with variables
      ... >
5}
6ORDER BY <conditions ... >
7...
```

The **SELECT** query returns all or a subset of variables bound in a query pattern match. Listing 6.9 shows an (excerpt) of its syntax with projection variables, graph patterns and order by clause.

Filter Evaluation. SPARQL provides a subset of XQuery [W3C01] functions and operators including logic operators, negation and existence filters. Again, we focus on filter expressions that are used subsequently, which is the **NOT EXISTS** filter operator. It returns a Boolean value depending on the bindings of the current graph pattern (i.e., **true**, if no binding exists):

xsd:boolean NOT EXISTS { pattern }

Example 6.16. A query on information represented in RDF corresponding to the virtual human ontology that determines, whether there are subjects *s* with no face *o* translates to an **ASK** query in Listing 6.10.

Listing 6.10: Virtual human **ASK** query example

```

1PREFIX vh: ...
2ASK { FILTER NOT EXISTS { ?s vh:hasFace ?o } }
```

For a set of RDF triples with only humans, the query will return **true**. ■

6.3.2 Literature and Application Analysis

We conduct a literature and application (short app) review targeting MM1 and MM2. The first goal is to compile a list of industries. Based on their scenarios, multimedia operations are discussed that are related to the integration patterns.

Methodology

The analysis is conducted along the methodology from Kitchenham [Kit04] that allows for systematic reviews as subsequently conducted.

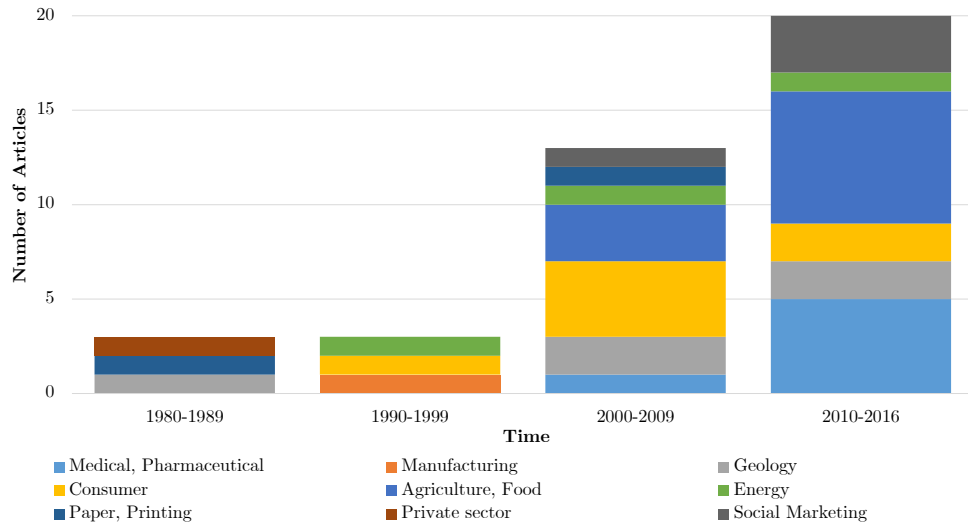


Figure 6.37: Literature analysis: image processing in industries over time

Literature Review. The primary selection of industrial multimedia scenarios from the literature was conducted using google scholar (scholar.google.com) on 2017-04-03 without patents and citations, for the keyword “image processing industry” and `allintitle`. The search results in 54 articles, of which we selected 29 articles with selection criteria “image processing” (e.g., in abstract, theme) and added the 10 papers as expert knowledge (i.e., examples from the introduction), resulting in 39 articles. We grouped the articles chronologically by the decades they were published and by industry, shown in Figure 6.37. While the contributions per industry vary, the amount of work found per decade increases. Due to brevity, we subsequently discuss the top three industries from the current decade: agriculture / food, medical / pharmaceutical and social media management.

Multimedia Applications. Similar to [TMM⁺16], we analyze current multimedia applications. The leading app store in terms of the number of applications in 06/2016 is Google Play with 2.2 million applications⁷. Hence, for the application review, we searched in Android Apps with tags “photo”, “collage” (e.g., similar to the Aggregator pattern), and “video” with “media” as context by applying the rating “4 stars+” and “for free” filters (e.g., *tags: collage, media*). We considered the first 100 entries and selected those applications with more than one million downloads. As in the literature analysis, the keywords are taken from the problem domain. This resulted in two selections for *tags:+photo,+media*, i.e., Retrica and Instagram, one for *tags:+collage,+media*, i.e., Photo Grid, and one for *tags:+video,+media*, i.e., InShot Video-Editor (only 707k downloads) without duplicates. Conducting a complementing search for a similar “photography” category search adds four more applications, i.e., Google Photo, Snapchat (both image processing), FotoRus (collage) and Textgram (Image+Text).

⁷Statista — Number of applications available in leading application stores as of 3rd quarter 2018, visited 5/2019: <https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/>.

Table 6.7: Industry and application analysis results

Applications	ADPT	SP	RT, MF	AGG	MT	CE	SEQ	DET	RES
Multimedia in Industries (from Literature Analysis)									
Farming [BH15, JKB13, Til91, YPL ⁺ 00, N ⁺ 12, DSC ⁺ 14]	✓	(✓)	✓	-	-	✓	-	✓	✓
Medical [KKP12, GR16, AGWC14, MSA14, JnYzZ11]	✓	(✓)	✓	-	✓	✓	-	✓	✓
Social [IC ⁺ 15, SAP19a, TYRW14]	✓	✓	✓	✓	-	✓	✓	✓	-
Multimedia in Mobile Apps (from App Analysis)									
photo+media: Instagram, Snapchat, Google Photo	✓	-	-	✓	✓	✓	✓	✓	-
collage+media: Retrica, FotoRus, Photo Grid	✓	-	-	✓	✓	✓	-	-	✓
video+media: InShot	✓	-	-	✓	✓	✓	-	-	✓
text+image: Textgram	(✓)	-	-	-	✓	✓	-	-	-

Required ✓, Not required (-), partly required (✓)

Abbreviations: Channel Adapter (ADPT), Splitter (SP), Message Filter (MF), Content-based Router (RT), Message Translator (MT) including Content Filter, Content Enricher (CE), Message Sequence (SEQ), Feature Detector (DET), Image Resizer (RES).

Multimedia Processing Analysis – Results

We consider the multimedia content — found in the literature and applications (e.g., image, text, video) — to be transferred and processed within an integration solution as message body (or attachment; not in EIP [HW04]) and their metadata as message header. The discussed integration patterns denote message processors that base their routing decisions or transformations on the image content (not the metadata).

Following the methodology defined in the previous section, the analysis of the selected literature identified 15 (i.e., nine explicitly and six implicitly named) out of the 48 message processing EIPs as relevant for multimedia processing. Existing patterns were selected, to which multimedia operations could be semantically assigned. New patterns constitute recurring solutions in the form of operations for a specific multimedia data problem (e.g., resize images). Table 6.7 shows those of the explicitly named patterns, for which multiple industry or application specific cases were found. The Idempotent Receiver was only named once, thus not shown. The implicitly named patterns (i.e., Datatype Channel, Document Message, Scatter-Gather, Claim Check, Canonical Data Model, Format Indicator) denote basic integration capabilities from [HW04] that are relevant for all of the multimedia integration scenarios. During the analysis we identified nine new patterns that could not be mapped to an existing EIP, of which two had multiple citations (i.e., Feature Detector, Image Resizer), and thus are in Table 6.7. All patterns with only one citation were also identified in Chapter 2 (i.e., Message Validator, Message Decay, Message Privacy, Signer, Verifier, and the implicitly named Format Converter) and are not shown, however, the validator is discussed further in subsequent sections due to its relevance for multimedia processing.

Literature Review. The results of the literature analysis are summarized in Table 6.7. Note that references are added for articles, for which integration patterns could be found. In all of the domains (i.e., agriculture / food, medicine, and social media) the captured images are optionally pre-processed (also mentioned in [LSO⁺08] as “Capture, Share”). The pre-processing usually includes format conversions (e.g., media formats; not shown), resizing (e.g., [GR16]), message translation (e.g., noise cancellation, consistent background [KKP12, GR16, AGWC14]), and content filters (e.g., hair removal for skin cancer [GR16]). An Image Resizer is also used to compress the data for agricultural monitoring (e.g., fruit monitoring [JKB13, BH15, MSA14]). Alternatively, a Splitter pattern is used to reduce the size of the individual features processed (e.g., [SAP19a]). The Feature Detector for semantic objects usually summarizes the low level image processing steps of segmentation, feature extraction and object recognition and can be found in all of the domains. The detected features are then either validated using a Validator (e.g., cancer classification [KKP12, AGWC14]; not shown) from Chapter 2 or filtered using a Content-based Router or a Message Filter, if they do not have the expected feature (i.e., found in all domains). A Content Enricher is used to add contextual information to images (e.g., weather conditions [Ti191, YPL⁺00], emotions [SAP19a]). Alternatively the image itself is enriched (e.g., by marking faces [SAP19a] or suspicious skin moles [AGWC14]). The clustering of images using Sequence and Aggregator patterns was found in social media [SSS07]. In [IC⁺15], the message deduplication is mentioned as removal of “near-duplicates”, which is covered by the Idempotent Receiver pattern (not shown).

Multimedia Applications. These results are backed by the review of the eight mobile multimedia applications, shown in Table 6.7. The channel adapters — supported by all applications — denote an important data access or collection facility to capture or load multimedia documents in the form of images or videos and share them with the contacts on other social media platforms (e.g., Twitter, Facebook). From the standard routing patterns, only the Aggregator was found — especially in the image fusion “collage” applications (e.g., Retrica, FotoRus). Special types of aggregators are the “photo to movie” function in Google Photo and “synchronize music and video” in InShot. Most of the applications make use of special image or video filters, which map to the Message Translator pattern. These filters allow to change all aspects within an image, comparable to the well-known textual Message Translator. The enrichment of images or videos with additional information like layouts, backgrounds, or texts can be seen as content enricher pattern. Notably, Instagram allows to group images as a story that vanishes after some time (i.e., Message Decay from Chapter 2; not shown). While this denotes a Message Sequence (i.e., single messages belonging together), the aspect of a timed decay of a message or sequence is not in [HW04]. Similarly, Instagram allows to send self-deleting images, which adds message decay or aging (cf. Chapter 2).

Further new functionality (i.e., not in EIPs) can be considered Message Privacy from Chapter 2 (not shown) as “private send” in Instagram, the detection of places or objects and the new processing style of streaming (not shown) in Google Photos, the signing and verification of images as “Retrica-Stamp” in Retrica (i.e., Message Authenticity from Chapter 2; not shown), and the cut, crop or resize capabilities in PhotoGrid, FotoRus and InShot that transform the images beyond the message translator pattern.

Summary: Multimodal Pattern Classification

The literature and application reviews identified several industrial domains and mobile applications that require multimedia EAI.

We collected the required integration aspects by mapping them to the existing EIP [HW04]

Complexity	Modality	
	Simple, Single modal	Simple, Multimodal
	- Capture, Share - Text-to-Text - Media-to-Media (X)	- Text-to-Media - Media-to-Text
	Complex, single modal	Complex, Multimodal
	- split, aggregate (X) - enrich (X) - resize image (X) - with resources	- Text-to-Text,Media - Media-to-Media,Text - split., aggregate - enrich media-with-text - with resources

Legend: existing (light grey), new (black), main focus (X)

Figure 6.38: Multimodal operation classification

that are affected by multimedia processing as well as identified several new patterns (i.e., Feature Detector, Image Resizer, Validator, Message Decay, Signer and Verifier), which the last four were already found in [Chapter 2](#), thus not further discussed here. In contrast, patterns like Wire Tap or Recipient List were not required by the applications, thus do not show any significant relation to media processing. For the subsequent definitions, we classify these patterns according to the dimensions “complexity” and “modality”, separating simpler from more complex operations as well as single modal (i.e., textual, multimedia) from multimodal processing (i.e., combined textual and multimedia). [Figure 6.38](#) depicts these categories that are currently not covered – apart from “Capture, Share” (Adapter Channel) and “Text-to-Text”. While many multimedia processing approaches focus on the metadata (e.g., [\[Gro97\]](#)), and thus are “Text-to-Text”, “Media-to-Media” denotes an exclusive processing of the multimedia data. Similarly, all complex, but single modal cases are either exclusively textual or multimedia processing (e.g., enrich image by adding geometrical shapes). For some of the complex cases, additional resources are required like a data store for the aggregator or a key store for the Signer pattern from [Chapter 2](#). The simple multimodal processing denotes transformations between textual and multimedia (e.g., text to image or image semantics to text). The more complex, multimodal processing includes multimodal operations like the “Media-to-Media,Text” case. We mainly focus on “Media-to-Media”, and the routing and transformation patterns from the analysis (e.g., filter, split, aggregate, enrich), required for the identified multimedia integration scenarios.

6.3.3 Multimedia EAI Concepts

We map the multimedia operations to the relevant integration patterns from [Table 6.7](#) (for MM3). Similar to [\[CDPV07\]](#), we then define a conceptual, logical representation toward a uniform user interaction (i.e., pattern configuration incl. conditions and expressions) and a physical representation for the evaluation during runtime execution, thus separating the runtime from the user interaction.

Integration Patterns in the Context of Multimedia

[Table 6.8](#) lists the relevant patterns from [Table 6.7](#) (by Pattern Name) and sets them into context to their multimedia operations. We focus on the explicitly mentioned patterns in [Table 6.7](#) (without the Sequence) and include the Idempotent Receiver, Message Validator from the list

Table 6.8: Integration pattern multimedia aspects (re-calculated as *recal*)

Pattern Name	Multimedia Operation	Arguments	Physical	Logical
explicit				
Channel Adapter	format conversion	format indicator	write	create
Splitter	fixed grid, object-based	grid: horizontal, vertical cuts; object	create	recal/write
Router, Filter	select object	object	-	read
Aggregator	fixed grid, object-based	grid: rows, columns, heights, width	create	recal/write
Translator, Content Filter	coloring	color (scheme)	write	recal/write
Content Enricher	add shape, OCR text	object, shape+color, text	write	recal/write
Feature Detector	segmentation, matching	object classifier	read	create
Image Resizer	scale image	size: height, width	write	write
extra				
Idempotent Receiver	detector, similarity	object for comparison	-	read
Message Validator	detector	validation criteria	-	read

of the patterns that were mentioned only once. All other non-listed as well as the implicitly required patterns are either covered implicitly (e.g., the Scatter-Gather pattern is a combination of the splitter and aggregator patterns) or left out due to brevity. In addition, to the pattern and the corresponding multimedia operation, the (semantic) configuration arguments relevant for the user interaction are added, while assuming that all operations are executed on multimedia messages that are part of the physical representation. For instance, all of the image collage mobile applications in Table 6.7 require grid-based image fusion for rows and columns or specify height and width parameters. The splitter, required in the social, but also partially in medical and farming industries, either requires simple (fixed) grid-based horizontal or vertical cutting or a more complex object based splitting. Subsequently, we introduce the physical and logical representation, in which contexts the relevant multimedia EAI concepts and patterns are defined.

Logical Representation

The logical representation targets the user interaction, and thus defines a Canonical Data Model based on the domain model / message schema of the messages and the access patterns. Note that through the grounding on semantic knowledge representation, changes of the logical representation implies rewriting the underlying RDF triples.

Canonical Data Model for User Interaction While there are standards for the representation of structured domain models (e.g., XSD, WSDL), in which business domain objects are encoded (e.g., business partner, customer, employee), multimedia models require a semantic representation with a confidence measure that denotes the probability of a detected feature. In contrast to [CDPV07], who defines a relational multimedia model, we assume a graph structured schema of the domain object (e.g., human expressing emotion) with properties on nodes and edges. Figure 6.39 depicts the conceptual representation of a property graph starting from the message root node (and its properties, e.g., the message identifier). For the domain object sub-graph (i.e., type **Type** with sub-types **SType**), we add another property to the (semantic) Document Message from Section 6.3.3, which is transient and removed from the message, before sent to a receiving Message Endpoint. To express the confidence on the detected domain object, all type and sub-type nodes get a **Conf.** field (e.g., type=human with conf.=0.85, stype=emotion,

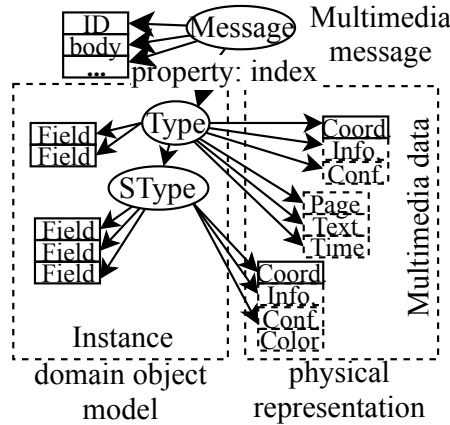


Figure 6.39: Conceptual object model

value=happy with conf.=0.95). With this compact definition, lists of arbitrary domain objects can be represented. We have chosen this representation similar to type hierarchies in ontologies introduced in Section 6.3.1, where the types and subtypes correspond to classes and subclasses in ontologies. Hence, the RDF graph of a virtual human ontology can be inserted as a concrete type beneath the **Message** type or class with their properties represented as **Field** elements. This way, the message format is semantically represented, and through the ontology-based schema information, these graphs can be formally evaluated. An instance of this model is created during message processing by the Feature Detector pattern (cf Table 6.8).

From Multimedia Features to Domain Objects / Message Schema. In our case, the term “Semantic Gap” [LSDJ06, SP15] denotes the difference between low-level image features (usually represented by n -dimensional, numerical vectors representing an object, called a feature vector) and the actual domain object that has a meaning to the user. According to the scenario analysis, we consider the following image features relevant: color, position, time (interval) in a video stream, during which the domain object was visible or the page number in an OCR document. We assume the creation of the domain object from the underlying features as given by the existing content-based media retrieval mechanisms (e.g., cf. Section 6.4), which is during the message processing in the physical runtime representation. However, for a mapping between the runtime and logical representation, we add the identified image features to our multimedia message index (cf. Figure 6.39).

Physical Representation

The basic EAI concepts located in the physical or runtime representation, according to Figure 6.33, are the (multimedia) Document Message, Message Channel, Channel Adapter, Message Endpoint (all from [HW04]), and Format Converter (see Chapter 2). In addition, all identified routing and transformation patterns have a physical representation, with which they interact. These patterns are grouped by their logical and physical data access (cf. Table 6.8) as *read/write* and *read-only*.

Basic Concepts For multimedia processing, the physical message representation covers the multimedia format, on which the multimedia operators are applied. Hence it is specific to the underlying multimedia runtime system or library. The current message definition from [HW04] of

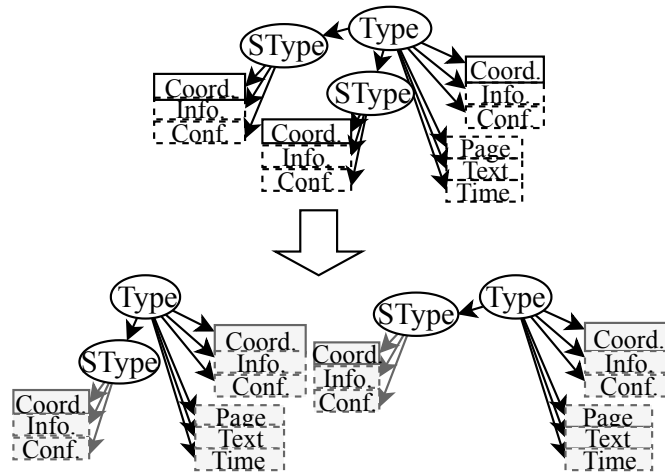


Figure 6.40: Evolution of the logical representation of a splitter

textual content (i.e., body) and their metadata (i.e., headers) is therefore extended by binary body entries and attachments. That allows to represent and query structured information in the message body together with unstructured information as attachments at the same time.

As denoted in Table 6.8, there are patterns that create, read and change / write to these messages. For instance, the Channel Adapter receives the multimodal messages from a Message Endpoint (e.g., Twitter, Flickr) and transforms (**write**; similar to the Type Converter) the textual and multimedia formats into a physical runtime representation (e.g., JPEG to TIFF for OCR processing) as part of a Canonical Data Model and creates (**create**) the logical representation that is based on the semantic features of the multimedia message content, for the user interaction. However, not all of the current integration adapters are able to handle binary content in the message body and / or attachments (e.g., SMTP separates both, while HTTP only sends one multi-part body).

Read / Write Patterns Subsequently, we differentiate different types of operations on the physical and logical representations for patterns with read / write access to the message.

Physical Write, Logical Create. The Channel Adapter transforms the incoming message into the physical message representation of the integration system (e.g., multimedia format). From the physical representation, it creates the logical object model (by schema) as conceptually shown in Figure 6.39.

Physical Create, Logical Re-calculate / Write. The Splitter cuts one multimedia message (e.g., with one image of a group), into several multimedia messages either by fixed grid (e.g., equi-distant slices) or by domain object (e.g., for each human). While the physical message representation has to be created for each part, the logical representation could be re-calculated based on the knowledge about the cuts. Thereby new physical messages are created, while the logical representation has to be updated, if it cannot be recalculated. Then the features have to be detected again and the logical model has to be updated (e.g., by exploiting the information on how the image was cut). Figure 6.40 shows the logical representation, before and after the split.

The Aggregator pattern denotes the fusion of several multimedia messages into one. Therefore, several images are combined using a correlation condition based on a multimedia object (e.g., happy customers), and aggregated, when a time-based or numerical completion condition is met

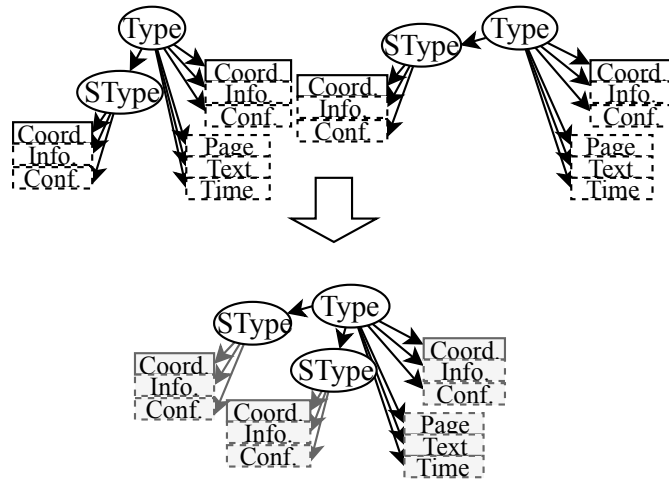


Figure 6.41: Evolution of the logical representation of an aggregator

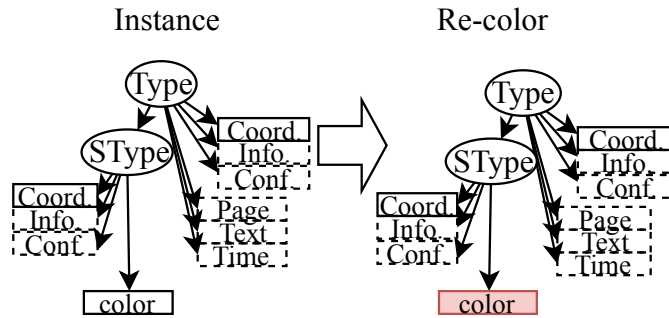


Figure 6.42: Message translator and content filter: re-coloring

(e.g., after one minute or four correlated multimedia messages). The aggregation function denotes a fixed grid operation that combines the multimedia objects along the grid (e.g., 2x2 image frames from a video stream). The logical and physical operations are the same as for the splitter. That means, the logical representation shown in Figure 6.41 can either be re-calculated or has to be re-detected and written.

Physical Write, Logical Re-calculate, Write. Similarly, the Translator and Content Filter change the properties of a multimedia object. For example, in the identified scenarios and applications, they were used to change the color of an image. Therefore the physical message has to be changed and the logical representation can be re-calculated as shown in Figure 6.42. Since this operation is less relevant for business application, it denotes a rather theoretical case, which might only slightly change the logical representation, but changes the physical representation.

In contrast, the Content Enricher adds geometrical features like shapes to images, e.g., relevant for marking or anonymization, or places OCR text, e.g., for explanation, or highlighting. Thereby, the physical and logical representations are changed or recalculated as shown in Figure 6.43.

Physical and logical Write. The Image Resizer scales the physical image and their logical representation, which cannot be recalculated in most cases. The resizer is used to scale down images similar to message compression.

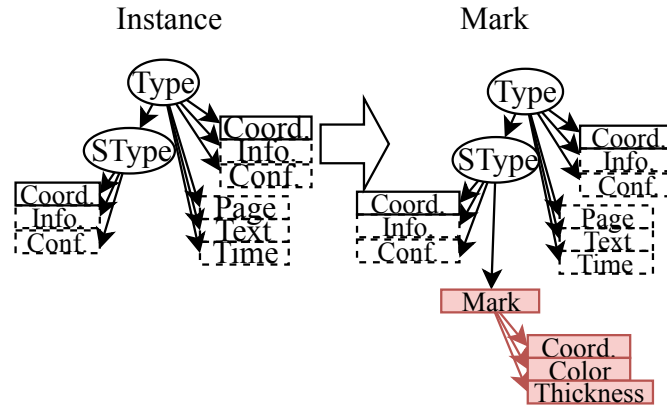


Figure 6.43: Content enricher: add shape

Read-only Patterns Subsequently, we differentiate types of operations on the physical and logical representations for patterns with read access to the message.

The Content-based Router and Message Filter patterns base their routing decision on a selected feature or object (e.g., product, OCR text) through reading the logical representation, while the physical multimedia data remains unchanged. Therefore, the features or objects within the multimedia data have to be detected. In the analysis, a separate Feature Detector was required, which reads the physical representation and returns a corresponding logical feature representation. Based on this logical representation, the Idempotent receiver and Message Validator patterns work in a read-only mode.

Access Patterns

The defined canonical data model is at the center of the user interaction. However, the user should mainly require knowledge about the actual domain model, and thus formulate all integration conditions and expressions accordingly. Subsequently, we identify and discuss common access patterns based on pattern arguments and the logical data access in [Table 6.8](#).

Feature Selector A *feature selector* specifies the interaction of a user with multimedia data through the logical and physical layers as shown in [Figure 6.44](#). The Content-based Router, Message Filter, Idempotent Receiver and Message Validator patterns as well as the correlation and completion conditions of the Aggregator (not shown), the object split condition of the Splitter, and the Content Enricher “mark object” operation are similar in the way they access the data and which multimedia artifacts they require. They require a Feature Detector to detect the domain object (by schema) and create the logical representation. Based on this information the object is selected and the corresponding operation is executed. For instance, the runtime system detects a human and his or her facial expression within an image, using the detector and creates the corresponding message model. Now, the user can configure the Splitter to select humans and add conditions for facial expressions, to select them using the selector. Once selected, the Splitter cuts the image according to the image coordinates of the selected feature and returns a list of sub-types in the number of humans and the corresponding cut images. The underlying integration runtime system takes the list of sub-types and images and creates new messages for each sub-type / image pair.

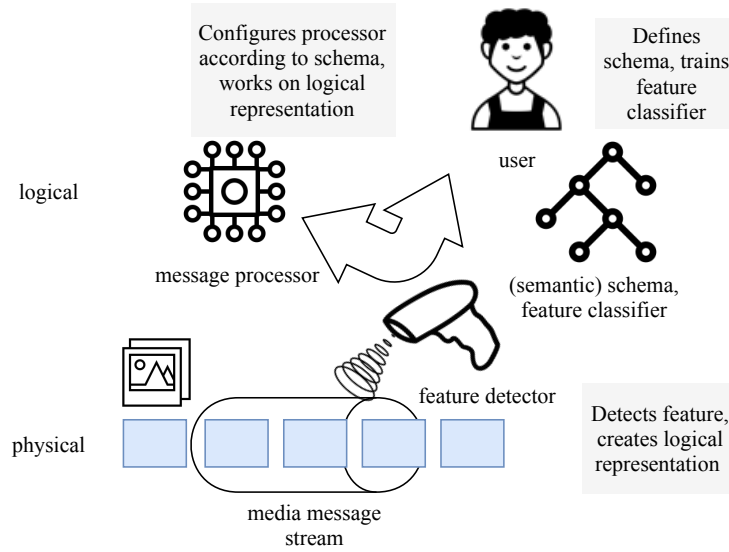


Figure 6.44: Feature selector (interaction)

Detector Region The creation of the defined message model through feature detection is computationally expensive, since it involves image processing. Each pattern in an integration process requires such a detect operation, if there is no detector prior to the pattern. Consequently, the detector can be built-in into each pattern or added as separate operation, before a sequence of several *read-only* patterns or patterns, for which the message graph can be re-calculated (e.g., for patterns like Aggregator, Splitter; cf. Table 6.8). For instance, for a fixed grid (with pre-defined cuts) and object splitters, the cut regions are known, and thus the properties of the model type can be computed (e.g., coordinates, color) and does not need to be detected. The Content Enricher mark operation appends the shape, color, coordinates and a new **mark** node in the graph, thus no detection is required. This way, all subsequent patterns after a detector share the same message property index and do not require further image operations. We call such a pattern sequence a Detector Region, as shown in Figure 6.45. Note that a new detection is required, if the message is altered in a way that does not allow for a re-calculation of the logical representation.

Parameterized Access Additional information is required for some of the patterns that change the physical representation like the Image Resizer, which requires scale parameters, or the shape and color information for the enricher and the translator. Therefore these patterns modify the feature vector directly (e.g., by changing the color or size). These changes are detected and executed on the physical multimedia object.

Pattern Realization and EAI System Architecture Extensions

We describe realizations for the specified logical and physical representations as well as the resulting architectural extensions to EAI systems. As EAI system, we chose the open-source Apache Camel [IA10] due to its broad support of the existing integration patterns (see Chapter 2) and its extensibility for new patterns and pattern realizations (e.g., multimedia).

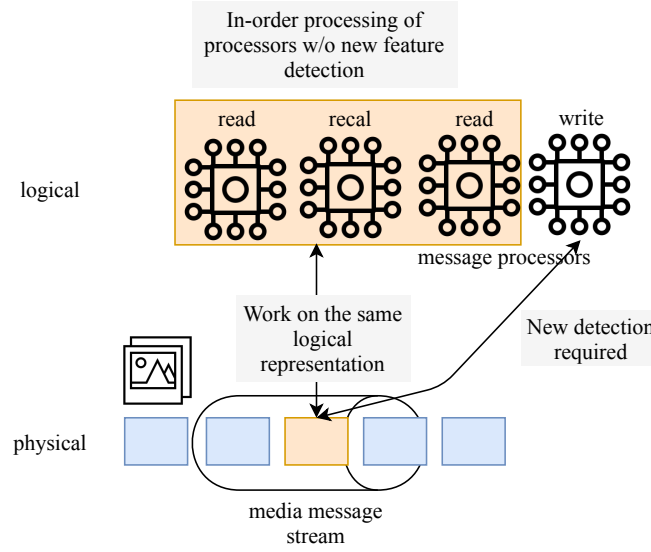


Figure 6.45: Detector region (improvement)

Pattern Realization For the pattern realization, we require the following decisions according to the definitions in Section 6.3.3. Besides Apache Camel as EAI system as part of the physical representation we chose JavaCV (i.e., based on the widely used OpenCV⁸ library) as open source multimedia processing system including their type converters. For the feature detection with JavaCV, we use Haar classifiers (e.g., for facial recognition [WF06]), which has to be trained with positive examples of a features (e.g., faces) as well as negative examples (i.e., arbitrary images without the feature). It is a cascading classifier, consisting of several simpler classifiers that are subsequently applied to an image or region and retrieve the coordinates as well as the object type that can be retrieved. All entering multimedia messages are processed by applying the classifiers.

The logical representation requires a semantic graph, for which we use the W3C Resource Definition Framework (RDF) semantic web standard, similar to metadata representation of images in Photo-RDF (cf. related work). For the schema representations, we chose ontologies similar to [TMM⁺16] that exist, e.g., for humans emotions (cf. virtual human ontology from Section 6.3.1) or real-world business products. For each ontology, a classifier is required for the physical runtime system. The selectors on the semantic RDF graph model are realized by SPARQL queries. The user interacts with the system (cf. Figure 6.33) by selecting a schema in the form of an ontology and adds the SPARQL query according to the access patterns in Section 6.3.3. If the system has built-in ontology / classifier combinations, only the query is added. Thereby only the domain ontology has to be understood. For parametrized access, our extensions from the physical representation have to be learned by the user.

EAI System Architecture Extensions The system aspects required for the pattern realization can be summarized to the conceptual architecture building-blocks in Figure 6.46. The physical system aspects include multimedia type converters and multimedia libraries. These libraries require feature learning components that learn classifiers for the semantic objects in multimedia data. The libraries evaluate the data according to the classifiers. For the mapping

⁸OpenCV, visited 5/2019: <http://opencv.org/>

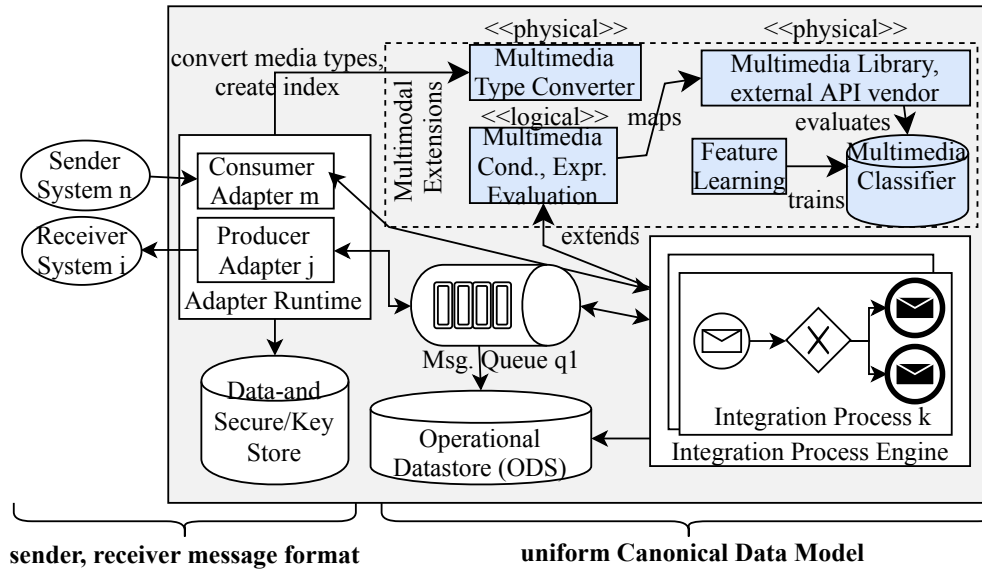


Figure 6.46: Conceptual EAI system architecture with multimedia extensions

between ontologies and classifiers, the Multimedia Cond., Expr. Evaluation contains the stored domain object models (e.g., ontologies; not shown) as well as the repository for user conditions and expressions (e.g., RDF statements).

For the evaluation of our approach, we extended the existing EIPs in Apache Camel by JavaCV multimedia processing and type converters as well as Apache Jena⁹ ontology, RDF representation and SPARQL queries.

6.3.4 Evaluation

In this section, we evaluate the pattern coverage and comprehensiveness of our multimedia integration pattern realizations from Section 6.3.3, and apply them to the motivating social media example in a realization and throughput study.

Pattern Coverage and Comprehensiveness

We aim for a compact model that is comprehensively usable with different image processing systems. Through the separation of the physical runtime and logical representation for user interaction, the comprehensiveness can be checked by its pattern coverage and by finding mappings to different image processing systems, while keeping the integration conditions and expressions stable. For this, we selected five multimedia processing systems / APIs from established artificial intelligence vendors: Google Vision API, HPE Haven OnDemand, IBM Watson / Alchemy Services (also used in [JBC⁺14] for textual analysis and semantic tagging), Microsoft Cognitive Services, and ABBYY, which focuses exclusively on OCR / Text.

Pattern Coverage. Figure 6.47 depicts an overview of the integration patterns that could be implemented by using the vendor systems. We added the open-source multimedia processing

⁹Apache Jena, visited 5/2019: <https://jena.apache.org/>

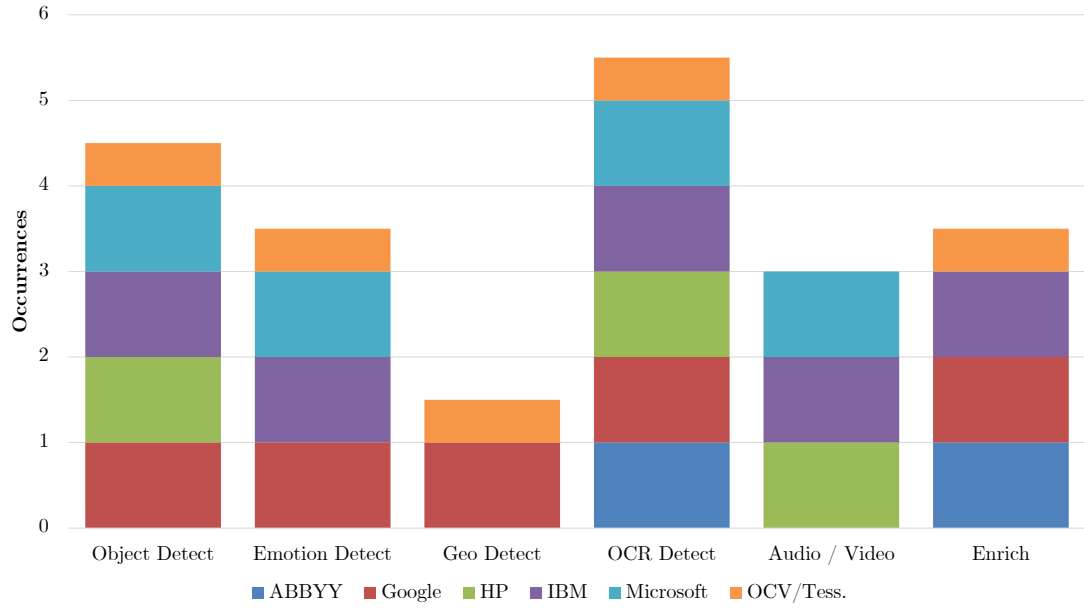


Figure 6.47: Vendor libraries as part of an integration system

libraries OpenCV and Tesseract — used to realize our reference system — for comparison as 0.5 (meaning partially supported due to implementation effort). From our pattern list (cf. Table 6.8), the Feature Detector (i.e., for object, emotion, geo, OCR) and the Content Enricher are explicitly covered. While all of the vendors offer object detection and enrichment capabilities in image or OCR texts (e.g., text, face and emotion detection) or geometrical shapes (e.g., Google, IBM), other operations are not supported (e.g., general message translation, resizing, aggregation, security). Therefore extensions in the form of custom media processing are usually required, which we realized as integration patterns using OpenCV.

Conclusion. (1) not all multimedia integration patterns can be represented with the capabilities of current vision APIs.

Comprehensiveness. The logical representation in our approach defines the following set of entities, for which a mapping to concepts from the vision APIs has to be found. All multimedia types have a domain object type that is derived from the domain model (ontology), the coordinates of the detected domain object within the medium, object metadata, and the probability for the correctness of the detection. For OCR, the actual text and a page number (for documents) is added, as well as the time (interval) in video streams. Table 6.9 sets our model entities into context to the vision APIs with respect to whether the concept exists and a mapping is possible. Since there was no information for ABBYY, and OpenCV, Tesseract OCR require explicit programming these systems are left out. Although the approaches are diverse in their terminology, provided features and focus areas, the analysis shows a broad coverage for the general model elements. Notably, the object type is represented as a list in HP, which we map to different feature vector dimensions. The metadata is mostly provided as name/value (e.g., in Google, HP) pairs or tags (e.g., in Microsoft). This information is only usable in integration conditions and expressions, if it can be mapped to the model domain. While all vendors add a likelihood or score to their

Table 6.9: Model coverage compared to vision API definitions

Vendor / Entity	Image/Any Object type	Coord.	Info	Prob.	OCR Page No.	Text	Video Time
Google	+	+	+/-	+/-	-	+	-
HP	+	+	+/-	+/-	+	+	+
IBM	+	+	+/-	+/-	-	+	+
Microsoft	+	+	+/-	+	-	+	+

(Exists and mappable +, Not supported -, partly exists or mappable +/-)

models, only Microsoft supports a fine-grain likelihood per feature vector dimension. In terms of compactness of our model, the page number in OCR documents could be left out, since it is only supported by HP. We decided to stick to it for convenience, in case it is available. In summary, our proposed model can be mostly mapped to concepts from heterogeneous vendors and appears compact in its representation.

Conclusion. (2) the multimedia model of our approach is sufficiently comprehensive to cover features of current vision APIs.

Case Study: Social Media Scenario

Now, we apply the presented approach on the motivating social marketing scenario and show its impact on the message throughput in terms of messages sizes and number of detected features (similar to Chapter 5). Therefore, we extended the open-source integration system Apache Camel introduced in Chapter 5 by the architecture components in Figure 6.46 to realize the multimedia patterns from Section 6.3.3. We discuss the trade-off between message sizes and throughput and compare the normal processing with the Detector Regions from Section 6.3.3. As indicated in Figure 6.32, the selector region comprises the Content-based Router, Message Filter, translator, splitter and the enricher, for which the logical representation can be re-calculated. For this case study we assume image message workloads from the social media Open Images Project data set based on Google Flickr [KRA⁺16], generated by the EIPBench benchmark from Chapter 5, which we extended by multimedia data and semantic query capabilities. For instance, for filtering image messages without a human, we use the SPARQL ASK query `ASK{FILTER NOT EXISTS {?s prefix:hasFace ?o}}`, evaluated using the Apache Jena library, which returns a Boolean that is mapped to the filter runtime component. Similarly, the selector for splitting image messages with multiple humans to single messages with only one is defined as `SELECT ?o WHERE {?s prefix:hasFace ?o}`, returning a list of feature vectors and their coordinates that are then cut and routed separately by our splitter extension. The measurements are conducted on an HP Z600 workstation, equipped with two Intel X5650 processors clocked at 2.67GHz with a 12 cores, 24GB of main memory, running a 64-bit Windows 7 SP1 and a JDK version 1.7.0 with 2GB heap space.

Figure 6.48 shows the message throughput of the implemented scenario for an increasing number of features detected in the images and message sizes. Notably, the number of features has less impact on the throughput than the message sizes (corresponding to the image’s resolution). Hence, an image resizer or splitter pattern could be used to improve the message throughput, as long as the features can still be detected. For the detector region measurement, a Feature Detector pattern is inserted before the content router. All subsequent patterns are contained in the detector region, and thus do not need to detect the features again. Figure 6.49 shows the message throughput of the scenario for mixed workload messages size intervals of 1-50 kB, 50-100 kB and 850-900 kB messages with one, eleven, and seven features, respectively (numbers

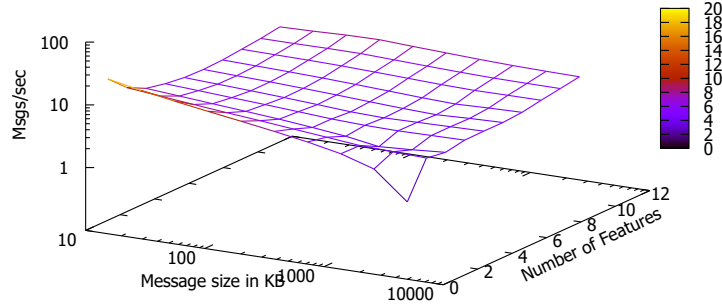


Figure 6.48: Multimedia message throughput of the social marketing scenario

from the image dataset). When using the detector region the throughput increases by 2.5% and 10.3% for the smaller message sizes, however, only 0.2% for the bigger message size. While the normal processing is limited by the pattern with the least throughput, the detector region is limited by the throughput of the detector. For larger images the normal and detector region throughput are similar, due to the increasing costs of the feature detection compared to the other pattern processing. Therefore only improved image processing techniques (out of scope), or parallel sub-process execution improve the throughput.

Conclusions. (3) EIPBench suitable foundation for benchmarking multimedia patterns, however, requires extensions in the form of multimedia messages; (4) message size vs. throughput trade-off depends on the size of the multimedia message (and not the number of features); (5) the Detector Region improvement is limited by the used Feature Detector; (6) further message processing techniques have to be evaluated.

6.3.5 Conclusions

In this section, we address the fundamental topics of multimedia application integration and provide a solution toward a more standard user interaction and configuration of multimedia scenarios as requested by [AAB⁺17]. We conducted literature and application studies to identify industrial and mobile scenarios requiring multimedia integration, which resulted in a list of patterns (mostly in [HW04, RMRM17]) relevant for multimedia processing (cf. questions MM1+2). For the underlying integration semantics of these patterns we defined multimedia pattern realizations, to which we mapped the operations from the analysis (cf. MM3). We outlined a compact logical multimedia representation — toward a uniform user interaction that takes the image semantics into account — and evaluated the compactness and comprehensiveness by comparison with a selection of vision API vendors. For multimedia processing, the common architecture of EAI has to be extended (cf. MM4). We discussed the fundamental components (cf. MM5) and conducted a case study based on the motivating social marketing scenario (cf. MM6). The evaluation showed the suitability and comprehensiveness of our approach (cf. conclusion (2)), applicability of our benchmark (cf. conclusion (3)), limitations and necessary improvements of current image processing solutions and message processing (cf. conclusions (1), (5) and (6)), and an interesting trade-off (cf. conclusion (4)). Thereby we identified further challenges targeting

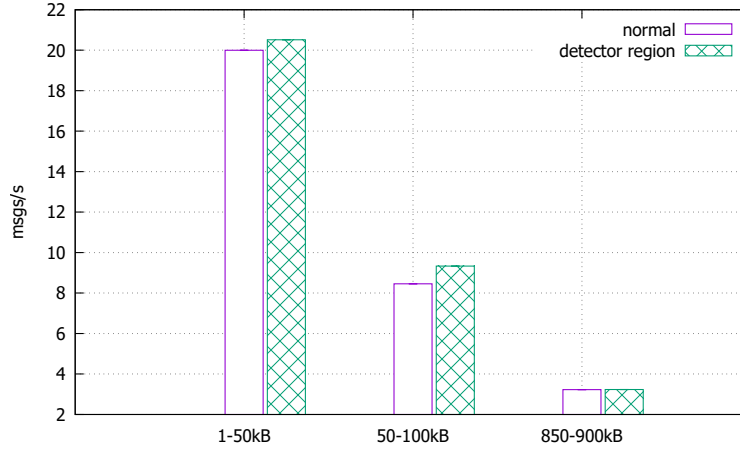


Figure 6.49: Message throughput of the social marketing scenario

more efficient message processing (e.g., read / write optimizations like message indexing, process optimizations), interactions with non-standard message transformation operations (e.g., image resizer), new processing types compared to the EIPs (e.g., streaming [Section 6.2](#)) and definition of visual integration scenario editors (e.g., query by sketch / visual queries).

We conclude that multimodal message processing allows for covering current variety requirements as well as for the development of novel applications and scenarios that are not possible today. While there are still many interesting challenges, when working with streams of multimedia data integration (e.g., cf. Abiteboul et al. [[AAB⁺17](#)]), we conjecture that multimodal processing and integration will become mandatory in the future.

6.4 Related Work

We set the data-centric, hardware and multimedia pattern solutions into context of related work.

6.4.1 Data-centric Pattern Solutions

The aspect of moving EIP semantics and processes to the database (i.e., case *(C1)* in [Section 6.1](#)) has been mentioned as future work by [[BLN86](#)] in the context of process integration, however, was not addressed so far. We picked up the topic in our position paper [[Rit14c](#)], which discusses the expressiveness of database operations for integration processing on the content level. However, there is some work (mostly) in related domains, which we discuss subsequently.

Message Queuing

In the domain of declarative XML message processing, [[BK11](#)] defines a network of queues and XQuery data processors that are similar to our (persistent) states and transitions. This targets a subset of our approach (i.e., persistent message queuing; case *(C3)*), however, it does not cover *(C1)* integration semantics as relational database processes, which we target for analytic and business applications. In [[GM03](#)], a Java Message Service (JMS) like message queuing engine is designed into the database, which allows for enqueueing and dequeuing messages, thus addressing

case (C2). This work is complementary to our approach, which uses the JMS endpoints as sources for EIP processing.

Business Processes

The work on executing business processes on the database, using an external process engine, evaluates nested views on the data and returns the results to the process engine (e.g., database engineering applications) [HMWMS87]. More generally, [Her03] addresses the functional integration of data in federated database systems. Similar to our approach, data and control flow have been considered in business process management systems [VSS⁺07], which run the process engine optimally synchronized with the database. However, the work exclusively focuses on the optimization of workflow execution and does not consider application integration semantics on the database server level. In our work, we consider integration operations that are executed directly on the database, while no data is passed to a remote process engine or integration system.

Data-Intensive and Scientific Workflows

Based on the data patterns in workflow systems described by Russel et al. [RTHEvdA05], modeling and data access approaches have been studied (e.g., by Reimann et al. [RS⁺13]) in simulation workflows. The basic data management patterns in simulation workflows are ETL operations (e.g., format conversions, filters), a subset of the EIPs, which can be represented among others by our approach. The (MapReduce-style) data iteration pattern can be represented by scatter-gather or splitter-gather.

Data Integration

The data integration domain uses integration systems for querying remote data that is treated as local or “virtual” relations (e.g., Garlic [HKWY97]) and evolved to relational logic programming, summarized by [CHK09]. In contrast to remote queries, we extend the current EIP semantics in the database for EAI.

6.4.2 Hardware Pattern Solutions

We are not aware of other work on implementing EIPs on FPGAs. However, there is a rich body of work in related domains (i.e., query-, complex event and stream processing, message queuing), relevant for our work. Especially, the lessons learned on system design (e.g., parallelism, automata for format processing), the identified trade-offs between synchronous and asynchronous designs, and system integration aspects (e.g., FPGA in the system’s data path) have influenced our work. We summarize the related contributions in Table 6.10 and subsequently discuss the approaches.

Query Processing

Several lessons learned on query processing are also relevant in the context of EIP. The design of the industrial solutions (e.g., “Netezza Performance Server”¹⁰), which consists of a number of “snippet processing units”, each tightly coupled with network CPU and FPGA, and Kickfire’s *MySQL Analytic Appliance*¹¹ with a so-called “SQL Chip”) do only give limited insight into their design decisions. However, both systems appear to use FPGAs primarily as customized hardware, with circuits that are geared toward very specific (data warehousing) workloads, but

¹⁰Netezza Corp, visited 5/2019: <http://www.netezza.com/>

¹¹Kickfire, visited 5/2019: <http://www.kickfire.com/>

Table 6.10: Lessons learned from related work

Design Decision	References	our approach
parallelism: systolic vs. MISD	[Cas05], [MT10]	systolic for higher throughput
automaton: (non-) deterministic	[ADGI08], [EHI10], [FU80], [GWC ⁺ 07], [MVB ⁺ 09], [NSJ15], [SJL ⁺ 10], [VLB ⁺ 10], [WTA10]	deterministic format handling
synch vs asynch circuits	[MT10]	synchronous with flow control
configurable clocks	[MT10]	- (for asynch. designs)
back-pressure	[Cas05]	for flow control
avoid long distance streams	[Cas05], [MTA09a]	system part
avoid deep logic	[Cas05]	system part
FIFO buffer	[EHI10, MT10, MTA09a, MTA09b]	BRAM for reliability

are immutable at runtime. In our approach, we aim at exploiting the configurability of FPGAs by compiling integration operations from an arbitrary workload into a tailor-made hardware circuit.

The research on the *Glacier* query to hardware compiler provided valuable lessons with respect to design decisions [MT10, MTA09b]. As in [Cas05] several types of parallelism are discussed (cf. Table 6.10): e.g., systolic (i.e., pipeline-chain: good scalability even across chips) and MISD (i.e., tree: long signal path), for which we follow their lead to higher throughput systolic parallelism. In contrast to *Glacier*, we do not use an asynchronous stream design (i.e., with lower latency, less flip-flop registers), but a modular synchronous stream design with re-usable logic. Hence, our design also does not make use of configurable clock frequencies, which is crucial in an asynchronous design. The frequencies only vary for integration adapters (e.g., TCP, UDP) and the integration processing steps. We also use finite state machines, which naturally translate into FPGAs (inspired by [FU80]), for message format conversions — in our case hierarchical data in the form of JSON messages. The asynchronous design approaches in [MT10, MTA09a, MTA09b] requires FIFO buffers for synchronization. We use FIFOs to implement flow control in the form of back-pressure [Cas05].

Complex Event and Stream Processing

There is a lot of research on stream queries on hardware, hence we only summarize the directly related work. Most influential work was conducted in [MTA09a, TW13], which analyzes the potential of FPGAs in the domain of data processing using sliding window operators for a median operator using a sorting network. The experimental analysis showed promising results compared to a small on-board PowerPC 405 processor. Compared to our synchronous approach, an asynchronous design fits well to the sorting network (cf. Table 6.10).

The work on event detection using regular expressions [WTA10], efficient pattern matching [ADGI08] non-deterministic finite automaton design, complex event processing [GWC⁺07, SJL⁺10], streaming system [NSJ15, VLB⁺10] and XML / XPath evaluation [MVB⁺09, MLC08] influenced our work on the message format handling. The latter also sketches the idea of an internet protocol router on the packet level, setting the addresses according to a routing table [MLC08], similar to a recipient list. Since no packet performance measurements were published (e.g., for throughput or latency), it remains a mere design sketch. All of these approaches use deterministic or non-deterministic state machines to represent user-defined conditions on the hardware. For the hierarchical format handling (e.g., JSON messages), we define a general, but resource-reduced way to represent the data (e.g., compared to approaches like [MLC08]) to spare

resources for the integration logic.

Publish/Subscribe and Queuing Systems

Message Brokers are used complementary to application integration systems. The most well-known commercial solution is the “Solace Message Router” [Sol16], which implements a JMS-like topic and queue processing as well as high-availability and disaster recovery on FPGA hardware. The comparison to one of the most popular software brokers shows superior message throughput rates on the hardware¹².

While Publish/Subscribe systems mostly route data without looking into their content — giving them an edge over application integration systems — there is some work on content-based routing in the sense of [HW04] using FPGAs. [EHI10], for example, proposes an architecture, which adds FPGAs to each message broker that implements XML / XPath message routing using a complex memory based XML parser (using content-addressable memory, and FIFO buffers) and a matcher leveraging the dual port memory for concurrent read / write. This work is comparable to the content-based routing and JSON processing approaches presented in this chapter. However, our approach manages to process JSON with much lower resource consumption. We only optionally use FIFO buffers based on on-chip BRAM for back-pressure handling.

In general, back-pressure is used for flow control holding off senders to further transmit data until the resource is available. We use back-pressure in our message processing pipeline to avoid scrambled data and allow for TCP-based flow control via the integration adapters. This idea is based on [Cas05], which defines a systolic message queuing system that uses back-pressure based on a wire protocol for synchronous communication (cf. Table 6.10). Furthermore, we should try to avoid long distance pipelines or streams and deep logic in processes [Cas05] in the system part of our pipeline design (cf. Table 6.10). However, user-defined code might violate this rule.

Hardware-accelerated EAI Processing

The EIP authors admit that the current foundations are defined for an asynchronous processing style, and a definition for synchronous streaming is missing [ZPHW16]. In this section, we fill this void and enumerate some of the most relevant EIPs as identified in Chapter 5 and specify semantically correct streaming semantics. Based on these semantics, we are able to define three template classes that are sufficient to represent all of the patterns and also to synthesize them to hardware. Overall, we are able to implement the complete integration system on hardware as ISoC (cf. Figure 6.16). In the experiments we test different aspects special to integration processing (cf. Chapter 5) that have not been tested on hardware before.

6.4.3 Multimedia Pattern Solutions

There is a large body of work produced by research conducted in multimedia processing in venues like ACM Multimedia, ACM Multimedia Systems, IEEE Multimedia, IEEE Transactions on Multimedia. While most of the work targets the image processing and feature extraction foundations — complementary to our work — we subsequently set the relevant work into the context of our solutions for the challenges in Section 6.3.

User Interaction and Interoperability

The work on queries on multimedia databases and streams targets multimedia data representation and queries, similar to our approach. In this context, many user interaction approaches focus on

¹²Solace vs. Kafka, visited 5/2019: <https://dev.solace.com/kafka/solace-kafka-comparison-summary/>

the media’s metadata (e.g., name, type, publisher) and not on the actual information within the image (e.g., [Gro97]). Commonly, this metadata is accessible by standards like Photo-RDF¹³, which represents the semantic information in images using RDF. While the metadata denotes textual processing, we focus on the multimedia processing. In our realization (cf. Section 6.3.3) RDF is used to represent multimedia semantics.

Further known related work targets the retrieval of multimedia information from (distributed) databases [CDPV07]. The multimedia semantics are represented by semantic attributes based on extended generalized icons with a logical and physical representation on a database. While our approach separates these different representations as well, [CDPV07] targets extended normal forms and functional dependencies between different attributes and does not define user interaction with the multimedia semantics on a business application-relevant feature level that could be used for message processing. More recently, Lin et al. developed a similarity query mechanism for images [LOON01] in the area of multimedia queries on multimedia databases. While no query syntax is provided, the operator could be used to formulate decisions based on image similarity.

System Architecture

Our evaluations in Section 6.3.4 identified the need for a change in the common EAI system architectures [Lin00]. While we collected the components required for EAI, we consider the existing work on multimedia processing system architectures complementary to our approach. For instance, there are several systems for parallel media processing. Processing large video streams is handled using distributed resource management in [WB97]. Similarly [Ama07] introduces a dataflow process network of actors, connected by FIFO queues, that process multimedia data and fire events based on rules according to a domain-specific metamodel.

The OCAPI system [CT93] was developed for the semantic integration of programs using a knowledge base approach including a query processor and reasoner over image data for syntactic and semantic integration. The knowledge base is used similarly to the ontologies in our approach, giving a semantic context, while the query language works on the image primitives, thus rather technical. No standard query mechanism is provided, however, the R^* -based indexing technique might be considered for optimizing the image message processing. The EADS *WebLab* project denotes a service oriented architecture for developing multimedia processing applications [GBB⁺08]. It neither targets integration processes, nor the EIPs, but defines an exchange format based on a *Media Unit* to solve the problem of semantic interoperability between the information processing components. Similar to our approach, the media types image, OCR text and video are distinguished, coordinates are specified, and a temporal segment is defined for videos. The query approach is based on a proprietary model.

In the related business workflow domain, [PCM⁺07] define the ARIA system with quality of service (QoS) guaranteeing multimedia workflow processing. The defined media filter and fusion multimedia operators in ARIA are similar to our message filter and aggregator patterns. However, the processing is limited to simple 1:1 and fork 1: n workflows.

Multimedia Processing

The recent survey on event-based media processing and analysis addresses approaches and challenges in the domain of multimedia event processing considering audio, video and social events [TMM⁺16]. Events are human actions or spatial, temporal, relationship state changes of objects, which are mostly represented in event or situational calculi as well as contextual ontologies. Similarly, we use domain-specific ontologies to represent the message schema in our

¹³W3C - Photo-RDF, visited 5/2019: <http://www.w3.org/TR/photo-rdf/>

realization (cf. [Section 6.3.3](#)). The application analysis is based on mobile apps like Flickr. The challenges name the discussed “Semantic gap” as well as a “Model Gap”, which is the trade-off between an event model’s complexity and its detection performance, which we discussed as part of our evaluation (cf. [Section 6.3.4](#)).

Overlapping with the challenges of interoperability and system architecture, [\[CDG05\]](#) defines an interoperable interface for distributed image processing using grid computing based on CORBA object exchange. However, the interface and the operations target low-level image processing (e.g., for point and image arithmetic operations). [\[JDM⁺01\]](#) defines a system that segments and indexes TV programs according to their audio, visual, and transcript information. However, the approach uses a “Media-To-Text” preprocessing, while subsequent operations are then executed “Text-to-Text”. More recent work on parallel processing of multimedia data mining for computer vision uses map-reduce techniques [\[VC15\]](#) or cloud-based Hadoop systems [\[ZSSZ14\]](#). The solutions provided target the efficient multimedia program execution on a lower level (e.g., edge detection and segmentation), which could be considered for more efficient message processor implementations.

6.5 Discussion

The pattern composition formalism, the optimization strategies and the EIPBench pattern benchmark denote valuable foundations in the form of design science research (DSR) artifacts for the practical impact study of new technological trends. The study targets an answer to the stated research question on more efficient realization of integration patterns and their compositions that we call pattern solutions (cf. RQ3-1: “*Which related concepts and technology trends can be used to improve integration processing and how can this be practically realized?*”). The question and its solution-specific sub-questions are answered by the pattern solutions and their evaluations in this chapter that again denote contributions in the form of DSR artifacts:

- Pattern realization concepts for micro-batch, vectorization (volume), dataflow / stream, specialization (velocity), and multimedia data (variety) processing styles (cf. pattern templates), and their compositions;
- Novel integration system design concepts: a database transition system (DTS), an integration system on a chip (ISoC), and a multimedia integration system (incl. evolving integration system architecture components);
- Instantiations of the concepts and system designs in the form of prototypes: DTS on a (relational) BDMS, ISoC on reprogrammable hardware, multimedia integration system as extension of an existing system.

These answers thus provide conceptual foundations as well as prototypical implementations of pattern compositions that will allow for the development of innovative and efficient integration systems tailored for the needs of current and future application integration. In particular, we recall that the “volocity” challenge C5 (i.e., volume, velocity in [Section 1.2](#)) on the integration scenario level was partially addressed by optimization strategies in [Chapter 4](#). The optimizations improve the comprehension of integration scenarios through reducing the modeling complexity, and we showed that it helps to reduce the data volume through data reduction and improve the processing velocity (i.e., message throughput, latency). The improvements were achieved by rewritings on an abstract integration scenario level, for which we proved correctness-preserving properties. In this chapter, we studied improvements on the system level by applying new technology trends to EAI processing. Notably, the big data management and processing platforms together with the *vectorization* or *micro-batch* technique not only show promising results for high volume message processing, but can also be seen as a natural implementation of timed db-nets close to the data

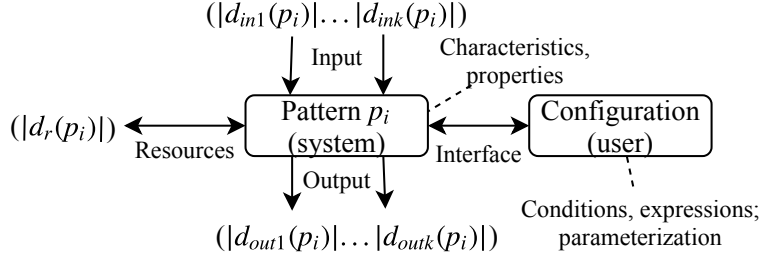


Figure 6.50: Abstract pattern building block

in the form of a database transition system. Similarly, for fast message processing we found that a specialization towards a network-attached, hardware ISoC denotes an unmatched contemporary solution. When combined with the optimization strategies, even better message throughput results could be achieved for some of the optimizations. For the variety challenge C6, we proposed a solution that covers all aspects from the configuration of integration scenarios to the runtime and showed some improvements.

While each system serves a special purpose by addressing one of the challenges, combining them into one system or defining their interaction would be practically preferable. For a combination, either the ISoCs need to be improved with respect to high data volume processing (e.g., for multimedia data) or the throughput vs. latency trade-off in BDMS needs to be solved. However, neither conventional reprogrammable hardware nor BDMS have expressive means to process multimedia data. Consequently, it is more likely that each system is used for their dedicated purpose and simply interacts with the others. For example, the DTS can store and forward multimedia data that is routed and transformed by the multimedia integration system. Smaller portions of the data could be exchanged via the fast ISoC interconnect.

Based on these more practical considerations, we subsequently reflect on pattern and compositional, formal foundations (cf. [Chapters 2 and 3](#)) and optimization strategies (cf. [Chapter 4](#)) in the context of the pattern solutions. More precisely, we elaborate on the patterns as building blocks of an EAI system, discuss the findings of the realizations with respect to the execution semantics, revisit the applicability of optimization strategies, and assess the architecture evolution of EAI systems. Finally, we summarize and discuss further limitations and open research challenges.

Patterns as Basic Building Blocks The foundational work in [Chapter 3](#) not only provides formal building-blocks of patterns and their compositions (cf. [Section 2.1.3](#)) for a modeling language as well as when implementing integration systems for the first time, but also underlines the value of patterns as abstraction of integration domain concepts. Although there is a multitude of different patterns and even “break-outs” like user defined functions, where additional functionality is needed, the patterns adhere to a finite set of domain concepts (e.g., quality of service, security), fundamental pattern characteristics and their structural properties (see [Chapter 3](#)) that immediately apply to every new pattern. While other domains (e.g., artificial intelligence, machine learning) strive to find suitable abstractions to structure and evaluate their research, the practical work on the pattern solutions in this chapter show that the (potentially evolving) integration patterns denote a suitable abstraction of EAI or even application programming interface (e.g., similar to MapReduce [\[DG08\]](#) in the data processing domain).

This abstraction essentially designates a *type system* for the development of a sound and comprehensive integration programming language, which we leave for future work. Thereby,

the template-based representation already used in [Chapter 3](#) for the translation of the single types (i.e., integration patterns) and their compositions into a runtime system (e.g., CPNTools models) turned out to be beneficial during the implementation of the pattern solutions in this chapter. [Figure 6.50](#) gives a conceptual view on a general pattern template, which combines the abstract operator used for the cost model considerations in [Figure 4.11](#) (on [page 153](#)) with pattern configurations (e.g., [\[Sch10, Rit15a\]](#)). In addition, a separation of the part that is strictly pattern-specific from the configurable, scenario or user-specific part was beneficial in all pattern solutions. The pattern-specific part is based on the *Pattern Characteristics*, which allows for a further combination of implementations by their category. In our pattern solutions, this part could be tailored to the runtime (e.g., in SQL, PL/SQL for DTS or in HDL for reprogrammable hardware), since the pattern semantics are not subject to (frequent) change. The *Configuration* part is invoked through interfaces, which allow for the invocation of user-defined functions (e.g., conditions, expressions, programs) and parameterization (e.g., endpoint configuration for integration adapters and external resources, timings). This part is frequently changed, and thus provides flexible parameterization for frequently changing use cases and scenarios.

Example 6.17. The Message Translator and Content Filter patterns are in the same category with respect to their characteristics from [Chapter 3](#) (i.e., data-only), and thus both are implemented as the same solution templates in the evaluated pattern solutions: database pattern template in DTS (not shown), expression template in ISoC (cf. [Figure 6.25\(a\)](#) on [page 234](#)) and multimedia pattern template with the same physical and logical representation (cf. [Table 6.8](#) on [page 258](#)) in the multimedia integration system. ■

Pattern Execution Semantics The execution semantics of the single patterns and their compositions are well-defined through the formalism introduced in [Chapter 3](#). However, the integration patterns were originally described for single message processing (not batch processing) and without the knowledge about more recent technological trends like stream or multimedia data processing [\[ZPHW16\]](#). Consequently, the work on the pattern solutions addressing these trends can be understood as an experimental evaluation of the suitability of these trends with respect to the original execution semantics of the patterns.

The execution semantics of database-centric integration processing is conceptually the closest to timed db-nets. However, the design decision to represent the complete integration processing (i.e., control, data logic and persistence layers) within the database as DTS led to changes in the transactional processing (i.e., everything executed within a transaction) and performance improvements, when moving from single to batched message processing. The latter raised questions about changes of the pattern characteristics and semantics triggered by the specific instantiation. As described in [Figure 2.5](#) (on [page 27](#)), an instantiation might lead to additional information that is useful for the pattern identification phase and helps to decide, whether the variance is sufficient for a new or a variant of the existing pattern.

The defined dataflow and streaming pattern solutions developed in this thesis use a different execution engine, however, ensure the actual pattern semantics. While the (reprogrammable) hardware programming model with synchronized clocks fits well to the time aspects in timed db-nets (i.e., requiring a clock for the token aging), the study mostly focused on transient message processing, and thus would not consider the timed db-net persistent storage semantics.

The multimedia patterns are already part of the pattern catalog, and thus could be simulated by timed db-nets, if it were not for the binary, multimedia data and the semantic knowledge representation. However, the foundation of timed db-nets is partially motivated by *Data-Centric Dynamic Systems* (DCDS) [\[BHCDG⁺13\]](#), which means that it partially inherits the logic, and thus all DCDS extensions or variants can be naturally mapped to timed db-nets. Therefore, we recall the timed db-net design, which uses a FOL-based data logic layer to mediate between the

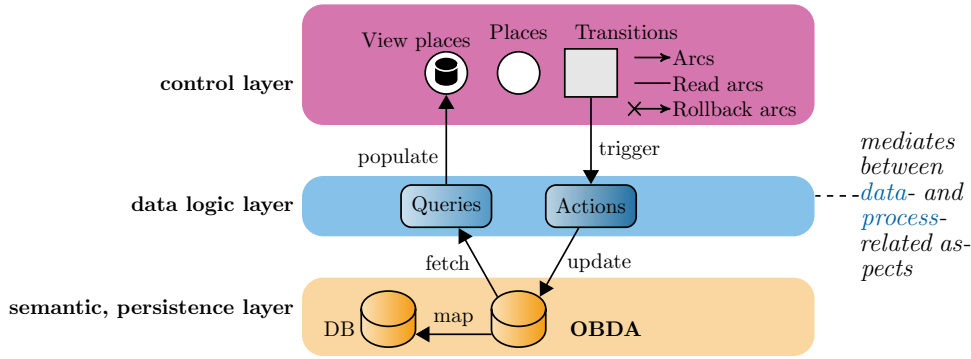


Figure 6.51: Timed db-net with Ontology-based Data Access (OBDA)

control and persistence layers. The representation and access of the multimedia data semantics requires a *knowledge and action base* (KAB) [HCDG⁺12, HCM⁺13], which use description logic, and thus support queries over semantic knowledge representation (e.g., semantic web queries). The theoretical grounding comes from *semantically-enhanced data-aware processes* (SEDAPs) [San16], which is reducible to DCDS, and allow for the verification of sophisticated FOL temporal properties, and addresses the verification using existing techniques developed for DCDS. SEDAPs subsumes DCDS by enhancing a relational layer constituted by a DCDS-based system with an ontology, constituting a semantic layer. Essentially, in SEDAPs, the ontology captures the domain of interest, in which a SEDAP is executed. Additionally, it allows for seeing the data and their manipulation at a conceptual level through an *Ontology-based Data Access* (OBDA) system [CDGL⁺09, RMC12], reflecting the relevant concepts. Consequently, as depicted in Figure 6.51, the control or process layer remains the same, and thus only the queries and actions on the data logic layer has to be adapted for all multimedia pattern realizations in timed db-nets. The data logic layer needs to be adapted to mediate between the persistence layer, which stores the initial data of the system that will be manipulated by the net defined in the control layer and an OBDA, including (the intentional level of) an ontology, a relational database schema, and a mapping between the ontology and the database.

Applicability of Optimization Strategies During the evaluation of the pattern solutions, some of the optimization strategies from Chapter 4 were already studied in the context of real-world scenarios, where applicable. The applicability of these strategies potentially depends on the capabilities of the runtime system. For example, the *Parallel Process Re-Scheduling* from [RFRM19] is not relevant on reprogrammable hardware, since the hardware configurations do not need to be scheduled in a software-like way. Subsequently, we assess the fit of the found optimization strategies OS-1 to OS-3 to the evaluated runtime systems.

The process simplifications (cf. OS-1) mainly target the modeling complexity reduction of a pattern composition, and have a positive effect on the processing latency. Some of the optimizations in this category additionally help to make the process more resilient, since less unused resources are allocated. For example, the combine sibling patterns optimization helped to reduce the amount of resources on the reprogrammable hardware. While this strategy makes all pattern solutions more efficient, the reprogrammable hardware solution benefits the most from the more concise compositions. Due to the limited hardware resources, the freed parts can be more effectively used for a higher throughput (e.g., through parallelization), higher volume processing per composition by more resource assignment, or cost-efficient partitioning by sharing more compositions on one chip (even in the presence of virtualization techniques [PCC⁺14, CCP⁺16]).

The data reduction strategy (cf. OS-2) provides several optimizations that allows for the

reduction or offloading of data. While the database systems naturally deal with high volume data, the other two solutions greatly benefited from optimizations like early-filter, early-mapping and early-split. In all cases, the resulting effect was an increased message throughput. For example, the message throughput of the multimedia message processing highly depended on the data size.

The parallelization on a pattern (sub-) composition level (cf. OS-3) is very beneficial to reach higher message throughput, however, extremely resource intensive. Due to the exceptionally promising results on reprogrammable hardware, such an optimization might only be needed in few scenarios. However, for the software-based solutions, parallelization seems to be one of the only means for high throughput solutions (e.g., container-level scaling [VRMB11], serverless computing [MB17] including function as a service¹⁴) — apart from minor improvements through more efficient implementations. In current software solutions, the performance vs. cost or energy trade-off (e.g., [FLP⁺07]) seems to be largely focused on performance in academic research.

Integration System Architecture Evolution The study of the pattern solutions showed that EAI semantics can be transferred to concepts and systems from related domains, which already solve many of the new challenges identified. For example, big data management systems handle large data volumes and dataflow systems like Flink and Spark allow for efficient micro-batched as well as stream processing. Hence, future EAI solutions will have to consider integration semantics as part of these platforms and decide on the contribution of extensions or building a completely new software runtime. In the related database domain low-level machine programming (e.g., [Neu11]) promise efficient processing and foster considerations on the suitability of JVM-based runtimes with memory management seemingly not built for messaging.

The most promising results were those of the specialization on re-programmable hardware, which not only excels in message throughput and latency, but also reduces the energy consumption. We already discussed the difficulties in developing and productively using the pattern solution as well as challenges like virtualization (cf. [Rit17b]). Despite these challenges, our ISoC solution sets new standards in terms of performance and sustainability, which has to become part of the design decision process soon. Especially the execution close to the network might become an imperative for integration systems for more efficient processing. This could go hand in hand with promising results on Publish/Subscribe systems on software-defined network (e.g., [TKBR14, BTK⁺17]) and multi-purpose network attached processing (e.g., [ZLM⁺16, EJ17]).

The efficient and multimodal processing will become even more important in the context of emerging applications like Connected Car Telemetry and multimedia data that is processed online on the vehicles (e.g., for object detection; requiring low energy, high throughput systems), and moreover, requires new architectural components like for machine learning or event-driven, reactive additions (e.g., on mobile platforms [RH18]). In this way, the new variety challenge of multimodal processing bridges to machine learning and semantic web as well as mobile computing (generating most of the multimedia data), and event-driven and microservice architectures. These bridges will have to be built into the integration systems or interface with specialized systems or service offerings.

Limitations Limitations of our pattern solutions concern the selection of the challenges and the technological trends. The most relevant and promising challenges in the context of emerging scenarios were selected. Out of the vast number of technologies available from related domains, the most directly related were chosen: big data volume (big data management system), speed of change velocity (dataflow streaming), and multimedia data variety (semantic knowledge representation). Nonetheless conducting further studies and comparisons with other technologies and systems will help to conceptually and experimentally explore the relatedness of the different domains and

¹⁴Serverless Architectures, visited 5/2019: <https://martinfowler.com/articles/serverless.html#unpacking-faas>

might lead to interesting conclusions about the necessity of their separate treatment. Moreover, combining the three proposed solutions into one would be preferable, however, conceptually and practically difficult due the different purposes they were built for (volume, velocity, variety).

Impact The early impact of the pattern solutions on research and industry is already remarkable. The resemblance of the DCDS and database transition system could eventually bridge between the process and database communities, especially due to the focus on *responsible programming* (e.g., as recent work on SQL validation [GL17] indicates). Moreover, the new format of ACTIVE workshops on modern hardware combined with the top middleware [SP17, Rit17b] and database [DBL17, DBL18] conferences indicates a focus on more efficient data processing through specialization with reconfigurable hardware.

In industry, more and more software business and technology companies like SAP or vendors of the emerging applications discussed in this thesis like NASA or Daimler collaborate with hardware technology vendors like Solace¹⁵ or Intel¹⁶. Furthermore, the multimedia pattern solutions were picked up early by industry and incorporated in several showcases. For example, the image processing integration capabilities¹⁷ were shown in drone logistics scenarios and a keynote featuring “The Martian” at SAP TechEd and SAPPHIRE conferences that caught a lot of attention and sparked customer interest.

¹⁵Solace, SAP and NASA, Daimler partner profiles, visited 5/2019: <https://solace.com/customer-profile/sap>, <https://solace.com/customer-profile/nasa>

¹⁶Intel and SAP partnership, visited 5/2019: <https://www.intel.com/content/www/us/en/big-data/partners/sap/overview.html>

¹⁷SAP CTO’s “The Martian” keynote incl. “Communicate with Earth”, visited 5/2019: <https://news.sap.com/2015/11/how-saps-bjorn-goerke-became-the-martian/>.

Chapter 7

Conclusions

Contents

7.1 Summary	281
7.2 Outlook	285

In this final chapter, we summarize the content and contributions of the thesis and give an outlook on future work.

7.1 Summary

In the past decades we have seen an accelerated emergence of trends in business, technology and society that became significant for our daily life (e.g., cloud and mobile computing, business networks, social media). In particular, the trends led to an even more digitalized, modularized, and multimedia world, in which the integration of applications and processes in the form of integration scenarios is of fundamental importance. Moreover, the trends foster new operation models outside of the scenario creator's control and in many cases closer to potentially non-technical end users, whose lives become tailored around correctly integrated applications and devices.

The resulting demand for *trust* in Enterprise Application Integration (EAI) solutions, connecting the different participants (e.g., businesses, applications, devices), as well as the required *efficiency*, matching more demanding message processing requirements, goes beyond the capabilities of current EAI systems and solutions. This is partially due to the informal description of the EAI foundations in the form of integration patterns from 2004, which makes it difficult to build *trustworthy* integration solutions or to *responsibly* improve these solutions for a more efficient processing. Integration logic can only be trusted if it is possible in principle to prove that it behaves correctly. In turn, this means that users must be enabled by the means to express and specify what their integration solution should do. We call a methodology that takes this into account a *responsible* development of integration solutions.

The main goals of this thesis were to develop trustworthy and more efficient EAI solutions. The most important results of this work can be summarized as follows:

Revisiting the EAI Pattern Foundations For trustworthy integration solutions, an actual and comprehensive EAI foundation in the form of integration patterns is required. We approach

this by revisiting the actuality and comprehensiveness of the *original* patterns from 2004 in the context of the discussed trends and their requirements (e.g., exception handling, security). Thereby, the challenge is to consider the vast amount of existing literature as well as the numerous system implementations to analyze the relevance of existing solutions and identify new practices as patterns in a rigorous way. In particular, we follow a design science research approach by combining a deductive approach for the literature with an inductive approach from the current implementations as part of a systematic pattern engineering process. The rigor of this study allows for a twofold result: (i) identifying trends and research gaps as well as (ii) analyzing integration patterns and identifying new patterns. The trends and research gaps locate the thesis in the integration domain and determines the new challenges and requirements in that domain. These challenges influence the current integration solutions and lead to recurring practices, for which we found common solutions, and thus adapted old or developed new patterns.

In summary, we address research question *RQ1* “*To which extent are the current conceptual foundations of application integration in the form of the EIPs still sufficient for the new challenges and how can they be updated?*”. The analyses show that many of the original patterns are still relevant and practically used (i.e., EIPs denote current conceptual foundations), however, the trends (e.g., stateful conversations, exception handling) led to new practices (as recently acknowledged by the pattern authors [ZPHW16]), which we again collect in the form of patterns. This *extended* pattern catalog denotes an actual and comprehensive state of EAI, which is sufficient in the context of the new challenges. We envision future updates again in the form of extensions of the pattern catalog.

Allowing for Trustworthy EAI To develop trustworthy EAI it is necessary to ground integration solutions on comprehensive foundations that can be formally analyzed. A formal foundation is required for deciding whether integration solutions are functionally correct and whether improvements preserve correctness.

Starting from the pattern catalog, we categorize common characteristics and requirements (e.g., data, time, transacted resources) from the single patterns and select a formalism based on Petri nets, namely db-nets, already covering many of the requirements. Petri nets are a natural choice for the formal treatment of concurrent, distributed systems (e.g., firing is non-deterministic, multiple tokens can be present anywhere in the net). Moreover, during the gap analysis we identify the so far only attempt to formalize integration patterns using Colored Petri nets (CPNs), whose solution concepts could be leveraged for some of the original patterns. From that we develop a temporal extension of db-nets that itself is based on CPN, which we call *timed db-nets*. The challenge is to extend db-nets carefully with a sufficiently expressive, but tractable notion of time. We construct an extension inspired by Timed-Arc Petri Nets. The goal is to inherit the formal analysis results of db-nets (e.g., liveness) and re-assure the desirable property of *reachability* that is sufficient for assessing the functional correctness of pattern realizations. We show that all but one pattern out of the catalog can be formally defined as timed db-nets. The one missing pattern requires dynamic structural changes of the net, which might require more elaborate PNs, if at all possible. The realizations cover pattern candidates of the different categories, which we prototypically evaluate with respect to their correctness in an extension of a state-of-the-art PN tool that allows for simulation. Note that our prototype closes the conceptual vs. implementation gap by simulation. For real-world integration scenarios, we ensured the correctness of composed patterns by careful manual construction.

This leads over to the second aspect of trustworthy integration solutions, which requires the correctness of pattern compositions (i.e., denoting integration scenarios). The challenge is to find a suitable formal representation for compositions that is more comprehensible than timed db-nets, however, provides built-in structural and semantic correctness. We select a graph-structure similar to PNs and define *Integration Pattern Type Graphs* (IPTGs), in which patterns represent nodes

and edges denote data exchange between them. An IPTG subsumes the characteristics of the underlying pattern (representing the internal data flow), however does not allow for expressing the correctness of compositions. Hence, we define *Integration Pattern Contract Graphs* (IPCGs) by adding input and output contracts (i.e., representing the external data flow) that have to be fulfilled in a structurally correct composition. The semantic correctness can only be decided on the timed db-net level, but plain timed db-nets do not have the notion of *contracts*. Hence, we first extend timed db-nets by *boundaries* (e.g., as in *Open Nets* [BM18]) that take pattern characteristics and the external data flow into account, and secondly we translate from IPCGs to *timed db-nets with boundaries*. Consequently, IPCGs are not only more comprehensible than timed db-nets (with boundaries) and more tractable (e.g., for deciding on improvements of compositions), but give the same semantic correctness guarantees. In the evaluation we again prototypically show that the previous real-world pattern compositions are correct with respect to their composition and expected functionality.

In summary, we address *RQ2* “How to formulate integration requirements and scenarios in a usable, expressive and executable integration language?”. We formally defined integration scenarios as IPCGs that together with the construction of timed db-nets with boundaries enable formal specification and analysis of pattern compositions and their underlying execution semantics of single patterns. The IPCGs are more usable compared to timed db-nets due to their higher level of abstraction, but are still executable through a translation to timed db-nets. The resulting formally defined EAI foundations are expressive enough to represent all studied real-world integration scenarios from SAP CPI. With IPCGs and timed db-nets, integration scenarios can be responsibly developed by using their formal semantics to prove that they satisfy their specifications, which denotes a step towards trustworthy EAI solutions. However, the presented solution is still quite technical, which makes a more usable and visually appealing integration language defined on top of IPCGs desirable.

Making Improvements Preserve the Correctness Although IPCGs allow for a responsible development of integration scenarios, this is not immediately guaranteed for the application of improvements (targeting objectives like higher message throughput, lower processing latency or reduced model complexity). The improvements discussed in this work denote optimizations from related domains that we transferred and adapted for EAI and categorized according to their impact: process simplification (cf. comprehensibility), data reduction (cf. volume), and parallelization (cf. velocity). It is important that also the optimizations themselves can be proven correct, in the sense that they do not change the functional behavior of the integration scenarios (essentially the timed db-nets with boundaries corresponding to the compositions before and after applying the improvement are bisimilar). This is challenging, since each improvement has to be formally specified, preferably in a formalism close to IPCGs. For that, we select a graph rewriting framework to formalize each improvement by a rewrite rule and show that the rewriting results in functionally equivalent integration scenarios. We evaluate the impact of these improvements on real-world integration scenarios, showing actual improvements (e.g., immense data volume and modeling complexity reduction potentials).

In summary, we developed a formalism to specify *correctness-preserving* change operations (i.e., improvements) on IPCGs. This extends the notion of *trust* in pattern compositions based on IPCGs to specific optimizations, and thus contributes to *RQ2* in terms of expressive formal foundations that can be used for optimizations. We envision further optimizations to be again formally represented as change operations on IPCGs.

Making Pattern Solutions Justifiable The improvements on the integration scenario level showed promising results for improving the runtime. Some of the improvements like *data reduction* even require actual runtime data to decide on the rewriting. However, it remains unclear how improvements can be measured and compared for real-world systems. One way to do that is

by constructing a benchmark, a pattern benchmark in our case. Such a benchmark can serve as a runtime data provider as required for graph rewritings like data reduction. Beyond the improvements on the scenario level, some of the trends offer opportunities to improve the *efficiency* of integration solutions on the system level. For both levels the challenge is to define a common benchmark that is relevant, scalable, portable and simple to understand. In this work we define *EIPBench* as the first pattern benchmark that addresses relevant integration aspects like message routing, transformation, and delivery qualities. We argue that the focus on patterns, abstracting from the complexity of the underlying domain, makes the benchmark comprehensible. The benchmark defines micro- and macroscale based on the pattern characteristics (e.g., route branching, complex conditions) and cross pattern aspects (e.g., concurrent users, increasing data volumes), respectively. Although the benchmark is portable and can be extended to other technologies or systems, we evaluate EIPBench prototypically for Apache Camel, a contemporary integration system that implements the original integration patterns, and compare the results to the promising *vectorization* or *micro-batching* approach (cf. [Rit15b]). One interesting finding is that vectorized processing requires semantic changes for some patterns (e.g., Content-based Router) to leverage its full potential, which is picked up later in this thesis. For the existing implementations, the results show the strengths (e.g., good baseline processing) and weaknesses (e.g., decreasing throughput for increased data volume, route branching, threading) of contemporary systems and areas, in which vectorization is promising (e.g., scenarios that require an overall higher message throughput).

In summary, we lay the basis for answering *RQ3* “Which related concepts and technology trends can be used to improve integration processing and how can the resulting integration solutions be practically realized and compared?” by comparing the efficiency of integration solutions. With EIPBench, realizations of integration patterns and compositions can be measured and compared to each other. The benchmark helps to justify one improvement or related concept over another.

Improving Pattern Solutions Besides the responsible development of EAI solutions the correctness-preserving optimizations already target an improved processing on the scenario level. On the system level, the *vectorization* benchmarks show promising results, which suggests an exploration of new trends (e.g., software, hardware) for system level improvements. Since the prototypical timed db-net implementation does not provide an artifact that would perform well in benchmarks, more *efficient* pattern realizations are required. However, we recall that challenges in EAI are diverse (i.e., volume, velocity, variety), and thus require different system considerations.

First, for high *volume* data processing big data management systems (BDMS) from the related database domain is a natural choice. In these systems, vectorization is the predominant processing style, which lets us revisit potential semantic integration pattern changes. For a prototypical evaluation, some integration patterns are conceptually specified similar to timed db-nets, however, completely within the contemporary BDMS (e.g., SAP HANA database). That means that the design leverages the active database concepts within the BDMS. Although an equivalence of timed db-net and our BDMS solution is not formally shown, the experimental results suggest that the pattern implementations are correct. The evaluation is conducted using EIPBench and shows that despite the fully transactional processing within the BDMS the high data volume routing and transformation message throughput can be significantly improved.

Second, for high *velocity* data processing, we propose a combination of moving the processing closer to the network and part with software implementations in favor of a *specialization* on modern, reconfigurable hardware. Network attached data processing is naturally more efficient for EAI, since it gets the integration logic closer to the communication wire. Moreover, reconfigurable hardware is a technology trend that promises unmatched efficiency with respect to data processing and energy consumption. Hardware dataflow architectures do not only have the advantage that they can be put close to the network (i.e., they have no additional operating system interactions

and are not control flow centric, compared to software), but they also allow for highly parallel data pipelining. For a prototypical evaluation, some integration patterns are conceptually defined directly in FPGA hardware, which denotes the most advanced reprogrammable hardware currently available. The evaluation of this integration system on a chip benchmarked in EIPBench not only shows so far unmatched message throughput for routing and transformation patterns, but also that it is not vulnerable to the identified weaknesses of the software solutions. For example, an increasing number of route branchings or complex conditions have no or only minimal impact on the performance compared to the software systems. However, the major downside to reconfigurable hardware that we discussed in this thesis is the trade-off: velocity vs. resource limits. For example, the resources on the chip like on-chip memory are limited, while higher capacity external resource access slows down the velocity. With respect to the discussed scenario level optimization, the application of the optimization strategies show slight performance improvements, however, less significant than those gained through the specialization. Moreover, given a hardware integration scenario, the performance is predictable and guaranteed to remain stable.

Finally, the third system that we built deals with the *variety* aspect for multimedia data integration. We argue that a multimedia application can be seen as an integration scenario. Instead of textual message formats, messages are assumed to contain multimedia data (e.g., image, video stream) that are directly considered for routing and transformation (e.g., allowing for query by sketch style routing conditions). Although we extend EIPBench for multimedia data and benchmark our solution, this system was not defined with efficient message processing in mind. Instead it gives a conceptual design for dealing with multimedia data in an EAI system from the integration scenarios down to the runtime. The actual information in the multimedia data is addressed by semantic web concepts (e.g., ontologies as schema, SPARQL routing conditions). The resulting approach could be formally represented by timed db-nets through an extension for the semantic data processing.

In summary, we addressed the pattern solution part of *RQ3* “Which related concepts and technology trends can be used to improve integration processing and how can the resulting integration solutions be practically realized and compared?”, and thus targeted crucial problems of current EAI systems (volume, velocity, variety) by proposing three novel pattern solutions. The proposed system implementations improve the integration processing for current scenarios with respect to performance (volume and velocity) and emerging multimedia data (variety of message formats). Each of the systems leverages emerging technological trends, which could influence the design of future EAI systems. For practical considerations combining the three systems into one could be preferable.

Overall Contribution Altogether, this thesis contributes an updated, comprehensive pattern catalog as basis for formal EAI foundations. The formal foundation in the form of composed pattern and their improvements denotes a necessary step towards trustworthy EAI. The novel pattern solutions denote efficient implementations and pave the way for future EAI systems and system architectures.

7.2 Outlook

While the formal results and novel system designs are already being incorporated into commercial EAI solutions (e.g., SAP Cloud Platform Integration), we see various ways, in which the results of this thesis can be extended. The four major extensions are set into the context of the conceptual system overview from [Chapter 1](#) in [Figure 7.1](#). Besides the continuous task of pattern identification and adaptation (not shown), for which we have given an extended pattern identification approach, the four areas of extension are enclosed by dashed lines (i.e., modeling, leveraging the formal

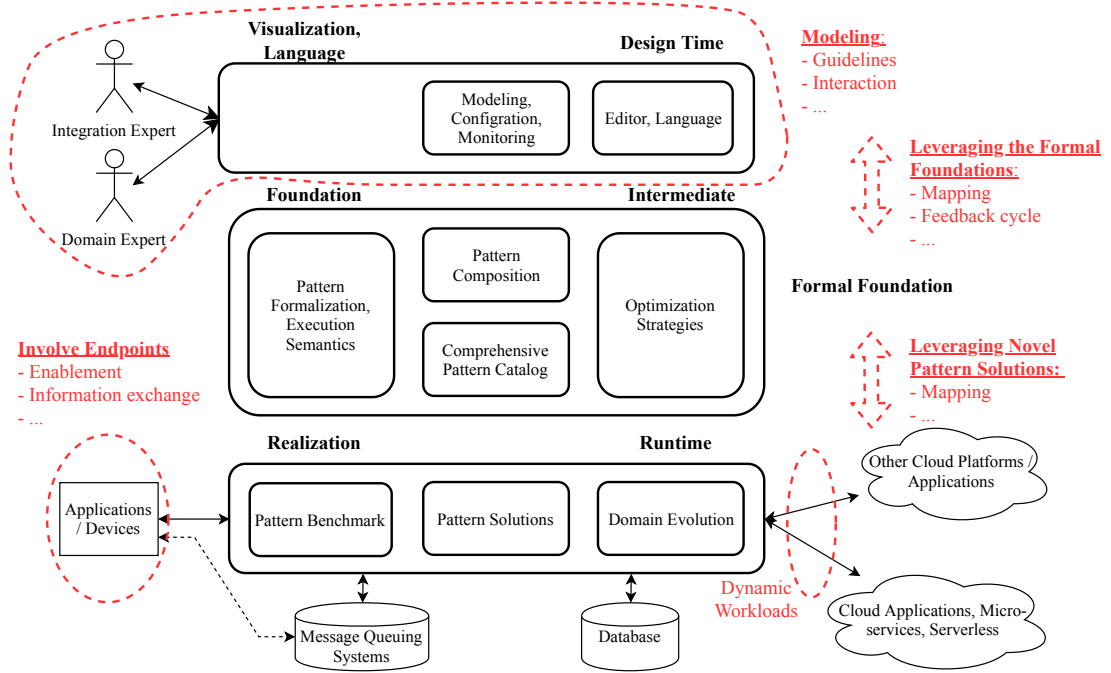


Figure 7.1: Conceptual EAI system overview — the big picture

foundations and novel pattern solutions, and involving endpoints) and discussed subsequently.

Modeling The modeling research gap identified in [Chapters 1](#) and [2](#) was not addressed in this thesis and still remains unsolved. While IPCGs and timed db-nets denote suitable formal EAI foundations, a more comprehensible and visually appealing integration language is required to further empower the user to efficiently develop and interact with integration solutions. We propose to base this language on IPCGs to enjoy built-in formal analysis results and optimization strategies. The semantic analysis should be supported by the development of a timed db-net model checker for an analysis beyond validation through simulation. Although we used BPMN as visual representation of integration scenarios in this thesis, BPMN can only be used for modeling in this context, if extended for or adapted to EAI execution semantics. Alternatively, a new notation is required.

An interactive development of integration solutions in such an integration language will further require the development of new editors. These editors will not only need to support the efficient interaction with different information layers (e.g., for formal analysis and optimization; cf. [Chapter 3](#)) and new formats together with system supported configuration (e.g., for query by sketch conditions on multimedia data; cf. [Section 6.3](#)), but also enable smarter ways of scenario development like templating (cf. [Chapter 2](#)), composition modeling guidelines (cf. [Chapter 3](#)), proposal of suitable subsequent patterns during modeling (e.g., [\[MLZN09, MN10\]](#)), and immediate runtime feedback (e.g., based on simulation). We envision the usage of IPCGs and timed db-nets for the conceptual foundations of such editors. While IPCGs provide formally defined optimizations and timed db-nets help with the formal analysis on the pattern level, together they could allow for feedback and analysis capabilities in the editors, covering all layers from the scenario or process to the data level.

When considering the shift in usage from computers to mobile devices, special editors for

the integration modeling on smaller and resource constraint devices could be (cf. [RH18]). This would enable (non-technical) users to integrate context services, mobile and business applications more flexibly, beyond simple task automation. With IPCGs as foundations for mobile integration scenarios, the structural correctness can be checked and optimization strategies can be applied without the need of the computationally more expensive verification of timed db-nets.

Leveraging the Formal Foundations The definition of IPCGs is independent of modeling languages, however, to enjoy their benefits (e.g., structural and semantic correctness, formal analysis), existing or new modeling languages have to be grounded on or integrated with IPCGs. Such an integration can be facilitated by mappings from user facing models to IPCGs and vice versa, which allows for the study of their (cross-layer) interactions (e.g., soundness checks, feedback from model checking or simulation, static optimizations). To leverage the full potential of the formal foundations developed in this thesis and make the results accessible in practice, an automatic or tool assisted translation from IPCGs to timed db-nets is required (cf. model vs. system gap in Chapter 3). We assume that new patterns might be formally developed using such a tool as an interplay between IPCGs and timed db-nets, which should be connected to the modeling language.

Moreover, with the translation of IPCGs to timed db-nets, more sophisticated cost models can be developed (e.g., taking the inherent complexities of the patterns into account). With that a more systematic analysis of optimizing dynamic workloads (e.g., cf. [Böh11]) can be analyzed and studied in the context of timed db-nets (e.g., leading to more precise cost semantics) and IPCGs (for model complexity). In addition, the propagation of the optimizations to all levels (e.g., visual and formal models) could be studied considering varying requirements of users (e.g., integration or domain experts, analysts).

Currently, the optimization rules are stratified according to their effects for the application (i.e., “simplification before parallelization” and “structure before data”). More elaborate rule application and execution schemes could be investigated to give better optimality guarantees.

Leveraging Novel Pattern Solutions The new pattern solutions (i.e., BDMS and FPGA pattern implementations) are functionally correct, however, not yet formally connected to timed db-nets. A mapping from timed db-nets to the pattern solutions would allow for an automatic translation, and thus for a practical use and a more systematic study of different realization in different settings and platforms with varying demands and opportunities. This would also provide an end-to-end perspective from the modeling language to the pattern solutions, and thus make implementation-dependent studies of the complete system more tractable (i.e., without manual steps). In the other direction, a mapping from pattern solutions to timed db-nets would allow for soundness check of existing system implementations (e.g., Apache Camel) as well as formal analysis during their evolution.

With the advent of general data processing systems (e.g., Apache Spark, Apache Flink), the EAI domain should be further analyzed with respect to conceptual and implementation overlaps with related domains. For example, our studies in Chapter 6 experimentally show that some patterns can be implemented on BDMS and stream processing hardware. This is similar to the experimental studies conducted for overlaps with workflow management [SRL10] and complex event processing systems [ES13]. We argue that a more systematic and formal attempt to link between the related domains would benefit research, but also practical system implementations and their users.

Involving Endpoints As also observed by [Böh11] the current optimization approaches focus on the (data) integration system itself. Having seen the emergence of patterns like *Commutative Receiver* or *Timed Redelivery until Acknowledge* (both identified in Chapter 2), some of the current integration logic within EAI systems should be better moved to the endpoints, where they

belong. Moreover, the endpoints have relevant information that would improve the optimizations in this thesis and lead to the development of new optimizations (e.g., required data elements for the *early-filter* optimization and acceptable load as capacity for the configuration of throttling patterns in the *reduce requests* optimization).

We argue that the involvement of endpoints should be further studied, for which IPCGs provide a formal foundation (cf. example optimizations in [Appendix A](#) and [\[RFRM19\]](#)). While this might be challenging for packaged, legacy applications, this would give each participant in the communication more responsibility and information, which will not only make the message exchange more efficient (e.g., only required data is exchanged, dynamic workloads can be detected earlier and reacted to in a collaborative way), but also more robust (e.g., avoiding overloaded or reacting to unavailable endpoints).

Altogether, the presented EAI foundations, formalizations and solutions provide relevant contributions and solutions for the further work beyond this thesis.

Bibliography

- [AAA⁺14] Sattam Alsubaiee, Yasser Altowim, Hotham Altwaijry, Alexander Behm, Vinayak R. Borkar, Yingyi Bu, Michael J. Carey, Inci Cetindil, Madhusudan Cheelangi, Khurram Faraaz, Eugenia Gabrielova, Raman Grover, Zachary Heilbron, Young-Seok Kim, Chen Li, Guangqiang Li, Ji Mahn Ok, Nicola Onose, Pouria Pirzadeh, Vassilis J. Tsotras, Rares Vernica, Jian Wen, and Till Westmann. AsterixDB: A scalable, open source BDMS. *International Conference on Very Large Data Bases (VLDB)*, 7(14):1905–1916, 2014. (cit. on pp. 203, 208, and 209)
- [AAB⁺17] Serge Abiteboul, Marcelo Arenas, Pablo Barceló, Meghyn Bienvenu, Diego Calvanese, Claire David, Richard Hull, Eyke Hüllermeier, Benny Kimelfeld, Leonid Libkin, Wim Martens, Tova Milo, Filip Murlak, Frank Neven, Magdalena Ortiz, Thomas Schwentick, Julia Stoyanovich, Jianwen Su, Dan Suciu, Victor Vianu, and Ke Yi. Research directions for principles of data management (abridged). *International Conference on Management of Data (SIGMOD)*, 45(4):5–17, 2017. (cit. on pp. 1, 2, 137, 246, 247, 268, and 269)
- [ABMR10] Kunal Agrawal, Anne Benoit, Loic Magnan, and Yves Robert. Scheduling algorithms for linear workflow optimization. In *International Parallel & Distributed Processing Symposium (IPDPS)*, pages 1–12. IEEE, 2010. (cit. on p. 139)
- [ACÇ⁺03] Daniel J Abadi, Don Carney, Ugur Çetintemel, Mitch Cherniack, Christian Convey, Sangdon Lee, Michael Stonebraker, Nesime Tatbul, and Stan Zdonik. Aurora: a new model and architecture for data stream management. *International Journal on Very Large Data Bases*, 12(2):120–139, 2003. (cit. on p. 4)
- [ACG⁺04] Arvind Arasu, Mitch Cherniack, Eduardo F. Galvez, David Maier, Anurag Maskey, Esther Ryvkina, Michael Stonebraker, and Richard Tibbetts. Linear road: A stream data management benchmark. In *International Conference on Very Large Data Bases (VLDB)*, 2004. (cit. on p. 169)
- [ACKM04] Gustavo Alonso, Fabio Casati, Harumi Kuno, and Vijay Machiraju. *Web services*. Springer, 2004. (cit. on p. 3)
- [ACL96] Paulo SC Alencar, Donald D Cowan, and Carlos José Pereira de Lucena. A formal approach to architectural design patterns. In *International Symposium of Formal Methods Europe (FME)*, pages 576–594. Springer, 1996. (cit. on p. 130)
- [Act96] Act-Net Consortium. The active database management system manifesto: A rulebase of ADBMS features. *International Conference on Management of Data (SIGMOD)*, 25(3):40–49, 1996. (cit. on p. 216)

- [ADGI08] Jagrati Agrawal, Yanlei Diao, Daniel Gyllstrom, and Neil Immerman. Efficient pattern matching over event streams. In *International Conference on Management of Data (SIGMOD)*, pages 147–160. ACM, 2008. (cit. on p. 271)
- [Adr13] AdroitLogic. ESB performance. <http://esbperformance.org/>, 2013. (cit. on pp. 184, 185, 196, and 197)
- [AFG⁺05] Cristinel Ababei, Yan Feng, Brent Goplen, Hushrav Mogal, Tianpei Zhang, Kia Bazargan, and Sachin Sapatnekar. Placement and routing in 3D integrated circuits. *Design & Test of Computers*, 22(6):520–531, 2005. (cit. on p. 226)
- [AFP16] Steve Asmus, Ahmed Fattah, and Chris Pavlovski. Enterprise cloud deployment: Integration patterns and assessment model. *Cloud Computing*, 3(1):32–41, 2016. (cit. on pp. 31, 34, and 35)
- [AGH16] S Akshay, Blaise Genest, and Loïc Hélouët. Decidable classes of unbounded Petri nets with time and urgency. In *International Conference on Applications and Theory of Petri Nets and Concurrency (PN)*, pages 301–322. Springer, 2016. (cit. on p. 78)
- [AGWC14] Robert Amelard, Jeffrey Glaister, Alexander Wong, and David A Clausi. Melanoma decision support using lighting-corrected intuitive feature models. In *Computer Vision Techniques for the Diagnosis of Skin Cancer*, pages 193–219. Springer, 2014. (cit. on pp. 246, 255, and 256)
- [AI83] Arvind and Robert A. Iannucci. A critique of multiprocessing von Neumann style. In *International Symposium on Computer Architecture (ISCA)*, pages 426–436. ACM, 1983. (cit. on p. 204)
- [AIM10] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The Internet of Things: A survey. *Computer networks*, 54(15):2787–2805, 2010. (cit. on p. 1)
- [Ale77] Christopher Alexander. *A pattern language: towns, buildings, construction*. Oxford university press, 1977. (cit. on p. 19)
- [All70] Frances E. Allen. Control flow analysis. *SIGPLAN Notices*, 5(7):1–19, 1970. (cit. on p. 106)
- [All97] Robert J Allen. A formal approach to software architecture. Technical report, Carnegie-Mellon University, School of Computer Science, 1997. (cit. on p. 130)
- [ALR⁺14] Saima Gulzar Ahmad, Chee Sun Liew, M Mustafa Rafique, Ehsan Ullah Munir, and Samee U Khan. Data-intensive workflow optimization based on application task graph partitioning in heterogeneous computing systems. In *International Conference on Big Data and Cloud Computing (BdCloud)*, pages 129–136. IEEE, 2014. (cit. on p. 139)
- [ALSU06] Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools (2nd Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2006. (cit. on p. 161)
- [Ama07] Xavier Amatriain. A domain-specific metamodel for multimedia processing systems. *Transactions on Multimedia*, 9(6):1284–1298, 2007. (cit. on p. 273)

- [And81] Norman H. Anderson. *Foundations of Information Integration Theory*. Number 1 in Foundations of Information Integration Theory. Academic Press, 1981. (cit. on pp. 3 and 4)
- [Apa17a] Apache Foundation. Apache Flume. <https://flume.apache.org/>, 2017. (cit. on pp. 38, 39, 47, and 48)
- [Apa17b] Apache Foundation. Apache Nifi. <https://nifi.apache.org/>, 2017. (cit. on pp. 38, 39, 46, 47, and 48)
- [ASPT15] Marco Autili, Amleto Di Salle, Alexander Perucci, and Massimo Tivoli. On the automated synthesis of enterprise integration patterns to adapt choreography-based distributed systems. In *International Workshop on Foundations of Coordination Languages and Self-Adaptive Systems (FOCLASA)*, pages 33–47, 2015. (cit. on pp. 31, 32, 37, 44, and 49)
- [AT08] David Aumüller and Andreas Thor. Mashup-Werkzeuge zur ad-hoc-Datenintegration im Web. *Datenbank-Spektrum*, 8(26):4–10, 2008. (cit. on p. 3)
- [AVH04] Grigoris Antoniou and Frank Van Harmelen. *A semantic web primer*. MIT press, 2004. (cit. on p. 250)
- [Bac78] John W. Backus. *Can programming be liberated from the von Neumann style?: a functional style and its algebra of programs*. Research report, IBM Research Division: Computer science. IBM Corporation, 1978. (cit. on p. 204)
- [Bal01] Gianfranco Balbo. Introduction to stochastic Petri nets. In *School organized by the European Educational Forum*, volume 2090, pages 84–155. Springer, 2001. (cit. on p. 79)
- [BB14] Samik Basu and Tevfik Bultan. Automatic verification of interactions in asynchronous systems with unbounded buffers. In *International conference on Automated software engineering (ASE)*, pages 743–754, 2014. (cit. on p. 36)
- [BBC⁺12] Vinayak R. Borkar, Yingyi Bu, Michael J. Carey, Joshua Rosen, Neoklis Polyzotis, Tyson Condie, Markus Weimer, and Raghu Ramakrishnan. Declarative systems for large-scale machine learning. *IEEE Data Eng. Bull.*, 35(2):24–32, 2012. (cit. on pp. 204 and 209)
- [BBC⁺18] Paolo Baldan, Roberto Bruni, Andrea Corradini, Fabio Gadducci, Hernán C. Melgratti, and Ugo Montanari. Event structures for Petri nets with persistence. *Logical Methods in Computer Science*, 14(3), 2018. (cit. on p. 130)
- [BBG11] R. Buyya, J. Broberg, and A.M. Goscinski. *Cloud Computing: Principles and Paradigms*. Wiley Series on Parallel and Distributed Computing. Wiley, 2011. (cit. on p. 1)
- [BBGM15] Paolo Baldan, Filippo Bonchi, Fabio Gadducci, and Giacomina Valentina Monreale. Modular encoding of synchronous and asynchronous interactions using open Petri nets. *Science of Computer Programming*, 109:96–124, 2015. (cit. on pp. 104, 110, 131, and 132)

- [BBM12] Hema Banati, Punam Bedi, and Preeti Marwaha. Extending BPEL for WSDL-temporal based web services. In *International Conference on Hybrid Intelligent Systems (HIS)*, pages 484–489. IEEE, 2012. (cit. on p. 132)
- [BBN⁺12] Chaitan Baru, Milind Bhandarkar, Raghunath Nambiar, Meikel Poess, and Tilmann Rabl. Big data benchmarking. In *Workshop on Management of Big Data Systems (MBDS)*, pages 39–40, 2012. (cit. on pp. 169 and 196)
- [BC11] Shekhar Borkar and Andrew A Chien. The future of microprocessors. *Communications of the ACM*, 54(5):67–77, 2011. (cit. on p. 221)
- [BCDM08] Daniele Braga, Stefano Ceri, Florian Daniel, and Davide Martinenghi. Mashing up search services. *IEEE Internet Computing*, 12(5):16–23, 2008. (cit. on p. 33)
- [BCN⁺12] Anne Benoit, Mathias Coqblin, Jean-Marc Nicod, Laurent Philippe, and Veronika Rehn-Sonigo. Throughput optimization for pipeline workflow scheduling with setup times. In *Euro-Par Workshops*, pages 57–67, 2012. (cit. on p. 139)
- [BCPV04] Antonio Brogi, Carlos Canal, Ernesto Pimentel, and Antonio Vallecillo. Formalizing web service choreographies. *Electronic notes in theoretical computer science*, 105:73–94, 2004. (cit. on p. 130)
- [BCSS99] Guruduth Banavar, Tushar Chandra, Robert Strom, and Daniel Sturman. A case for message oriented middleware. In *International Symposium on Distributed Computing*, pages 1–17. Springer, 1999. (cit. on pp. 3 and 23)
- [BD91] Bernard Berthomieu and Michel Diaz. Modeling and verification of time dependent systems using time Petri nets. *IEEE Trans. Soft. Eng.*, 17(3), 1991. (cit. on p. 78)
- [BDTH05] Alistair Barros, Marlon Dumas, and Arthur HM Ter Hofstede. Service interaction patterns. In *International Conference on Business Process Management (BPM)*, pages 302–318, 2005. (cit. on pp. 19, 31, 34, 38, 44, and 49)
- [BEH⁺10] Dominic Battré, Stephan Ewen, Fabian Hueske, Odej Kao, Volker Markl, and Daniel Warneke. Nephele/PACTs: a programming model and execution framework for web-scale analytical processing. In *Symposium on Cloud Computing (SoCC)*, pages 119–130, 2010. (cit. on p. 107)
- [BGG⁺05] Nadia Busi, Roberto Gorrieri, Claudio Guidi, Roberto Lucchi, and Gianluigi Zavattaro. Choreography and orchestration: A synergic approach for system design. In *International Conference on Service-Oriented Computing (ICSOC)*, pages 228–240. Springer, 2005. (cit. on p. 130)
- [BGP⁺10] Helge Buckow, Hans-Jürgen Groß, Gunther Piller, Karl Prott, Johannes Willkomm, and Alfred Zimmermann. Integration strategies and patterns for SOA and standard platforms. In *GI Jahrestagung (1)*, pages 398–403, 2010. (cit. on p. 35)
- [BH15] Manisha Bhange and HA Hingoliwala. Smart farming: Pomegranate disease detection using image processing. *Procedia Computer Science*, 58:280–288, 2015. (cit. on pp. 1, 246, 255, and 256)

- [BHCDG⁺13] Babak Bagheri Hariri, Diego Calvanese, Giuseppe De Giacomo, Alin Deutsch, and Marco Montali. Verification of relational data-centric dynamic systems with external services. In *International Conference on Management of Data (SIGMOD)*, pages 163–174. ACM, 2013. (cit. on pp. 85 and 276)
- [BHLW08a] Matthias Böhm, Dirk Habich, Wolfgang Lehner, and Uwe Wloka. DIPBench: An independent benchmark for data-intensive integration processes. In *International Conference on Data Engineering (ICDE) Workshops*, pages 214–221, 2008. (cit. on pp. 197 and 198)
- [BHLW08b] Matthias Böhm, Dirk Habich, Wolfgang Lehner, and Uwe Wloka. DIPBench toolsuite: A framework for benchmarking integration systems. In *International Conference on Data Engineering (ICDE)*, pages 1596–1599, 2008. (cit. on pp. 197 and 198)
- [BHLW09] Matthias Böhm, Dirk Habich, Wolfgang Lehner, and Uwe Wloka. Systemübergreifende Kostennormalisierung für Integrationsprozesse. In *Datenbanksysteme für Business, Technologie und Web (BTW)*, pages 67–86, 2009. (cit. on p. 153)
- [BHM15] Eric Badouel, Loïc Hélouët, and Christophe Morvan. Petri nets with structured data. In *International Conference on Application and Theory of Petri Nets (PN)*, pages 212–233. Springer, 2015. (cit. on p. 67)
- [BHP⁺11] Matthias Böhm, Dirk Habich, Steffen Preissler, Wolfgang Lehner, and Uwe Wloka. Cost-based vectorization of instance-based integration processes. *Information Systems*, 36(1):3–29, 2011. (cit. on pp. 9, 139, 140, 160, 161, and 169)
- [BJ87] K. Birman and T. Joseph. Exploiting virtual synchrony in distributed systems. *SIGOPS - Operating Systems Review*, 21(5):123–138, 1987. (cit. on p. 3)
- [BK09] Alexander Böhm and Carl-Christian Kanne. Processes are data: A programming model for distributed applications. In *International Conference on Web Information Systems Engineering (WISE)*, pages 43–56, Berlin, Heidelberg, 2009. Springer-Verlag. (cit. on p. 131)
- [BK11] Alexander Böhm and Carl-Christian Kanne. Demaq/transscale: Automated distribution and scalability for declarative applications. *Information Systems*, 36(3):565–578, 2011. (cit. on pp. 136, 139, 211, and 269)
- [BLFM05] Tim Berners-Lee, Roy Fielding, and Larry Masinter. Uniform resource identifier (URI): Generic syntax. Technical report, RFC Editor, 2005. (cit. on pp. 250 and 251)
- [BLN86] Carlo Batini, Maurizio Lenzerini, and Shamkant B. Navathe. A comparative analysis of methodologies for database schema integration. *ACM Comput. Surv.*, 18(4):323–364, 1986. (cit. on pp. 206 and 269)
- [BM10] Mark Birbeck and Shane McCarron. Curie syntax 1.0: A syntax for expressing compact URIs. W3C working group note, december 2010. *World Wide Web Consortium*, <https://www.w3.org/TR/2010/NOTE-curie-20101216>, 2010. (cit. on p. 251)

- [BM11] Luiz Fernando Bittencourt and Edmundo Roberto Mauro Madeira. HCOC: a cost optimization algorithm for workflow scheduling in hybrid clouds. *Journal of Internet Services and Applications*, 2(3):207–227, 2011. (cit. on p. 139)
- [BM18] John C. Baez and Jade Master. Open Petri nets. *CoRR*, abs/1808.05415, 2018. (cit. on pp. 104, 110, 131, 132, and 283)
- [BMR⁺96] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. *Pattern-Oriented Software Architecture - Volume 1: A System of Patterns*. Wiley Publishing, 1996. (cit. on pp. 19 and 21)
- [BN09] Philip A Bernstein and Eric Newcomer. *Principles of transaction processing*. Morgan Kaufmann, 2009. (cit. on pp. 207, 208, 213, and 214)
- [Böh10] Alexander Böhm. *Building Scalable, Distributed Applications with Declarative Messaging*. PhD thesis, University of Mannheim, 2010. (cit. on pp. 136 and 139)
- [Böh11] Matthias Böhm. *Cost-based optimization of integration flows*. PhD thesis, Dresden University of Technology, 2011. (cit. on pp. 3, 205, and 287)
- [Bol90] Tommaso Bolognesi. From timed Petri nets to timed LOTOS. In *International Workshop on Protocol Specification, Testing and Verification, IFIP WG6. 1*, pages 377–406, 1990. (cit. on pp. 68 and 73)
- [BPV06] Bernard Berthomieu, Florent Peres, and François Vernadat. Bridging the gap between timed automata and bounded time Petri nets. In *International Conference on Formal Modeling and Analysis of Timed Systems*, pages 82–97. Springer, 2006. (cit. on p. 75)
- [BRC10] Rajkumar Buyya, Rajiv Ranjan, and Rodrigo N Calheiros. Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services. In *International Conference on Algorithms and Architectures for Parallel Processing*, pages 13–31. Springer, 2010. (cit. on p. 5)
- [BRFR12] Carsten Binnig, Robin Rehrmann, Franz Faerber, and Rudolf Riewe. FunSQL: It is time to make SQL functional. In *International Conference on Extending Database Technology / International Conference on Database Theory (EDBT/ICDT) Workshops*, pages 41–46. ACM, 2012. (cit. on pp. 204 and 209)
- [Bri04] Dan Brickley. RDF vocabulary description language 1.0: RDF schema. *World Wide Web Consortium*, <http://www.w3.org/TR/rdf-schema/>, 2004. (cit. on p. 251)
- [BTK⁺17] Sukanya Bhowmik, Muhammad Adnan Tariq, Boris Koldehofe, Frank Dürr, Thomas Kohler, and Kurt Rothermel. High performance publish/subscribe middleware in software-defined networks. *IEEE/ACM Trans. Netw.*, 25(3):1501–1516, 2017. (cit. on p. 278)
- [BV06] András Balogh and Dániel Varró. Pattern composition in graph transformation rules. In *European Workshop on Composition of Model Transformations (CMT)*, pages 33–37, 2006. (cit. on p. 141)

- [BWHL08] Matthias Böhm, Uwe Wloka, Dirk Habich, and Wolfgang Lehner. Model-driven generation and optimization of complex integration processes. In *International Conference on Enterprise Information Systems (ICEIS) (1)*, pages 131–136, 2008. (cit. on pp. 136 and 139)
- [BZ10] Ian Bayley and Hong Zhu. A formal language of pattern composition. In *International conference on pervasive patterns (PATTERNS 2010), XPS (Xpert Publishing Services), Lisbon, Portugal*, pages 1–6, 2010. (cit. on p. 131)
- [CAO⁺07] Semih Cetin, N Ilker Altintas, Halit Oguztüzün, Ali H Dogru, Ozgur Tufekci, and Selma Suloglu. A mashup-based strategy for migration to service-oriented computing. In *International Conference on Pervasive Service*, pages 169–172, 2007. (cit. on pp. 31 and 33)
- [Cas05] Eylon Caspi. *Design Automation for Streaming Systems*. PhD thesis, UC Berkeley, Berkeley, CA, USA, 2005. (cit. on pp. 227, 229, 240, 271, and 272)
- [Cat11] Rick Cattell. Scalable SQL and NoSQL data stores. *SIGMOD Record*, 39(4):12–27, 2011. (cit. on p. 203)
- [CC77] Patrick Cousot and Radhia Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Symposium on Principles of programming languages (POPL)*, pages 238–252. ACM, 1977. (cit. on p. 161)
- [CC02] Patrick Cousot and Radhia Cousot. Systematic design of program transformation frameworks by abstract interpretation. In *Symposium on Principles of Programming Languages (POPL)*, pages 178–190. ACM, 2002. (cit. on p. 161)
- [CCP⁺16] Adrian M Caulfield, Eric S Chung, Andrew Putnam, Hari Angepat, Jeremy Fowers, Michael Haselman, Stephen Heil, Matt Humphrey, Puneet Kaur, Joo-Young Kim, et al. A cloud-scale acceleration architecture. In *Annual International Symposium on Microarchitecture*, page 7. IEEE Press, 2016. (cit. on pp. 222 and 277)
- [CÇR⁺03] Donald Carney, Ugur Çetintemel, Alex Rasin, Stanley B. Zdonik, Mitch Cherniack, and Michael Stonebraker. Operator scheduling in a data stream manager. In *International Conference on Very Large Data Bases (VLDB)*, pages 838–849, 2003. (cit. on p. 215)
- [CDG05] Andrea Clematis, Daniele D’Agostino, and Antonella Galizia. An object interface for interoperability of image processing parallel library in a distributed environment. In *International conference on image analysis and processing (ICIAP)*, pages 584–591, 2005. (cit. on p. 274)
- [CDGL⁺09] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, Antonella Poggi, Mariano Rodriguez-Muro, and Riccardo Rosati. Ontologies and databases: The DL-lite approach. In *Reasoning Web International Summer School*, pages 255–356. Springer, 2009. (cit. on p. 277)
- [CDN11] Surajit Chaudhuri, Umeshwar Dayal, and Vivek Narasayya. An overview of business intelligence technology. *Communications of the ACM*, 54(8):88–98, 2011. (cit. on p. 222)

- [CDPV07] S. K. Chang, V. Deufemia, G. Polese, and M. Vacca. A normalization framework for multimedia databases. *IEEE Transactions on Knowledge and Data Engineering*, 19(12):1666–1679, Dec 2007. (cit. on pp. 248, 257, 258, and 273)
- [CDV08] David Chen, Guy Doumeingts, and François Vernadat. Architectures for enterprise integration and interoperability: Past, present and future. *Computers in industry*, 59(7):647–659, 2008. (cit. on p. 36)
- [Cha04] David Chappell. *Enterprise Service Bus*. O’Reilly Media, Inc., 2004. (cit. on p. 196)
- [CHK09] Michael J. Cafarella, Alon Halevy, and Nodira Khoussainova. Data integration for the relational web. *International Conference on Very Large Data Bases (VLDB)*, 2(1):1090–1101, 2009. (cit. on p. 270)
- [CJL⁺08] Ronnie Chaiken, Bob Jenkins, Per-Åke Larson, Bill Ramsey, Darren Shakib, Simon Weaver, and Jingren Zhou. Scope: easy and efficient parallel processing of massive data sets. *VLDB Endowment*, 1(2):1265–1276, 2008. (cit. on p. 209)
- [CKE⁺15] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. Apache Flink: Stream and batch processing in a single engine. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 36(4), 2015. (cit. on pp. 170, 173, 204, and 215)
- [Clo17] Cloudpipes. Cloudpipes Documentation. <https://docs.cloudpipes.com/>, 2017. (cit. on pp. 39 and 48)
- [CLRS09] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2009. (cit. on p. 102)
- [CMR⁺97] Andrea Corradini, Ugo Montanari, Francesca Rossi, Hartmut Ehrig, Reiko Heckel, and Michael Löwe. Algebraic approaches to graph transformation—part i: Basic concepts and double pushout approach. In *Handbook Of Graph Grammars And Computing By Graph Transformation: Volume 1: Foundations*, pages 163–245. World Scientific, 1997. (cit. on pp. 141, 142, and 143)
- [Coc70] John Cocke. Global common subexpression elimination. In *Symposium on Compiler Optimization*, pages 20–24. ACM, 1970. (cit. on p. 161)
- [Cod70] Edgar F Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970. (cit. on p. 203)
- [Cor02] IBM Corporation. Business process execution language for web services BPEL4WS (version 1.1), 2002. (cit. on pp. 3 and 132)
- [CR13] Stephen Cranefield and Surangika Ranathunga. Embedding agents in business processes using enterprise integration patterns. In *International Workshop on Engineering Multi-Agent Systems*, pages 97–116, 2013. (cit. on p. 37)
- [CRP14] Glaucia Melissa Medeiros Campos, Nelson Souto Rosa, and Luis Ferreira Pires. A survey of formalization approaches to service composition. In *International Conference on Services Computing (SCC)*, pages 179–186. IEEE, 2014. (cit. on p. 132)

- [CRRCA11] Cristina Cabanillas, Manuel Resinas, Antonio Ruiz-Cortés, and Ahmed Awad. Automatic generation of a data-centered view of business processes. In *International Conference on Advanced Information Systems Engineering (CAiSE)*, pages 352–366. Springer, 2011. (cit. on p. 131)
- [CSZ⁺14] Fei Chen, Yi Shan, Yu Zhang, Yu Wang, Hubertus Franke, Xiaotao Chang, and Kun Wang. Enabling FPGAs in the cloud. In *ACM Conf. on Computing Frontiers*, pages 3:1–3:10, 2014. (cit. on pp. 222 and 236)
- [CT93] Véronique Clément and Monique Thonnat. A knowledge-based approach to integration of image processing procedures. *Graphical Model and Image Processing (CVGIP)*, 57(2):166–184, March 1993. (cit. on p. 273)
- [Cul86] David E Culler. Dataflow architectures. *Annual review of computer science*, 1(1):225–253, 1986. (cit. on pp. 204 and 224)
- [Cur04] Edward Curry. Message-oriented middleware. *Middleware for communications*, pages 1–28, 2004. (cit. on pp. 3 and 23)
- [DAF⁺03] Yanlei Diao, Mehmet Altinel, Michael J. Franklin, Hao Zhang, and Peter Fischer. Path sharing and predicate evaluation for high-performance XML filtering. *ACM Trans. Database Syst.*, 28(4):467–516, December 2003. (cit. on p. 192)
- [DBL17] *33rd IEEE International Conference on Data Engineering, ICDE 2017, San Diego, CA, USA, April 19-22, 2017*. IEEE Computer Society, 2017. (cit. on p. 279)
- [DBL18] *34th IEEE International Conference on Data Engineering Workshops, ICDE Workshops 2018, Paris, France, April 16-20, 2018*. IEEE Computer Society, 2018. (cit. on p. 279)
- [DD99] Ruxandra Domenig and Klaus R Dittrich. An overview and classification of mediated query systems. *International Conference on Management of Data (SIGMOD)*, 28(3):63–72, 1999. (cit. on p. 4)
- [DD09] Lucas Dixon and Ross Duncan. Graphical reasoning in compact closed categories for quantum computation. *Annals of Mathematics and Artificial Intelligence*, 56(1):23, 2009. (cit. on p. 143)
- [DEL17] DELL Boomi. AtomSphere User Guide. <http://help.boomi.com/atomsphere/GUID-A98714FA-9EAB-4B69-BCC8-7D8984F0B0EC.html>, 2017. (cit. on pp. 2, 38, 39, 46, 47, and 48)
- [DG08] Jeffrey Dean and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008. (cit. on pp. 172 and 275)
- [DG10] Remco M. Dijkman and Pieter Van Gorp. BPMN 2.0 execution semantics formalized as graph rewrite rules. In *International Workshop on Business Process Modeling Notation (BPMN)*, pages 16–30, 2010. (cit. on p. 162)
- [DGV⁺14] Umeshwar Dayal, Chetan Gupta, Ravigopal Vennelakanti, Marcos R Vieira, and Song Wang. An approach to benchmarking industrial big data applications. In *Workshop on Big Data Benchmarking (WBDB)*, pages 45–60. Springer, 2014. (cit. on p. 169)

- [DHI12] AnHai Doan, Alon Halevy, and Zachary Ives. *Principles of data integration*. Elsevier, 2012. (cit. on p. 3)
- [Dia01] Cláudia Dias. Corporate portals: a literature review of a new concept in information management. *International Journal of Information Management*, 21(4):269–287, 2001. (cit. on p. 3)
- [DPW06] Gero Decker, Frank Puhlmann, and Mathias Weske. Formalizing service interactions. In *International Conference on Business Process Management (BPM)*, pages 414–419. Springer, 2006. (cit. on p. 130)
- [DSC⁺14] Qiong Dai, Da-Wen Sun, Jun-Hu Cheng, Hongbin Pu, Xin-An Zeng, and Zhenjie Xiong. Recent advances in de-noising methods and their applications in hyperspectral image processing for the food industry. *Comprehensive Reviews in Food Science and Food Safety*, 13(6):1207–1218, 2014. (cit. on p. 255)
- [DWC10] Tharam Dillon, Chen Wu, and Elizabeth Chang. Cloud computing: issues and challenges. In *International conference on advanced information networking and applications*, pages 27–33. IEEE, 2010. (cit. on p. 2)
- [DYZ⁺08] Wu Deng, Xinhua Yang, Huimin Zhao, Dan Lei, and Hua Li. Study on EAI based on web services and SOA. In *International Symposium on Electronic Commerce and Security*, pages 95–98, 2008. (cit. on p. 33)
- [Eas10] Roger L. Jr. Easton. Fundamentals of digital image processing. Center for Imaging Science, 2010. (cit. on pp. 249 and 250)
- [EBA⁺11] Hadi Esmaeilzadeh, Emily Blem, Renee St Amant, Karthikeyan Sankaralingam, and Doug Burger. Dark silicon and the end of multicore scaling. In *38th Annual International Symposium on Computer Architecture (ISCA)*, pages 365–376. IEEE, 2011. (cit. on p. 221)
- [EEPT06] Hartmut Ehrig, Karsten Ehrig, Ulrike Prange, and Gabriele Taentzer. *Fundamentals of Algebraic Graph Transformation*. Monographs in Theoretical Computer Science. Springer, 2006. (cit. on pp. 106, 141, and 142)
- [EFGK03] Patrick Th Eugster, Pascal A Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The many faces of publish/subscribe. *ACM computing surveys (CSUR)*, 35(2):114–131, 2003. (cit. on p. 3)
- [EGJW17] David Eyers, Avigdor Gal, Hans-Arno Jacobsen, and Matthias Weidlich. Integrating Process-Oriented and Event-Based Systems (Dagstuhl Seminar 16341). *Dagstuhl Reports*, 6(8):21–64, 2017. (cit. on p. 137)
- [EH06] Mahmoud M El-Halwagi. *Process integration*, volume 7. Elsevier, 2006. (cit. on p. 3)
- [EHI10] F. El-Hassan and D. Ionescu. A hardware architecture of an XML/XPath broker for content-based publish/subscribe systems. In *International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, pages 138–143, 2010. (cit. on pp. 222, 271, and 272)
- [EJ17] Geoffrey Elliott and Hans-Arno Jacobsen. Online reconfigurable line-rate matching of filter rules. In *International Workshop on Active Middleware on Modern Hardware, ACTIVE@Middleware*, page 11, 2017. (cit. on p. 278)

- [Ekm93] Paul Ekman. Facial expression and emotion. *American psychologist*, 48(4):384, 1993. (cit. on p. 252)
- [EN10] Ramez Elmasri and Shamkant Navathe. *Fundamentals of database systems*. Addison-Wesley Publishing Company, 2010. (cit. on p. 206)
- [Eng18] Engineering and Physical Sciences Research Council. UKRI Artificial Intelligence Centres for Doctoral Training: Priority Area Descriptions. <https://epsrc.ukri.org/files/funding/calls/2018/ai-2018-cdt-priority-area-document/>, 2018. (cit. on p. 134)
- [EPS73] Hartmut Ehrig, Michael Pfender, and Hans Jürgen Schneider. Graph-grammars: An algebraic approach. In *Switching and Automata Theory*, pages 167–180, 1973. (cit. on pp. 141, 142, and 143)
- [ERK99] Hartmut Ehrig, Grzegorz Rozenberg, and Hans-Jörg Kreowski. *Handbook of graph grammars and computing by graph transformation*, volume 3. world Scientific, 1999. (cit. on pp. 106, 141, and 142)
- [ES13] Christoph Emmersberger and Florian Springer. Tutorial: Open source enterprise application integration-introducing the event processing capabilities of Apache Camel. In *International conference on Distributed event-based systems (DEBS)*, pages 259–268. ACM, 2013. (cit. on pp. 204 and 287)
- [FBBL14] Christoph Fehling, Johanna Barzen, Uwe Breitenbücher, and Frank Leymann. A process for pattern identification, authoring, and application. In *European Conference on Pattern Languages of Programs (EuroPLoP)*, page 4. ACM, 2014. (cit. on pp. 20, 26, 27, and 64)
- [FBF11] Elie Fares, Jean-Paul Bodeveix, and Mamoun Filali. Verification of timed BPEL 2.0 models. In *Enterprise, Business-Process and Information Systems Modeling (BPMDS)*, pages 261–275. Springer, 2011. (cit. on p. 132)
- [FCP⁺12] Franz Färber, Sang Kyun Cha, Jürgen Primsch, Christof Bornhövd, Stefan Sigg, and Wolfgang Lehner. SAP HANA database: data management for modern business applications. *International Conference on Management of Data (SIGMOD)*, 40(4):45–51, 2012. (cit. on pp. 203 and 220)
- [FG12] Dirk Fahland and Christian Gierds. Using Petri nets for modeling enterprise integration patterns. *Technical Report BPM Center Report BPM-12-18*, 2012. (cit. on pp. 9, 24, 31, 36, 37, 62, and 68)
- [FG13] Dirk Fahland and Christian Gierds. Analyzing and completing middleware designs for enterprise integration using coloured Petri nets. In *International Conference on Advanced Information Systems Engineering (CAiSE)*, pages 400–416, 2013. (cit. on pp. 2, 6, 9, 19, 24, 31, 36, 37, 62, 64, 65, 66, 68, 89, 90, 95, 98, 102, 130, and 131)
- [FGH⁺98] David Ford, Lars-Erik Gadde, Håkan Håkansson, Anders Lundgren, Ivan Snehota, Peter Turnbull, David Wilson, Industrial Marketing, and Purchasing Group. *Managing business relationships*. J. Wiley, 1998. (cit. on p. 1)
- [Fir07] Joseph M Firestone. *Enterprise information portals and knowledge management*. Routledge, 2007. (cit. on p. 3)

- [FL01] Berndt Farwer and Irina A. Lomazova. A systematic approach towards object-based Petri net formalisms. In *Perspectives of System Informatics (PSI), Revised Papers*, pages 255–267, 2001. (cit. on p. 89)
- [FLP⁺07] Vincent W Freeh, David K Lowenthal, Feng Pan, Nandini Kappiah, Rob Springer, Barry L Rountree, and Mark E Femal. Analyzing the energy-time trade-off in high-performance computing applications. *IEEE Transactions on Parallel & Distributed Systems*, 18(6):835–848, 2007. (cit. on p. 278)
- [FLR⁺14] Christoph Fehling, Frank Leymann, Ralph Retter, Walter Schupeck, and Peter Arbitter. *Cloud Computing Patterns - Fundamentals to Design, Build, and Manage Cloud Applications*. Springer, 2014. (cit. on pp. 1, 19, 22, 45, 49, 50, and 51)
- [FLS63] Richard P Feynman, Robert B Leighton, and Matthew Sands. The Feynman lectures, vol. 1, 1963. (cit. on pp. 167 and 168)
- [FML⁺12] Franz Färber, Norman May, Wolfgang Lehner, Philipp Große, Ingo Müller, Hannes Rauhe, and Jonathan Dees. The SAP HANA database – an architecture overview. *IEEE Data Eng. Bull.*, 35(1):28–33, 2012. (cit. on pp. 203, 209, and 220)
- [Fon16] Brendan Fong. *The algebra of open and interconnected systems*. PhD thesis, University of Oxford, 2016. (cit. on pp. 110 and 112)
- [For16a] Forrester Research, Inc. The Forrester Wave: Hybrid Integration For Enterprises, Q4 2016. <https://www.forrester.com/report/The+Forrester+Wave+Hybrid+Integration+For+Enterprises+Q4+2016/-/E-RES131101>, 2016. (cit. on pp. 1, 5, and 54)
- [For16b] Forrester Research, Inc. The Forrester Wave: iPaaS For Dynamic Integration, Q3 2016. <https://www.forrester.com/report/The+Forrester+Wave+iPaaS+For+Dynamic+Integration+Q3+2016/-/E-RES115619>, 2016. (cit. on pp. 4 and 38)
- [For16c] Forrester Research, Inc. The Top 10 Technology Trends To Watch: 2016 To 2018. <https://www.forrester.com/report/The+Top+10+Technology+Trends+To+Watch+2016+To+2018/-/E-RES120075>, 2016. (cit. on p. 4)
- [Fow02] Martin Fowler. *Patterns of enterprise application architecture*. Addison-Wesley Longman Publishing Co., Inc., 2002. (cit. on p. 19)
- [FPM⁺18] Daniel Firestone, Andrew Putnam, Sambhrama Mundkur, Derek Chiou, Alireza Dabagh, Mike Andrewartha, Hari Angepat, Vivek Bhanu, Adrian Caulfield, Eric Chung, et al. Azure accelerated networking: Smartnics in the public cloud. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2018. (cit. on p. 222)
- [FRQC11] Rafael Z Frantz, Antonia M Reina Quintero, and Rafael Corchuelo. A domain-specific language to design enterprise application integration solutions. *International Journal of Cooperative Information Systems*, 20(02):143–176, 2011. (cit. on p. 131)

- [FSLM06] Marina Fisher, Sonu Sharma, Ray Lai, and Laurence Moroney. *Java EE and Net Interoperability: Integration Strategies, Patterns, and Best Practices*. Prentice Hall Professional, 2006. (cit. on pp. 31 and 34)
- [FU80] R. W. Floyd and J. D. Ullman. The compilation of regular expressions into integrated circuits. In *Annual Symposium on Foundations of Computer Science*, pages 260–269, 1980. (cit. on pp. 235 and 271)
- [Gar13] Nishant Garg. *Apache Kafka*. Packt Publishing Ltd, 2013. (cit. on p. 204)
- [Gar16] Gartner, Inc. Magic Quadrant for Enterprise Integration Platform as a Service, Worldwide. <https://www.gartner.com/doc/3263719/magic-quadrant-enterprise-integration-platform>, 2016. (cit. on pp. 4 and 38)
- [Gar17] Gartner, Inc. Gartner Newsroom Emerging Technologies from 2005 To 2017. <http://www.gartner.com/newsroom/id/492152>, <http://www.gartner.com/newsroom/id/495475>, <http://www.slideshare.net/dinhledat/dinh-ledat-top-10-technology-trends-20072014-gartner>, <http://www.gartner.com/newsroom/id/530109>, <http://www.gartner.com/newsroom/id/777212>, <http://www.gartner.com/newsroom/id/1210613>, <http://www.gartner.com/newsroom/id/1454221>, <http://www.gartner.com/newsroom/id/1826214>, <http://www.gartner.com/newsroom/id/2209615>, <http://www.gartner.com/newsroom/id/2603623>, <http://www.gartner.com/newsroom/id/2867917>, <http://www.gartner.com/newsroom/id/3143521>, <http://www.gartner.com/newsroom/id/3482617>, 2017. (cit. on pp. 4, 9, 31, 37, 44, and 49)
- [GBB⁺08] Patrick Giroux, Stephan Brunessaux, Sylvie Brunessaux, Jérémie Doucy, Gérard Dupont, Bruno Grilhaes, Yann Mombrun, and Arnaud Saval. Weblab: An integration infrastructure to ease the development of multimedia processing applications. In *International Conference on Software & Systems Engineering and their Applications (ICSSEA)*, pages 129–138, 2008. (cit. on p. 273)
- [GC15] Raman Grover and Michael J Carey. Data ingestion in AsterixDB. In *International Conference on Extending Database Technology (EDBT)*, pages 605–616, 2015. (cit. on p. 209)
- [GCKP13] Martin Grund, Philippe Cudré-Mauroux, Jens Krüger, and Hasso Plattner. Hybrid graph and relational query processing in main memory. In *International Conference on Data Engineering (ICDE) Workshops*, pages 23–24, 2013. (cit. on p. 209)
- [GDP09] Veronica Gacitua-Decar and Claus Pahl. Ontology-based patterns for the integration of business processes and enterprise application architectures. *Semantic Enterprise Application Integration for Business Processes: Service-Oriented Frameworks*, IGI Publishers, pages 36–60, 2009. (cit. on pp. 31 and 32)
- [Geo11] Lars George. *HBase: The Definitive Guide*. O’Reilly, 2011. (cit. on p. 192)
- [Get11] Janusz R. Getta. Static optimization of data integration plans in global information systems. In *International Conference on Enterprise Information Systems (ICEIS)*, pages 141–150, 2011. (cit. on pp. 139, 140, and 141)

- [GGL05] Roberto Gorrieri, Claudio Guidi, and Roberto Lucchi. Reasoning about interaction patterns in choreography. In *Formal Techniques for Computer Systems and Business Processes*, pages 333–348. Springer, 2005. (cit. on p. 130)
- [GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley Longman Publishing Co., Inc., 1995. (cit. on pp. 19, 51, and 131)
- [GHS95] Diimitrios Georgakopoulos, Mark Hornick, and Amit Sheth. An overview of workflow management: From process modeling to workflow automation infrastructure. *Distributed and parallel Databases*, 3(2):119–153, 1995. (cit. on p. 3)
- [GKRH13] Denise Ganly, Andy Kyte, Nigel Rayner, and Carol Hardcastle. Predicts 2014: The rise of the postmodern ERP and enterprise applications world, 2013. (cit. on pp. 1 and 2)
- [GL17] Paolo Guagliardo and Leonid Libkin. A formal semantics of SQL queries, its validation, and applications. *International Conference on Very Large Data Bases (VLDB)*, 11(1):27–39, 2017. (cit. on p. 279)
- [Gla89] Andrew S Glassner. *An introduction to ray tracing*. Elsevier, 1989. (cit. on p. 157)
- [GLW⁺11] Philipp Große, Wolfgang Lehner, Thomas Weichert, Franz Färber, and Wen-Syan Li. Bridging two worlds with rice integrating R into the SAP in-memory computing engine. *International Conference on Very Large Data Bases (VLDB)*, 4(12):1307–1317, 2011. (cit. on pp. 204 and 209)
- [GM03] Dieter Gawlick and Shailendra Mishra. Information sharing with the Oracle database. In *International Conference on Distributed Event-Based Systems (DEBS)*, 2003. (cit. on pp. 206, 209, and 269)
- [Gme12] Oliver Gmelch. *User-Centric Application Integration in Enterprise Portal Systems*. EUL-Verlag, 2012. (cit. on p. 179)
- [GMM⁺16] Michael Grossniklaus, David Maier, James Miller, Sharmadha Moorthy, and Kristin Tufte. Frames: data-driven windows. In *International Conference on Distributed Event-Based Systems (DEBS)*, pages 13–24, 2016. (cit. on p. 230)
- [GMW12] Christian Gierds, Arjan J. Mooij, and Karsten Wolf. Reducing adapter synthesis to controller synthesis. *IEEE Trans. Serv. Comput.*, 5(1):72–85, January 2012. (cit. on pp. 31 and 33)
- [GN05] Venkat N Gudivada and Jagadeesh Nandigam. Enterprise application integration using extensible web services. In *International Conference on Web Services (ICWS)*, pages 41–48, 2005. (cit. on p. 33)
- [GNVV04] Zhi Guo, Walid Najjar, Frank Vahid, and Kees Visser. A quantitative analysis of the speedup factors of FPGAs over processors. In *ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pages 162–170, 2004. (cit. on p. 222)
- [GÖ03] Lukasz Golab and M Tamer Özsu. Issues in data stream management. *International Conference on Management of Data (SIGMOD)*, 32(2):5–14, 2003. (cit. on pp. 4 and 204)

- [Gol18] Diane Golay. The expanding landscape of computer science. *Crossroads (XRDS)*, 25(1):5–6, October 2018. (cit. on p. 62)
- [GR92] Jim Gray and Andreas Reuter. *Transaction processing: concepts and techniques*. Elsevier, 1992. (cit. on pp. 207, 208, 213, and 214)
- [GR11] Laura González and Raúl Ruggia. Addressing QoS issues in service based systems through an adaptive ESB infrastructure. In *Workshop on Middleware for Service Oriented Computing*, page 4, 2011. (cit. on pp. 31, 36, 37, 44, and 57)
- [GR16] S Gopinathan and S Nancy Arokia Rani. The melanoma skin cancer detection and feature extraction through image processing techniques. *International Journal of Emerging Trends & Technology in Computer Science*, 5(4):106–112, 2016. (cit. on pp. 255 and 256)
- [Gra90] Goetz Graefe. Encapsulation of parallelism in the Volcano query processing system. In *International Conference on Management of Data (SIGMOD)*, pages 102–111. ACM, 1990. (cit. on pp. 169 and 172)
- [Gra93] Jim Gray. The benchmark handbook for database and transaction systems. *Morgan Kaufmann*, 1993. (cit. on pp. 168 and 182)
- [Gro97] William I. Grosky. Managing multimedia information in database systems. *Commun. ACM*, 40(12):72–80, December 1997. (cit. on pp. 257 and 273)
- [Gro10] Object Management Group. Business process model and notation (BPMN). *Object management group*, 2010. (cit. on p. 24)
- [GRVT⁺06] Alejandra García-Rojas, Frédéric Vexo, Daniel Thalmann, Amaryllis Raouzaïou, Kostas Karpouzis, S Kollias, Laurent Moccozet, and Nadia Magnenat-Thalmann. Emotional face expression profiles supported by virtual human ontology. *Journal of Visualization and Computer Animation*, 17(3-4):259–269, 2006. (cit. on pp. 251 and 252)
- [GSS07] Georg Grossmann, Michael Schrefl, and Markus Stumptner. Exploiting semantics of inter-process dependencies to instantiate predefined integration patterns. In *International conference on Conceptual modeling (ER)*, pages 155–160, 2007. (cit. on p. 34)
- [GWC⁺07] Daniel Gyllstrom, Eugene Wu, Hee-Jin Chae, Yanlei Diao, Patrick Stahlberg, and Gordon Anderson. SASE: complex event processing over streams (demo). In *Conference on Innovative Data Systems Research (CIDR)*, pages 407–411, 2007. (cit. on p. 271)
- [H⁺08] Jan Hidders et al. DFL: A dataflow language based on Petri nets and nested relational calculus. *Information Systems*, 33(3):261–284, 2008. (cit. on p. 67)
- [HA10] Markus Heller and Matthias Allgaier. Model-based service integration for extensible enterprise systems with adaptation patterns. In *International Conference on e-Business (ICE-B)*, pages 1–6, 2010. (cit. on pp. 9, 31, and 32)
- [HAB⁺05] Alon Y. Halevy, Naveen Ashish, Dina Bitton, Michael J. Carey, Denise Draper, Jeff Pollock, Arnon Rosenthal, and Vishal Sikka. Enterprise information integration: successes, challenges and controversies. In *International Conference on Management of Data (SIGMOD)*, 2005. (cit. on p. 4)

- [Ham04] Imed Hammouda. Towards tool-support for pattern composition. Technical report, Tampere University of Technology, 2004. (cit. on p. 131)
- [HAMR13] Irfan Habib, Ashiq Anjum, Richard Mcclatchey, and Omer Rana. Adapting scientific workflow structures using multi-objective optimization strategies. *TAAS*, 8(1):4, 2013. (cit. on pp. 139 and 140)
- [Han93] Hans-Michael Hanisch. Analysis of place/transition nets with timed arcs and its application to batch process control. In *International Conference on Application and Theory of Petri Nets (PN)*, pages 282–299. Springer, 1993. (cit. on pp. 68 and 73)
- [Han12] Robert S Hanmer. Pattern mining patterns. In *Conference on Pattern Languages of Programs (PLoP)*, page 23. The Hillside Group, 2012. (cit. on pp. 20, 26, 27, and 64)
- [Har97] Reiner Hartenstein. The microprocessor is no longer general purpose: why future reconfigurable platforms will win. In *International Conference on Innovative Systems in Silicon*, pages 2–12. IEEE, 1997. (cit. on pp. 204 and 221)
- [Har99] Neil B Harrison. The language of shepherding. *Pattern languages of program design*, 5:507–530, 1999. (cit. on pp. 20 and 26)
- [HB11] Tom Heath and Christian Bizer. Linked data: Evolving the web into a global data space. *Synthesis lectures on the semantic web: theory and technology*, 1(1):1–136, 2011. (cit. on p. 251)
- [HC10] Alan Hevner and Samir Chatterjee. Design science research in information systems. In *Design research in information systems*, pages 9–22. Springer, 2010. (cit. on pp. 12 and 168)
- [HCDG⁺12] Babak Bagheri Hariri, Diego Calvanese, Giuseppe De Giacomo, Riccardo De Masellis, Paolo Felli, and Marco Montali. Verification of description logic knowledge and action bases. In *European Conference on Artificial Intelligence (ECAI)*, volume 242, pages 103–108, 2012. (cit. on p. 277)
- [HCM⁺13] Babak Bagheri Hariri, Diego Calvanese, Marco Montali, Giuseppe De Giacomo, Riccardo De Masellis, and Paolo Felli. Description logic knowledge and action bases. *Journal of Artificial Intelligence Research*, 46:651–686, 2013. (cit. on p. 277)
- [HDX14] Wu He and Li Da Xu. Integration of distributed enterprise applications: a survey. *IEEE Transactions on Industrial Informatics*, 10(1):35–42, 2014. (cit. on pp. 31, 37, 38, and 49)
- [Her03] Klaudia Hergula. *Daten- und Funktionsintegration durch föderierte Datenbanksysteme*. PhD thesis, University of Kaiserslautern, 2003. (cit. on p. 270)
- [HFKV08] Rainer Friedrich Hauser, Michael Friess, Jochen Malte Kuster, and Jussi Vanhatalo. An incremental approach to the analysis and transformation of workflows using region trees. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 38(3):347–359, 2008. (cit. on p. 161)

- [HGV⁺08] Martin C Herbordt, Yongfeng Gu, Tom VanCourt, Josh Model, Bharat Sukhwani, and Matt Chiu. Computing models for FPGA-based accelerators. *Computing in science & engineering*, 10(6):35–45, 2008. (cit. on p. 227)
- [HK07] Imed Hammouda and Kai Koskimies. An approach for structural pattern composition. In *International Conference on Software Composition*, pages 252–265. Springer, 2007. (cit. on p. 131)
- [HKWY97] Laura M. Haas, Donald Kossmann, Edward L. Wimmers, and Jun Yang. Optimizing queries across diverse data sources. In *International Conference on Very Large Data Bases (VLDB)*, pages 276–285, 1997. (cit. on p. 270)
- [HM08] Mark D Hill and Michael R Marty. Amdahl’s law in the multicore era. *Computer*, 41(7), 2008. (cit. on p. 221)
- [HMPR04] Alan R. Hevner, Salvatore T. March, Jinsoo Park, and Sudha Ram. Design science in information systems research. *MIS Quarterly*, 28(1):75–105, March 2004. (cit. on p. 12)
- [HMWMS87] Theo Härder, Klaus Meyer-Wegener, Bernhard Mitschang, and Andrea Sikeler. PRIMA – a DBMS prototype supporting engineering applications. In *International Conference on Very Large Data Bases (VLDB)*, pages 433–442, 1987. (cit. on pp. 206 and 270)
- [Hoh02] Gregor Hohpe. Enterprise integration patterns. In *Conference on Pattern Language of Programs (PLOP)*. Citeseer, 2002. (cit. on pp. 2, 6, 19, 20, 21, and 26)
- [Hoh05] Gregor Hohpe. Your coffee shop doesn’t use two-phase commit. *IEEE Software*, 22(2):64–66, March 2005. (cit. on pp. 31, 38, 39, and 44)
- [Hoh06] Gregor Hohpe. Conversation patterns. In *The Role of Business Processes in Service Oriented Architectures*, number 06291 in Dagstuhl Seminar Proceedings, page 7, 2006. (cit. on pp. 19, 25, 31, 36, 38, 41, 44, 49, 50, 52, and 53)
- [HOS⁺15] Michael Heiss, Andreas Oertl, Monika Sturm, Peter Palensky, Stefan Vielguth, and Florian Nadler. Platforms for industrial cyber-physical systems integration: contradicting requirements as drivers for innovation. In *Modeling and Simulation of Cyber-Physical Energy Systems*, pages 1–8, 2015. (cit. on pp. 31, 35, 37, and 49)
- [HP02] Annegret Habel and Detlef Plump. Relabelling in graph transformation. In *International Conference on Graph Transformation (ICGT)*, volume 2505, pages 135–147. Springer, 2002. (cit. on p. 143)
- [HPG⁺12] Chris Howard, Daryl C Plummer, Yvonne Genovese, Jeffrey Mann, David A Willis, and D Mitchell Smith. The nexus of forces: Social, mobile, cloud and information. *Gartner, Inc. Report G*, 234840:14, 2012. (cit. on p. 1)
- [HSP13] Steve Harris, Andy Seaborne, and Eric Prud’hommeaux. SPARQL 1.1 query language. *W3C recommendation*, 21(10), 2013. (cit. on pp. 251, 252, and 253)
- [HSS07] Ying Hu, Seema Sundara, and Jagannathan Srinivasan. Supporting time-constrained SQL queries in Oracle. In *International Conference on Very Large Data Bases (VLDB)*, pages 1207–1218, 2007. (cit. on p. 79)

- [HST05] Joachim Hammer, Michael Stonebraker, and Oguzhan Topsakal. THALIA: test harness for the assessment of legacy information integration approaches. In *International Conference on Data Engineering (ICDE)*, 2005. (cit. on p. 187)
- [HW04] Gregor Hohpe and Bobby Woolf. *Enterprise integration patterns: Designing, building, and deploying messaging solutions*. Addison-Wesley, 2004. (cit. on pp. 2, 3, 4, 6, 19, 20, 21, 22, 23, 24, 29, 30, 31, 37, 38, 41, 44, 45, 46, 50, 51, 52, 53, 56, 58, 62, 63, 65, 68, 80, 82, 89, 107, 131, 132, 137, 154, 155, 168, 169, 172, 178, 182, 187, 195, 196, 204, 212, 230, 233, 237, 245, 246, 249, 255, 256, 259, 268, and 272)
- [HZ06] Carsten Hentrich and Uwe Zdun. Patterns for business object model integration in process-driven and service-oriented architectures. In *Conference on Pattern Languages of Programs (PLoP)*, page 23, 2006. (cit. on pp. 31 and 32)
- [HZ07] Carsten Hentrich and Uwe Zdun. Service integration patterns for invoking services from business processes. In *European Conference on Pattern Languages of Programs (EuroPLoP)*, pages 235–278, 2007. (cit. on pp. 31 and 32)
- [HZ09] Carsten Hentrich and Uwe Zdun. A pattern language for process execution and integration design in service-oriented architectures. In *Transactions on Pattern Languages of Programming I*, pages 136–191. Springer, 2009. (cit. on pp. 31 and 32)
- [IA10] Claus Ibsen and Jonathan Anstey. *Camel in Action*. Manning, 2010. (cit. on pp. 2, 25, 38, 39, 47, 48, 131, 155, 169, 170, 171, 172, 181, 191, 194, 195, 204, 219, 235, 236, 246, 248, and 263)
- [IBM17] IBM. WebSphere Cast Iron Cloud integration. <https://www.ibm.com/support/knowledgecenter/SSGR73>, 2017. (cit. on pp. 2, 38, 39, 47, and 48)
- [IBM18] IBM. Building trust in AI. <https://www.ibm.com/watson/advantage-reports/future-of-artificial-intelligence/building-trust-in-ai.html>, 2018. (cit. on p. 134)
- [IC⁺15] Muhammad Imran, Carlos Castillo, et al. Processing social media messages in mass emergency: A survey. *ACM Comput. Surv.*, 47(4):67:1–67:38, 2015. (cit. on pp. 255 and 256)
- [Inf17] Informatica. Cloud-Integration. <https://www.informatica.com/de/products/cloud-integration.html>, 2017. (cit. on pp. 38, 39, 46, and 47)
- [IPE14] Alexandru Iosup, Radu Prodan, and Dick Epema. IaaS cloud benchmarking: approaches, challenges, and experience. In *Cloud Computing for Data-Intensive Applications*, pages 83–104. Springer, 2014. (cit. on p. 199)
- [IS15] Muhammad Hussain Iqbal and Tariq Rahim Soomro. Big data analysis: Apache Storm perspective. *International journal of computer trends and technology*, 19(1):9–14, 2015. (cit. on p. 204)
- [Jai89] Anil K Jain. *Fundamentals of digital image processing*. Englewood Cliffs, NJ: Prentice Hall, 1989. (cit. on pp. 249 and 250)
- [JBC⁺14] Jelena Jovanovic, Ebrahim Bagheri, John Cuzzola, Dragan Gasevic, Zoran Jeremic, and Reza Bashash. Automated semantic tagging of textual content. *IT Professional*, 16(6):38–46, 2014. (cit. on p. 265)

- [JDM⁺01] Radu S Jasinschi, Nevenka Dimitrova, Thomas McGee, Lalitha Agnihotri, John Zimmerman, and Dongge Li. Integrated multimedia processing for topic segmentation and classification. In *International Conference on Image Processing (ICIP)*, volume 3, pages 366–369, 2001. (cit. on p. 274)
- [Jit17] Jitterbit. Harmony Cloud Integration. <https://www.jitterbit.com/harmony/>, 2017. (cit. on pp. 38, 39, 47, and 48)
- [JJMS11] Lasse Jacobsen, Morten Jacobsen, Mikael H. Møller, and Jiří Srba. Verification of timed-arc Petri nets. In *SOFSEM*, pages 46–72. Springer, 2011. (cit. on pp. 67, 69, 73, 74, 75, 78, and 130)
- [JKB13] Monika Jhuria, Ashwani Kumar, and Rushikesh Borse. Image processing for smart farming: Detection of disease and fruit grading. In *International Conference on Image Information Processing (ICIIP)*, pages 521–526, 2013. (cit. on pp. 255 and 256)
- [JKW07] Kurt Jensen, Lars Michael Kristensen, and Lisa Wells. Coloured Petri nets and CPN Tools for modelling and validation of concurrent systems. *International Journal on Software Tools for Technology Transfer*, 9(3-4):213–254, 2007. (cit. on pp. 90 and 122)
- [JNB14] Ashok Joshi, Raghunath Nambiar, and Michael Brey. Benchmarking internet of things solutions. In *Workshop on Big Data Benchmarking (WBDB)*, 2014. (cit. on p. 169)
- [JnYzZ11] Yang Juan-ning, Wang Ying-zhuo, and Yong-yuan Zhang. DSP-based image processing technology applied to online detection of pharmaceutical industry. *Internet of Things Technologies*, 8:034, 2011. (cit. on p. 255)
- [JZGH11] Jian-min Jiang, Shi Zhang, Ping Gong, and Zhong Hong. Message dependency-based adaptation of services. In *Asia-Pacific Services Computing Conference (APSCC)*, pages 442–449. IEEE, 2011. (cit. on pp. 31 and 34)
- [KB12] Eliana Kaneshima and Rosana T Vaccare Braga. Patterns for enterprise application integration. In *Latin-American Conference on Pattern Languages of Programming*, page 2, 2012. (cit. on pp. 31 and 32)
- [KCM04] Graham Klyne, Jeremy J Carroll, and Brian McBride. Resource description framework (RDF): Concepts and abstract syntax. W3C recommendation. <https://www.w3.org/TR/2004/REC-rdf-concepts-20040210/>, 2004. (cit. on p. 250)
- [KE11] Alfons Kemper and André Eickler. *Datenbanksysteme - Eine Einführung*, 8. Auflage. Oldenbourg, 2011. (cit. on pp. 4, 206, 207, and 209)
- [Kel02] Wolfgang Keller. Enterprise application integration. *Erfahrungen aus der Praxis. dpunkt*, page 21, 2002. (cit. on pp. 2 and 6)
- [KG14] Georgia Kougka and Anastasios Gounaris. Optimization of data-intensive flows: Is it needed? Is it solved? In *International Workshop on Data Warehousing and OLAP (DOLAP)*, pages 95–98. ACM, 2014. (cit. on pp. 136, 137, 139, and 160)
- [KGM12] Norbert Koppenhagen, Oliver Gaß, and Benjamin Müller. Design science research in action-anatomy of success critical activities for rigor and relevance. Technical report, University of Mannheim, 2012. (cit. on p. 12)

- [KGS17] Georgia Kougka, Anastasios Gounaris, and Alkis Simitsis. The many faces of data-centric workflow optimization: A survey. *CoRR*, abs/1701.07723, 2017. (cit. on pp. 139 and 160)
- [KH06] Till Köllmann and Carsten Hentrich. Synchronization patterns for process-driven and service-oriented architectures. In *European Conference on Pattern Languages of Programs (EuroPLoP)*, pages 199–228, 2006. (cit. on pp. 31 and 34)
- [Kit04] Barbara Kitchenham. Procedures for performing systematic reviews. Technical Report TR/SE-0401, Keele University, Keele, UK, 2004. (cit. on pp. 29, 38, 139, and 253)
- [KJM⁺11] Ryan KL Ko, Peter Jagadpramana, Miranda Mowbray, Siani Pearson, Markus Kirchberg, Qianhui Liang, and Bu Sung Lee. Trustcloud: A framework for accountability and trust in cloud computing. In *World Congress on Services*, pages 584–588. IEEE, 2011. (cit. on p. 2)
- [KKL15] David Kernert, Frank Köhler, and Wolfgang Lehner. Bringing linear algebra objects to life in a column-oriented in-memory database. In *In Memory Data Management and Analysis*, pages 44–55. Springer, 2015. (cit. on pp. 204 and 209)
- [KKLW80] David Kuck, Robert Kuhn, Bruce Leasure, and Michael J Wolfe. Analysis and transformation of programs for parallel computation. In *International Computer Software & Applications Conference (COMPSAC)*, pages 709–715. IEEE, 1980. (cit. on p. 161)
- [KKP⁺81] David J Kuck, Robert H Kuhn, David A Padua, Bruce Leasure, and Michael Wolfe. Dependence graphs and compiler optimizations. In *Symposium on Principles of programming languages (POPL)*, pages 207–218. ACM, 1981. (cit. on p. 161)
- [KKP12] Alexandros Karargyris, Orestis Karargyris, and Alexandros Pantelopoulous. Derma/care: An advanced image-processing mobile application for monitoring skin cancer. In *IEEE International Conference on Tools with Artificial Intelligence*, volume 2, pages 1–7, 2012. (cit. on pp. 255 and 256)
- [Kle13] Jiri J Klemes. *Handbook of process integration (PI): minimisation of energy and water use, waste and emissions*. Elsevier, 2013. (cit. on p. 3)
- [KMC72] David J Kuck, Yoichi Muraoka, and Shyh-Ching Chen. On the number of operations simultaneously executable in Fortran-like programs and their resulting speedup. *IEEE Transactions on Computers*, 100(12):1293–1310, 1972. (cit. on p. 161)
- [KMR00] Ekkart Kindler, Axel Martens, and Wolfgang Reisig. Inter-operability of workflow applications: Local criteria for global soundness. In *International Conference on Business Process Management (BPM)*, pages 235–253. Springer, 2000. (cit. on pp. 110 and 132)
- [KMS12] Aleks Kissinger, Alex Merry, and Matvey Soloviev. Pattern graph rewrite systems. In *Developments in Computational Models (DCM)*, pages 54–66, 2012. (cit. on pp. 114 and 143)

- [KN10] Alfons Kemper and Thomas Neumann. One size fits all, again! The architecture of the hybrid OLTP&OLAP database management system HyPer. In *International Computer Software & Applications Conference (BIRTE)*, volume 84 of *Lecture Notes in Business Information Processing*, pages 7–23. Springer, 2010. (cit. on p. 209)
- [KN11] Alfons Kemper and Thomas Neumann. HyPer: A hybrid OLTP&OLAP main memory database system based on virtual memory snapshots. In *International Conference on Data Engineering (ICDE)*, pages 195–206. IEEE Computer Society, 2011. (cit. on p. 209)
- [Knu74] Donald E Knuth. Computer programming as an art. *Communications of the ACM*, 17(12):667–673, 1974. (cit. on pp. 135 and 136)
- [Kol33] A. N Kolmogorov. Sulla determinazione empirica di una legge di distribuzione. *Giorn. Ist. It. Attuari Giorn.*, 4, 83, 91, 1933. (cit. on p. 86)
- [KRA⁺16] Alina Kuznetsova, Hassan Rom, Neil Alldrin, Jasper R. R. Uijlings, Ivan Krasin, Jordi Pont-Tuset, Shahab Kamali, Stefan Popov, Matteo Mallocci, Tom Duerig, and Vittorio Ferrari. OpenImages: A public dataset for large-scale multi-label and multi-class image classification. *Dataset available from <https://github.com/openimages>*, 2016. (cit. on p. 267)
- [KS08] Akhil Kumar and Zhe Shan. Algorithms based on pattern analysis for verification and adapter creation for business process composition. In *OTM Confederated International Conferences*, pages 120–138, 2008. (cit. on pp. 31 and 34)
- [KSNK13] Rick Kazman, Klaus Schmid, Claus Ballegaard Nielsen, and John Klein. Understanding patterns for system of systems integration. In *System of Systems Engineering*, pages 141–146, 2013. (cit. on p. 36)
- [KWG13] Avita Katal, Mohammad Wazid, and RH Goudar. Big data: issues, challenges, tools and good practices. In *International conference on contemporary computing (IC3)*, pages 404–409. IEEE, 2013. (cit. on p. 2)
- [Las16] Slawomir Lasota. Decidability border for Petri nets with data: WQO dichotomy conjecture. In *International Conference on Application and Theory of Petri Nets (PN)*, pages 20–36. Springer, 2016. (cit. on pp. 67 and 77)
- [LCL05] Rikard Land, Ivica Crnkovic, and Stig Larsson. Process patterns for software systems in-house integration and merge-experiences from industry. In *Conference on Software Engineering and Advanced Applications (CAiSE)*, pages 180–187, 2005. (cit. on p. 36)
- [Len02] Maurizio Lenzerini. Data integration: A theoretical perspective. In *International Conference on Management of Data (SIGMOD)*, pages 233–246. ACM, 2002. (cit. on pp. 3 and 4)
- [LG03] David Lomet and Johannes Gehrke. Special issue on data stream processing. *Bull. Technical Committee on Data Eng*, 26(1), 2003. (cit. on p. 204)
- [LGG⁺18] Shangyu Luo, Zekai Gao, Michael Gubanov, Luis Leopoldo Perez, and Chris Jermaine. Scalable linear algebra on a relational database system. *IEEE Transactions on Knowledge and Data Engineering*, 2018. (cit. on p. 209)

- [Lin00] David S Linthicum. *Enterprise application integration*. Addison-Wesley Professional, 2000. (cit. on pp. 2, 6, 7, 41, 55, 58, 178, 182, 204, 214, 246, 248, and 273)
- [Lin03] David S Linthicum. *Next generation application integration: from simple information to Web services*. Addison-Wesley Longman Publishing Co., Inc., 2003. (cit. on pp. 2, 6, and 7)
- [LJVWF04] David Lacey, Neil D Jones, Eric Van Wyk, and Carl Christian Frederiksen. Compiler optimization correctness by temporal logic. *Higher-Order and Symbolic Computation*, 17(3):173–206, 2004. (cit. on p. 161)
- [LLX⁺09] Yan Liu, Xin Liang, Lingzhi Xu, Mark Staples, and Liming Zhu. Using architecture integration patterns to compose enterprise mashups. In *Software Architecture & European Conference on Software Architecture*, pages 111–120, 2009. (cit. on pp. 31 and 33)
- [LLX⁺11] Yan Liu, Xin Liang, Lingzhi Xu, Mark Staples, and Liming Zhu. Composing enterprise mashup components and services using architecture integration patterns. *Journal of Systems and Software*, 84(9):1436–1446, 2011. (cit. on pp. 31 and 33)
- [LMSW08] Niels Lohmann, Peter Massuthe, Christian Stahl, and Daniela Weinberg. Analyzing interacting WS-BPEL processes using flexible model generation. *Data & Knowledge Engineering*, 64(1):38–54, 2008. (cit. on pp. 31, 34, and 38)
- [LÖON01] Shu Lin, M. Tamer Özsu, Vincent Oria, and Raymond T. Ng. An extendible hash for multi-precision similarity querying of image databases. In *International Conference on Very Large Data Bases (VLDB)*, pages 221–230, 2001. (cit. on pp. 246 and 273)
- [LSDJ06] Michael S. Lew, Nicu Sebe, Chabane Djeraba, and Ramesh Jain. Content-based multimedia information retrieval: State of the art and challenges. *ACM Trans. Multimedia Comput. Commun. Appl.*, 2(1):1–19, February 2006. (cit. on pp. 246, 248, and 259)
- [LSO⁺08] Hyowon Lee, Alan F Smeaton, Noel E O’Connor, Gareth Jones, Michael Blighe, Daragh Byrne, Aiden Doherty, and Cathal Gurrin. Constructing a SenseCam visual diary as a media process. *Multimedia Systems*, 14(6):341–349, 2008. (cit. on pp. 246 and 256)
- [M⁺15] Andreas Michaels et al. Vision-based high-speed manipulation for robotic ultra-precise weed control. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 5498–5505. IEEE/RSJ, 2015. (cit. on p. 246)
- [Man09] Dzaharudin Mansor. Moving to the cloud: Patterns, integration challenges and opportunities. In *International Conference on Advances in Mobile Computing and Multimedia*, pages 9–9, 2009. (cit. on pp. 31 and 35)
- [Mar05] Axel Martens. Analyzing web service based business processes. In *International Conference on Fundamental Approaches to Software Engineering*, pages 19–33. Springer, 2005. (cit. on p. 132)

- [MB17] Garrett McGrath and Paul R Brenner. Serverless computing: Design, implementation, and performance. In *International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pages 405–410. IEEE, 2017. (cit. on p. 278)
- [MBM08] Marcelo R. N. Mendes, Pedro Bizarro, and Paulo Marques. A framework for performance evaluation of complex event processing systems. In *International Conference on Distributed Event-Based Systems (DEBS)*, 2008. (cit. on pp. 169, 197, and 198)
- [MBM13] Marcelo R. N. Mendes, Pedro Bizarro, and Paulo Marques. Towards a standard event processing benchmark. In *Workshop on Software and Performance (WOSP)*. ACM/SPEC, 2013. (cit. on pp. 197 and 198)
- [MD97] Doble J Meszaros and Jim Doble. A pattern language for pattern writing. In *International Conference on Pattern languages of program design*, volume 131, page 164, 1997. (cit. on p. 45)
- [Mer74] Philip Merlin. *A study of the recoverability of computer systems*. PhD thesis, University of California, 1974. (cit. on p. 73)
- [Meu95] Regine Meunier. The pipes and filters architecture. In *Pattern languages of program design*, pages 427–440. ACM Press/Addison-Wesley Publishing Co., 1995. (cit. on p. 21)
- [MF76] Philip Merlin and David Farber. Recoverability of communication protocols—implications of a theoretical study. *IEEE transactions on Communications*, 24(9):1036–1043, 1976. (cit. on p. 73)
- [MFPR90] I. S. Mumick, S. J. Finkelstein, Hamid Pirahesh, and Raghu Ramakrishnan. Magic is relevant. In *International Conference on Management of Data (SIGMOD)*, pages 247–258, 1990. (cit. on p. 243)
- [Mic17] Microsoft. BizTalk Server. [https://msdn.microsoft.com/en-us/library/dd547397\(v=bts.10\).aspx](https://msdn.microsoft.com/en-us/library/dd547397(v=bts.10).aspx), 2017. (cit. on pp. 38, 39, 47, and 48)
- [MLC08] James Moscola, John W. Lockwood, and Young H. Cho. Reconfigurable content-based router using hardware-accelerated language parser. *ACM Trans. Design Autom. Electr. Syst.*, 13(2):28:1–28:25, 2008. (cit. on pp. 235 and 271)
- [MLCR12] J Marco Mendes, Paulo Leitão, Armando W Colombo, and Francisco Restivo. High-level Petri nets for the process description and control in service-oriented manufacturing systems. *International Journal of Production Research*, 50(6):1650–1665, 2012. (cit. on p. 131)
- [MLH⁺15] Norman May, Wolfgang Lehner, Shahul Hameed, Nitesh Maheshwari, Carsten Müller, Sudipto Chowdhuri, and Anil K Goel. SAP HANA—from relational OLAP database to big data infrastructure. In *International Conference on Extending Database Technology (EDBT)*, pages 581–592, 2015. (cit. on pp. 203 and 209)
- [MLK10] Christian Mauro, Jan Marco Leimeister, and Helmut Krcmar. Service oriented device integration—an analysis of SOA design patterns. In *Hawaii International Conference on System Sciences (HICSS)*, pages 1–10, 2010. (cit. on pp. 31 and 33)

- [MLZN09] Pavol Mederly, Marián Lekavý, Marek Závodský, and Pavol Návrát. Construction of messaging-based enterprise integration solutions using AI planning. In *IFIP Central and East European Conference on Software Engineering Techniques (CEE-SET)*, pages 16–29, 2009. (cit. on pp. 31, 36, 38, 49, 160, and 286)
- [MN10] Pavol Mederly and Pavol Návrát. Construction of messaging-based integration solutions using constraint programming. In *East-European Conference on Advances in Databases and Information Systems (ADBIS)*, pages 579–582, 2010. (cit. on pp. 31, 36, 38, 49, and 286)
- [Mob02] R Keith Mobley. *An introduction to predictive maintenance*. Elsevier, 2002. (cit. on p. 1)
- [MR16] Marco Montali and Andrey Rivkin. Model checking Petri nets with names using data-centric dynamic systems. *Formal Aspects of Computing*, 28(4):615–641, 2016. (cit. on p. 85)
- [MR17] Marco Montali and Andrey Rivkin. Db-nets: On the marriage of colored Petri nets and relational databases. *T. Petri Nets and Other Models of Concurrency*, 12:91–118, 2017. (cit. on pp. 67, 69, 71, 72, 73, 77, 78, 85, and 114)
- [MRS05] Peter Massuthe, Wolfgang Reisig, and Karsten Schmidt. An operating guideline approach to the SOA. Technical report, Humboldt-Universität zu Berlin, Mathematisch-Naturwissenschaftliche Fakultät II, Institut für Informatik, 2005. (cit. on p. 132)
- [MS95] Salvatore T March and Gerald F Smith. Design and natural science research on information technology. *Decision support systems*, 15(4):251–266, 1995. (cit. on pp. 12 and 168)
- [MS11] Anil Madhavapeddy and Satnam Singh. Reconfigurable data processing for clouds. In *Field-Programmable Custom Computing Machines (FCCM), 2011 IEEE 19th Annual International Symposium on*, pages 141–145. IEEE, 2011. (cit. on p. 222)
- [MSA14] Huvaida Manzoor, Yogeshwar SinghRandhawa, and Ece Deptt Gimet Amritsar. Comparative studies of algorithms using digital image processing in drug industry. *IJSRP*, page 418, 2014. (cit. on pp. 255 and 256)
- [MSHP15] Danny Merkel, Filippas Santas, Andreas Heberle, and Tarmo Ploom. Cloud integration patterns. In *European Conference on Service-Oriented and Cloud Computing*, pages 199–213, 2015. (cit. on pp. 31, 35, 37, 38, and 49)
- [MT10] René Müller and Jens Teubner. FPGAs: a new point in the database design space. In *International Conference on Extending Database Technology (EDBT)*, pages 721–723, 2010. (cit. on pp. 222, 223, 233, 240, and 271)
- [MTA09a] René Müller, Jens Teubner, and Gustavo Alonso. Data processing on FPGAs. *International Conference on Very Large Data Bases (VLDB)*, 2(1):910–921, 2009. (cit. on pp. 222, 237, 242, 244, and 271)
- [MTA09b] René Müller, Jens Teubner, and Gustavo Alonso. Streams on wires - A query compiler for FPGAs. *International Conference on Very Large Data Bases (VLDB)*, 2(1):229–240, 2009. (cit. on pp. 222, 223, 233, and 271)

- [Muc97] Steven Muchnick. *Advanced compiler design implementation*. Morgan Kaufmann, 1997. (cit. on p. 161)
- [Mul19] MuleSoft. Integration: The cloud’s big challenge, 2019. (cit. on pp. 41 and 169)
- [MVB⁺09] Abhishek Mitra, Marcos R. Vieira, Petko Bakalov, Vassilis J. Tsotras, and Walid A. Najjar. Boosting XML filtering through a scalable FPGA-based architecture. In *Conference on Innovative Data Systems Research (CIDR)*, 2009. (cit. on p. 271)
- [MVH04] Deborah L McGuinness and Frank Van Harmelen. OWL web ontology language overview. *W3C recommendation*, 10(10):2004, 2004. (cit. on p. 251)
- [MWA⁺03] Rajeev Motwani, Jennifer Widom, Arvind Arasu, Brian Babcock, Shivnath Babu, Mayur Datar, Gurmeet Manku, Chris Olston, Justin Rosenstein, and Rohit Varma. Query processing, resource management, and approximation in a data stream management system. In *Conference on Innovative Data Systems Research (CIDR)*, pages 245–256, 2003. (cit. on p. 4)
- [N⁺12] Marcus Nagle et al. Non-destructive mango quality assessment using image processing: Inexpensive innovation for the fruit handling industry. In *Conference on International Research on Food Security, Natural Resource Management and Rural Development*, pages 1–4, 2012. (cit. on p. 255)
- [Neu11] Thomas Neumann. Efficiently compiling efficient query plans for modern hardware. *International Conference on Very Large Data Bases (VLDB)*, 4(9):539–550, 2011. (cit. on p. 278)
- [New15] S. Newman. *Building Microservices*. O’Reilly, 2015. (cit. on p. 180)
- [Nie81] Flemming Nielson. Semantic foundations of data flow analysis. *DAIMI Report Series*, 10(131), 1981. (cit. on p. 161)
- [NRM11] Florian Niedermann, Sylvia Radeschütz, and Bernhard Mitschang. Business process optimization using formalized optimization patterns. In *International Conference on Business Information Systems (BIS)*, pages 123–135. Springer, 2011. (cit. on pp. 139, 140, 141, and 161)
- [NS11] Florian Niedermann and Holger Schwarz. Deep business optimization: Making business process optimization theory work in practice. In *Enterprise, Business-Process and Information Systems Modeling*, pages 88–102. Springer, 2011. (cit. on pp. 139 and 161)
- [NSJ15] M. Najafi, M. Sadoghi, and H. A. Jacobsen. Configurable hardware-based streaming architecture using online programmable-blocks. In *International Conference on Data Engineering (ICDE)*, pages 819–830, 2015. (cit. on p. 271)
- [Off13] Carl D Offner. Notes on graph algorithms used in optimizing compilers. *Notes from University of Massachusetts, Boston*, 2013. (cit. on p. 106)
- [Ora17] Oracle. BEA WebLogic Integration. https://docs.oracle.com/cd/E13214_01/wli/docs81/index.html, 2017. (cit. on pp. 38, 39, 47, and 48)

- [ORK⁺15] Kalin Ovtcharov, Olatunji Ruwase, Joo-Young Kim, Jeremy Fowers, Karin Strauss, and Eric S Chung. Accelerating deep convolutional neural networks using specialized hardware. *Microsoft Research Whitepaper*, 2(11), 2015. (cit. on p. 222)
- [OVVDA⁺07] Chun Ouyang, Eric Verbeek, Wil MP Van Der Aalst, Stephan Breutel, Marlon Dumas, and Arthur HM Ter Hofstede. Formal semantics and analysis of control flow in WS-BPEL. *Science of computer programming*, 67(2-3):162–198, 2007. (cit. on pp. 31, 34, and 132)
- [PCC⁺14] Andrew Putnam, Adrian M Caulfield, Eric S Chung, Derek Chiou, Kypros Constantinides, John Demme, Hadi Esmaeilzadeh, Jeremy Fowers, Gopi Prashanth Gopal, Jan Gray, et al. A reconfigurable fabric for accelerating large-scale data-center services. *ACM SIGARCH Computer Architecture News*, 42(3):13–24, 2014. (cit. on pp. 222 and 277)
- [PCM⁺07] Lina Peng, K Selçuk Candan, Christopher Mayer, Karamvir S Chatha, and Kyung Dong Ryu. Optimization of media processing workflows with adaptive operator behaviors. *Multimedia Tools and Applications*, 33(3):245–272, 2007. (cit. on p. 273)
- [PCW05] Ronald Porter, James O Coplien, and Tiffany Winn. Sequences as a basis for pattern language composition. *Science of Computer Programming*, 56(1-2):231–249, 2005. (cit. on p. 131)
- [PD99] Norman W Paton and Oscar Díaz. Active database systems. *ACM Computing Surveys (CSUR)*, 31(1):63–103, 1999. (cit. on pp. 204 and 205)
- [Pel03] Chris Peltz. Web services orchestration and choreography. *IEEE Computer*, 36(10):46–52, 2003. (cit. on p. 44)
- [Pet13] Dana Petcu. Multi-cloud: expectations and current approaches. In *International workshop on Multi-cloud applications and federated clouds*, pages 1–6. ACM, 2013. (cit. on p. 5)
- [PH94] Detlef Plump and Annegret Habel. Graph unification and matching. In *International Workshop on Graph Grammars and Their Application to Computer Science*, pages 75–88, 1994. (cit. on p. 143)
- [PJGM12] Hervé Panetto, Ricardo Jardim-Goncalves, and Arturo Molina. Enterprise integration and networking: theory and practice. *Annual Reviews in Control*, 36(2):284–290, 2012. (cit. on pp. 31 and 37)
- [POP98] Constantine P Papageorgiou, Michael Oren, and Tomaso Poggio. A general framework for object detection. In *International conference on Computer vision*, pages 555–562. IEEE, 1998. (cit. on p. 250)
- [PPSP14] Om P Patri, Anand V Panangadan, Vikrambhai S Sorathia, and Viktor K Prasanna. Semantic management of enterprise integration patterns: A use case in smart grids. In *International Conference on Data Engineering (ICDE) Workshops*, pages 50–55, 2014. (cit. on pp. 31 and 36)

- [PRC14] Meikel Poess, Tilmann Rabl, and Brian Caufield. TPC-DI: the first industry benchmark for data integration. *International Conference on Very Large Data Bases (VLDB)*, 7(13), 2014. (cit. on pp. 169, 197, and 198)
- [PRFD11] Meikel Poess, Tilmann Rabl, Michael Frank, and Manuel Danisch. A PDGF implementation for TPC-H. In *TPC Technology Conference (TPCTC)*, 2011. (cit. on pp. 182 and 198)
- [Pro59] Reese T Prosser. Applications of boolean matrices to the analysis of flow diagrams. In *Eastern Joint IRE-AIEE-ACM Computer Conference*, pages 133–138. ACM, 1959. (cit. on p. 106)
- [PRTV12] Ken Peffers, Marcus Rothenberger, Tuure Tuunanen, and Reza Vaezi. Design science research evaluation. In *International Conference on Design Science Research in Information Systems (DESRIST)*, pages 398–410. Springer, 2012. (cit. on p. 12)
- [PSPP14] Om Prasad Patri, Vikrambhai S Sorathia, Anand V Panangadan, and Viktor K Prasanna. The process-oriented event model (PoEM): A conceptual model for industrial events. In *International Conference on Distributed Event-Based Systems (DEBS)*, pages 154–165, 2014. (cit. on pp. 31 and 34)
- [PTRC07] Ken Peffers, Tuure Tuunanen, Marcus A Rothenberger, and Samir Chatterjee. A design science research methodology for information systems research. *J. Manage. Inf. Syst.*, 24(3):45–77, 2007. (cit. on p. 12)
- [PW92] Dewayne E. Perry and Alexander L. Wolf. Foundations for the study of software architecture. *SIGSOFT Softw. Eng. Notes*, 17(4):40–52, October 1992. (cit. on p. 21)
- [PW05] Frank Puhlmann and Mathias Weske. Using the π -calculus for formalizing workflow patterns. In *International Conference on Business Process Management (BPM)*, pages 153–168. Springer, 2005. (cit. on p. 130)
- [QYZL05] Xiangli Qu, Xuejun Yang, Jingwei Zhong, and Xuefeng Lv. Integration patterns of grid security service. In *International Conference on Parallel and Distributed Computing Applications and Technologies (PDCAT)*, pages 524–528, 2005. (cit. on pp. 31, 33, 37, and 46)
- [Ram73] Chander Ramchandani. *Analysis of asynchronous concurrent systems by timed Petri nets*. PhD thesis, Massachusetts Institute of Technology, 1973. (cit. on pp. 73 and 130)
- [RB14] Daniel Ritter and Jan Bross. Datalogblocks: relational logic integration patterns. In *International Conference on Database and Expert Systems Applications (DEXA)*, pages 318–325, 2014. (cit. on p. 35)
- [RCVB10] Sidhant Rajam, Ruth Cortez, Alexander Vazhenin, and Subhash Bhalla. Design patterns in enterprise application integration for e-learning arena. In *International Conference on Humans and Computers*, pages 81–88, 2010. (cit. on pp. 31, 37, and 49)

- [RDF⁺15] Tilmann Rabl, Manuel Danisch, Michael Frank, Sebastian Schindler, and Hans-Arno Jacobsen. Just can't get enough - Synthesizing Big Data. In *International Conference on Management of Data (SIGMOD)*, 2015. (cit. on p. 182)
- [RDMRM17] Daniel Ritter, Jonas Dann, Norman May, and Stefanie Rinderle-Ma. Hardware accelerated application integration processing: Industry paper. In *International Conference on Distributed Event-Based Systems (DEBS)*, pages 215–226. ACM, 2017. (cit. on pp. 14, 139, 140, 204, and 224)
- [RFRM19] Daniel Ritter, Fredrik Nordvall Forsberg, Stefanie Rinderle-Ma, and Norman May. Catalog of optimization strategies and realizations for composed integration patterns. *CoRR*, abs/1901.01005, 2019. (cit. on pp. 14, 138, 162, 243, 277, 288, and 327)
- [RFSK10] Tilmann Rabl, Michael Frank, Hatem Mousselly Sergieh, and Harald Kosch. A data generator for cloud-scale benchmarking. In *TPC Technology Conference (TPCTC)*, 2010. (cit. on p. 198)
- [RH15] Daniel Ritter and Manuel Holzleitner. Integration adapter modeling. In *International Conference on Advanced Information Systems Engineering (CAiSE)*, pages 468–482. Springer, 2015. (cit. on pp. 7, 13, 15, 21, 22, 24, 25, 40, 41, 44, 49, 50, 53, 65, 181, 188, 196, and 198)
- [RH18] Daniel Ritter and Manuel Holzleitner. Toward resilient mobile integration processes. In *International Conference on Business Information Systems (BIS)*, pages 278–291, 2018. (cit. on pp. 1, 62, 278, and 287)
- [Rit14a] Daniel Ritter. Experiences with business process model and notation for modeling integration patterns. In *European Conference on Modelling Foundations and Applications (ECMFA)*, pages 254–266. Springer, 2014. (cit. on pp. 15, 24, 25, 31, 35, 37, and 50)
- [Rit14b] Daniel Ritter. Using the business process model and notation for modeling enterprise integration patterns. *CoRR*, abs/1403.4053, 2014. (cit. on pp. 15 and 24)
- [Rit14c] Daniel Ritter. What about database-centric enterprise application integration? In *Central-European Workshop on Services and their Composition (ZEUS)*, pages 73–76, 2014. (cit. on pp. 205, 206, 211, and 269)
- [Rit15a] Daniel Ritter. Compilation of BPMN-based integration flows. In *Central-European Workshop on Services and their Composition (ZEUS)*, pages 1–9, 2015. (cit. on pp. 163 and 276)
- [Rit15b] Daniel Ritter. Towards more data-aware application integration. In *British International Conference on Databases (BICOD)*, pages 16–28. Springer, 2015. (cit. on pp. 9, 14, 43, 140, 168, 170, 172, 173, 184, 191, 192, 211, and 284)
- [Rit15c] Daniel Ritter. Towards more data-aware application integration (extended version). *CoRR*, abs/1504.05707, 2015. (cit. on pp. 31, 35, and 37)
- [Rit17a] Daniel Ritter. Database processes for application integration. In *British International Conference on Databases (BICOD)*, pages 49–61. Springer, 2017. (cit. on pp. 14, 67, 139, 204, and 206)

- [Rit17b] Daniel Ritter. Hardware accelerated application integration: challenges and opportunities. In *International Workshop on Active Middleware on Modern Hardware, ACTIVE@Middleware*, page 15, 2017. (cit. on pp. 14, 201, 204, 222, 224, 278, and 279)
- [RJ13] Tilmann Rabl and Hans-Arno Jacobsen. Big Data Generation. In *Workshop on Big Data Benchmarking (WBDB)*, 2013. (cit. on p. 182)
- [RMB02] William A Ruh, Francis X Maginnis, and William J Brown. *Enterprise application integration: a Wiley tech brief*. John Wiley & Sons, 2002. (cit. on pp. 2, 6, and 22)
- [RMC12] Mariano Rodriguez-Muro and Diego Calvanese. High performance query answering over DL-lite ontologies. In *International Conference on Principles of Knowledge Representation and Reasoning (KR)*, 2012. (cit. on p. 277)
- [RMFR18] Daniel Ritter, Norman May, Fredrik Nordvall Forsberg, and Stefanie Rinderle-Ma. Optimization strategies for integration pattern compositions. In *International Conference on Distributed Event-Based Systems (DEBS)*, pages 88–99. ACM, 2018. (cit. on pp. 14, 62, and 138)
- [RMRM17] Daniel Ritter, Norman May, and Stefanie Rinderle-Ma. Patterns for emerging application integration scenarios: A survey. *Information Systems*, 67:36–57, 2017. (cit. on pp. 13, 14, 21, 65, 107, 155, 246, and 268)
- [RMSRM16] Daniel Ritter, Norman May, Kai Sachs, and Stefanie Rinderle-Ma. Benchmarking integration pattern implementations. In *International Conference on Distributed Event-Based Systems (DEBS)*, pages 125–136. ACM, 2016. (cit. on pp. 14 and 170)
- [Ros69] Azriel Rosenfeld. Picture processing by computer. *ACM Computing Surveys (CSUR)*, 1(3):147–176, 1969. (cit. on p. 249)
- [Rot05] J. Roth. *Mobile Computing: Grundlagen, Technik, Konzepte*. Dpunkt Lehrbuch. dpunkt-Verlag, 2005. (cit. on p. 1)
- [RPBL13] Michael Rudolf, Marcus Paradies, Christof Bornhövd, and Wolfgang Lehner. The graph story of the SAP HANA database. In *Datenbanksysteme für Business, Technologie und Web (BTW)*, volume 13, pages 403–420. Citeseer, 2013. (cit. on pp. 204 and 209)
- [RR15] Daniel Ritter and Stefanie Rinderle-Ma. Toward a collection of cloud integration patterns. *CoRR*, abs/1511.09250, 2015. (cit. on pp. 10, 13, 21, 31, 35, 46, 49, 52, 58, 59, and 65)
- [RRM17] Daniel Ritter and Stefanie Rinderle-Ma. Toward application integration with multimedia data. In *IEEE International Enterprise Distributed Object Computing Conference (EDOC)*, pages 103–112. IEEE, 2017. (cit. on pp. 14, 15, 204, and 249)
- [RRM⁺18] Daniel Ritter, Stefanie Rinderle-Ma, Marco Montali, Andrey Rivkin, and Aman Sinha. Catalog of formalized application integration patterns. *CoRR*, abs/1807.03197, 2018. (cit. on pp. 13 and 62)

- [RRMM⁺18] Daniel Ritter, Stefanie Rinderle-Ma, Marco Montali, Andrey Rivkin, and Aman Sinha. Formalizing application integration patterns. In *IEEE International Enterprise Distributed Object Computing Conference (EDOC)*, pages 11–20. IEEE, 2018. (cit. on pp. 13 and 62)
- [RS⁺13] Peter Reimann, Holger Schwarz, et al. Datenmanagementpatterns in Simulationsworkflows. In *Datenbanksysteme für Business, Technologie und Web (BTW)*, pages 279–293, 2013. (cit. on p. 270)
- [RS14] Daniel Ritter and Jan Sosulski. Modeling exception flows in integration systems. In *IEEE International Enterprise Distributed Object Computing Conference (EDOC)*, pages 12–21. IEEE, 2014. (cit. on pp. 13, 15, 21, 24, 44, 46, and 188)
- [RS16] Daniel Ritter and Jan Sosulski. Exception handling in message-based integration systems and modeling using BPMN. *International Journal of Cooperative Information Systems*, 25(2):1–38, 2016. (cit. on pp. 13, 15, 21, 24, 44, 46, 50, 53, 65, 95, and 97)
- [RSD⁺17] Raghu Ramakrishnan, Baskar Sridharan, John R. Douceur, Pavan Kasturi, Balaji Krishnamachari-Sampath, Karthick Krishnamoorthy, Peng Li, Mitica Manu, Spiro Michaylov, Rogério Ramos, Neil Sharman, Zee Xu, Youssef Barakat, Chris Douglas, Richard Draves, Shrikant S. Naidu, Shankar Shastry, Atul Sikaria, Simon Sun, and Ramarathnam Venkatesan. Azure data lake store: A hyperscale distributed file service for big data analytics. In *International Conference on Management of Data (SIGMOD)*, pages 51–63, 2017. (cit. on p. 209)
- [RTHEvdA05] Nick Russell, Arthur HM Ter Hofstede, David Edmond, and Wil MP van der Aalst. Workflow data patterns: Identification, representation and tool support. In *International conference on Conceptual modeling (ER)*, pages 353–368, 2005. (cit. on p. 270)
- [RTKK02] Amaryllis Raouzaïou, Nicolas Tsapatsoulis, Kostas Karpouzis, and Stefanos Kollias. Parameterized facial expression synthesis based on mpeg-4. *EURASIP Journal on Advances in Signal Processing*, 2002(10):521048, 2002. (cit. on p. 251)
- [RVdFE11] Fernando Rosa-Velardo and David de Frutos-Escrig. Decidability and complexity of Petri nets with unordered data. *Theoretical Computer Science*, 412(34):4439–4451, 2011. (cit. on pp. 67 and 77)
- [RW12] Daniel Ritter and Till Westmann. Business network reconstruction using datalog. In *International Workshop on Datalog in Academia and Industry - Datalog 2.0*, pages 148–152, 2012. (cit. on p. 191)
- [SAB⁺15] David Sidler, Gustavo Alonso, Michaela Blott, Kimon Karras, Kees Vissers, and Raymond Carley. Scalable 10gbps TCP/IP stack architecture for reconfigurable hardware. In *International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 36–43. IEEE, 2015. (cit. on p. 222)
- [SAKB10] Kai Sachs, Stefan Appel, Samuel Kounev, and Alejandro Buchmann. Benchmarking publish/subscribe-based messaging systems. In *International Conference on Database Systems for Advanced Applications (DASFAA) Workshops: BenchmarX*, 2010. (cit. on p. 197)

- [San16] Ario Santoso. *Verification of Data-aware Business Processes in the Presence of Ontologies*. PhD thesis, Dresden University of Technology, Germany, 2016. (cit. on p. 277)
- [SAP18a] SAP SE. SAP Cloud Platform Integration – Prepackaged Content. <https://api.sap.com/>, 2018. (cit. on pp. 54, 55, 95, 122, and 171)
- [SAP18b] SAP SE. SAP Financial Services Network. <https://www.sap.com/germany/products/financial-services-network.html>, 2018. (cit. on p. 55)
- [SAP19a] SAP SE. SAP Cloud Platform Integration. <https://api.sap.com/shell/integration>, 2019. (cit. on pp. 2, 38, 39, 47, 48, 90, 134, 155, 163, 168, 178, 180, 187, 191, 246, 247, 255, and 256)
- [SAP19b] SAP SE. SAP Contract Accounts Receivable and Payable (FI-CA). https://help.sap.com/erp2005_ehp_06/helpdata/en/73/834f3e58717937e10000000a114084/frameset.htm, 2019. (cit. on p. 202)
- [SAP19c] SAP SE. SAP Vehicle Insights. <https://www.sap.com/products/vehicle-insights.html>, 2019. (cit. on pp. 202 and 203)
- [Sch94] Doug Schuler. Social computing. *Communications of the ACM*, 37(1):28–29, 1994. (cit. on p. 1)
- [Sch10] Thorsten Scheibler. *Ausführbare Integrationsmuster*. PhD thesis, Universität Stuttgart, 2010. (cit. on pp. 20, 25, and 276)
- [SEG08] R. Seguel, R. Eshuis, and P. Grefen. An overview on protocol adaptors for service component integration. Technical report, Technische Universiteit Eindhoven, 2008. (cit. on pp. 31 and 33)
- [Sel11] Peter Selinger. A survey of graphical languages for monoidal categories. In Bob Coecke, editor, *New Structures for Physics*, volume 813 of *Lecture Notes in Physics*, pages 289–355. Springer, 2011. (cit. on pp. 114 and 121)
- [SG08] Satnam Singh and David J. Greaves. Kiwi: Synthesis of FPGA circuits from parallel programs. In *International Symposium on Field-Programmable Custom Computing Machines*, pages 3–12, 2008. (cit. on p. 222)
- [SGGM⁺10] Laura Sánchez-González, Félix García, Jan Mendling, Francisco Ruiz, and Mario Piattini. Prediction of business process model quality based on structural metrics. In *International conference on Conceptual modeling (ER)*, pages 458–463, 2010. (cit. on pp. 137, 140, and 159)
- [Sha98] Ashok K Sharma. *Programmable Logic Handbook: PLDs, CPLDs, and FPGAs*. McGraw-Hill New York, 1998. (cit. on p. 224)
- [SHT⁺77] Nan C. Shu, Barron C. Housel, Robert W Taylor, Sakti P. Ghosh, and Vincent Y. Lum. Express: a data extraction, processing, and restructuring system. *ACM Transactions on Database Systems (TODS)*, 2(2):134–174, 1977. (cit. on p. 4)
- [Sif80] Joseph Sifakis. Use of Petri nets for performance evaluation. *Acta Cybernetica*, 4(2):185–202, 1980. (cit. on pp. 67 and 130)
- [Sim96] Herbert A Simon. *The sciences of the artificial*. MIT press, 1996. (cit. on p. 11)

- [SJL⁺10] Mohammad Sadoghi, Hans-Arno Jacobsen, Martin Labrecque, Warren Shum, and Harsh Singh. Efficient event processing through reconfigurable hardware for algorithmic trading. *International Conference on Very Large Data Bases (VLDB)*, 3(2):1525–1528, 2010. (cit. on p. 271)
- [SKBB09] Kai Sachs, Samuel Kounev, Jean Bacon, and Alejandro Buchmann. Performance evaluation of message-oriented middleware using the SPECjms2007 benchmark. *Performance Evaluation*, 66(8):410–434, Aug 2009. (cit. on p. 197)
- [SL90] Amit P Sheth and James A Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys (CSUR)*, 22(3):183–236, 1990. (cit. on p. 4)
- [SL05] Thorsten Scheibler and Frank Leymann. Realizing enterprise integration patterns in WebSphere. Technical report, University of Stuttgart, 2005. (cit. on p. 36)
- [SL08] Thorsten Scheibler and Frank Leymann. A framework for executable enterprise application integration patterns. In *Enterprise Interoperability III*, pages 485–497. Springer, 2008. (cit. on p. 36)
- [SMWM06] Utkarsh Srivastava, Kamesh Munagala, Jennifer Widom, and Rajeev Motwani. Query optimization over web services. In *International Conference on Very Large Data Bases (VLDB)*, pages 355–366. VLDB Endowment, 2006. (cit. on p. 3)
- [SO00] Wasim Sadiq and Maria E Orlowska. Analyzing process models using graph reduction techniques. *Information systems*, 25(2):117–134, 2000. (cit. on pp. 131 and 161)
- [Sob10] Paweł Sobociński. Representations of Petri net interactions. In *International Conference on Concurrency Theory*, pages 554–568. Springer, 2010. (cit. on pp. 110 and 112)
- [Sof17] Software AG. Webmethods Integration Cloud. <http://www.softwareag.com/corporate/products/cloud/integration/default.asp>, 2017. (cit. on pp. 2, 39, and 47)
- [Sol16] Solace Solutions. Solace message router. <https://solace.com/>, 2016. (cit. on pp. 222 and 272)
- [SP08] Deven Shah and Dhiren Patel. Dynamic and ubiquitous security architecture for global SOA. In *International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM)*, pages 482–487, 2008. (cit. on pp. 31, 34, 37, 46, and 49)
- [SP15] Leslie F Sikos and David MW Powers. Knowledge-driven video information retrieval with lod: from semi-structured to structured video metadata. In *ESAIR*, pages 35–37, 2015. (cit. on pp. 246 and 259)
- [SP17] Mohammad Sadoghi and Ilia Petrov, editors. *Second International Workshop on Active Middleware on Modern Hardware, ACTIVE@Middleware*. ACM, 2017. (cit. on p. 279)
- [SPE10] SPEC. SPEC SOA benchmark. <https://www.spec.org/soa/>, 2010. (cit. on pp. 169 and 170)

- [SRL10] Thorsten Scheibler, Dieter Roller, and Frank Leymann. From pipes-and-filters to workflows. In *Enterprise Interoperability IV*, pages 255–264. Springer, 2010. (cit. on pp. 4, 132, and 287)
- [SSGH15] Kristoffer Robin Stokke, Håkon Kvale Stensland, Carsten Griwodz, and Pål Halvorsen. Energy efficient continuous multimedia processing using the tegra K1 mobile SoC. In *International Workshop on Mobile Video (MoVid)*, pages 15–16. ACM, 2015. (cit. on p. 248)
- [SSS07] Ian Simon, Noah Snavely, and Steven M Seitz. Scene summarization for online image collections. In *International Conference on Computer Vision (ICCV)*, pages 1–8. IEEE, 2007. (cit. on pp. 246 and 256)
- [Sta14] CACM Staff. Responsible programming not a technical issue. *Communications of the ACM*, 57(10):8–9, 2014. (cit. on pp. 6, 61, 62, 63, and 134)
- [Sto02] Michael Stonebraker. Too much middleware. *International Conference on Management of Data (SIGMOD)*, 31(1):97–106, 2002. (cit. on p. 4)
- [Tai06] Toufik Taibi. Formalising design patterns composition. *IEE Proceedings-Software*, 153(3):127–136, 2006. (cit. on p. 131)
- [TBBG07] Maurice Ter Beek, Antonio Bucchiarone, and Stefania Gnesi. Web service composition approaches: From industrial standards to formal methods. In *International Conference on Internet and Web Applications and Services (ICIW)*, pages 15–15. IEEE, 2007. (cit. on p. 132)
- [Tha06] Markus J Thannhuber. *The intelligent enterprise: theoretical concepts and practical implications*. Springer Science & Business Media, 2006. (cit. on p. 2)
- [Tib17] Tibco. Tibco Cloud Integration Documentation. <https://integration.cloud.tibco.com/docs/index.html>, 2017. (cit. on pp. 2, 39, 47, and 48)
- [Til91] RD Tillett. Image analysis for agricultural processes: a review of potential opportunities. *Journal of agricultural Engineering research*, 50:247–258, 1991. (cit. on pp. 246, 255, and 256)
- [TKBR14] Muhammad Adnan Tariq, Boris Koldehofe, Sukanya Bhowmik, and Kurt Rothermel. PLEROMA: a SDN-based high performance publish/subscribe middleware. In *International Middleware Conference (Middleware)*, pages 217–228, 2014. (cit. on p. 278)
- [TMM⁺16] Christos Tzelepis, Zhigang Ma, Vasileios Mezaris, Bogdan Ionescu, Ioannis Kompatsiaris, Giulia Boato, Nicu Sebe, and Shuicheng Yan. Event-based media processing and analysis: A survey of the literature. *Image and Vision Computing*, 53:3–19, 2016. (cit. on pp. 246, 254, 264, and 273)
- [TN03] Toufik Taibi and David Chek Ling Ngo. Formal specification of design patterns - a balanced approach. *Journal of Object Technology*, 2(4):127–140, 2003. (cit. on p. 131)
- [TPC19] TPC. TPC-H. <http://www.tpc.org/tpch/>, 2019. (cit. on p. 175)
- [Tra17] Tray.io. Tray.io Docs. <http://docs.tray.io/>, 2017. (cit. on pp. 38, 39, 47, and 48)

- [TRH⁺04] David Trowbridge, Ulrich Roxburgh, Gregor Hohpe, Dragos Manolescu, and E.G. Nadhan. *Integration Patterns*. Microsoft Press, 2004. (cit. on p. 25)
- [TS16] Marvin Triebel and Jan Sürmeli. Homogeneous equations of algebraic Petri nets. *arXiv preprint arXiv:1606.05490*, 2016. (cit. on p. 67)
- [TSOB13] Minh Tu Ton That, Salah Sadou, Flavio Oquendo, and Isabelle Borne. Composition-centered architectural pattern description language. In *European Conference on Software Architecture*, pages 1–16. Springer, 2013. (cit. on p. 132)
- [TSS08] Robert Thullner, Alexander Schatten, and Josef Schiefer. Implementing enterprise integration patterns using open source frameworks. *Software Engineering Techniques in Progress*, pages 111–124, 2008. (cit. on p. 36)
- [TULA13] Tanyaporn Tirapat, Orachun Udomkasemsub, Xiaorong Li, and Tiranee Achalakul. Cost optimization for scientific workflow execution on cloud computing. In *International Conference on Parallel and Distributed Systems (ICPADS)*, pages 663–668, 2013. (cit. on p. 139)
- [TW13] Jens Teubner and Louis Woods. *Data Processing on FPGAs*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2013. (cit. on pp. 222, 224, 225, 226, 227, 228, and 271)
- [TYPM09] Hugh Taylor, Angela Yochem, Les Phillips, and Frank Martinez. *Event-driven architecture: How SOA enables the real-time enterprise*. Pearson Education, 2009. (cit. on p. 34)
- [TYRW14] Yaniv Taigman, Ming Yang, Marc’Aurelio Ranzato, and Lior Wolf. DeepFace: Closing the gap to human-level performance in face verification. In *Computer Vision and Pattern Recognition (CVPR)*, pages 1701–1708, 2014. (cit. on pp. 246 and 255)
- [UGMW01] Jeffrey D. Ullman, Hector Garcia-Molina, and Jennifer Widom. *Database Systems: The Complete Book*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 2001. (cit. on p. 206)
- [UP06] Karthikeyan Umapathy and Sandeep Purao. Designing enterprise solutions with web services and integration patterns. In *International Conference on Services Computing (SCC)*, pages 111–118. IEEE, 2006. (cit. on pp. 31, 33, and 38)
- [Vas09] Panos Vassiliadis. A survey of extract–transform–load technology. *International Journal of Data Warehousing and Mining*, 5(3):1–27, 2009. (cit. on p. 4)
- [VC15] S. Vemula and C. Crick. Hadoop image processing framework. In *2015 IEEE International Congress on Big Data*, pages 506–513, June 2015. (cit. on p. 274)
- [vdA93] Wil MP van der Aalst. Interval timed coloured Petri nets and their analysis. In *International Conference on Application and Theory of Petri Nets (PN)*, pages 453–472, 1993. (cit. on pp. 67, 84, and 130)
- [vDA02] Wil MP van Der Aalst. Making work flow: On the application of Petri nets to business process management. In *International Conference on Application and Theory of Petri Nets (PN)*, pages 1–22. Springer, 2002. (cit. on pp. 110, 114, and 132)

- [vDALM⁺10] Wil MP van Der Aalst, Niels Lohmann, Peter Massuthe, Christian Stahl, and Karsten Wolf. Multiparty contracts: Agreeing and implementing interorganizational processes. *The Computer Journal*, 53(1):90–106, 2010. (cit. on pp. [110](#) and [132](#))
- [vdAMSW09] Wil MP van der Aalst, Arjan J Mooij, Christian Stahl, and Karsten Wolf. Service interaction: Patterns, formalization, and analysis. In *International School on Formal Methods for the Design of Computer, Communication and Software Systems*, pages 42–88. Springer, 2009. (cit. on p. [132](#))
- [VDAVHvH04] Wil Van Der Aalst, Kees Max Van Hee, and Kees van Hee. *Workflow management: models, methods, and systems*. MIT press, 2004. (cit. on p. [3](#))
- [vdAW01] Wil MP van der Aalst and Mathias Weske. The P2P approach to interorganizational workflows. In *International Conference on Advanced Information Systems Engineering (CAiSE)*, pages 140–156. Springer, 2001. (cit. on p. [132](#))
- [Ver07] François B Vernadat. Interoperable enterprise systems: Principles, concepts, and methods. *Annual reviews in Control*, 31(1):137–145, 2007. (cit. on p. [34](#))
- [VHSV09] Kees M Van Hee, Natalia Sidorova, and Marc Voorhoeve. Generation of database transactions with Petri nets. *Fundamenta Informaticae*, 93(1-3):171–184, 2009. (cit. on p. [130](#))
- [VJ01] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages I–I. IEEE, 2001. (cit. on p. [250](#))
- [VK15] Vijay K Vaishnavi and William Kuechler. *Design science research methods and patterns: innovating information and communication technology*. CRC Press, 2015. (cit. on pp. [12](#) and [168](#))
- [VLB⁺10] Pranav S. Vaidya, Jaehwan John Lee, Francis Bowen, Yingzi Du, Chandima H. Nadungodage, and Yuni Xia. Symbiote: A reconfigurable logic assisted data stream management system (RLADSMS). In *International Conference on Management of Data (SIGMOD)*, pages 1147–1150. ACM, 2010. (cit. on p. [271](#))
- [VRMB11] Luis M Vaquero, Luis Roderio-Merino, and Rajkumar Buyya. Dynamically scaling applications in the cloud. *ACM SIGCOMM Computer Communication Review*, 41(1):45–52, 2011. (cit. on p. [278](#))
- [VSES08] M. Vrhovnik, O. Suhre, S. Ewen, and H. Schwarz. PGM/F: A framework for the optimization of data processing in business processes. In *International Conference on Data Engineering (ICDE)*, pages 1584–1587, April 2008. (cit. on p. [205](#))
- [VSS⁺07] Marko Vrhovnik, Holger Schwarz, Oliver Suhre, Bernhard Mitschang, Volker Markl, Albert Maier, and Tobias Kraft. An approach to optimize data processing in business processes. In *International Conference on Very Large Data Bases (VLDB)*, pages 615–626, 2007. (cit. on pp. [139](#), [140](#), [205](#), and [270](#))
- [VT17] Paul Vincent and Anne Thomas. Technology insight: The hybrid application platform, 2017. (cit. on pp. [1](#) and [5](#))

- [VTM08] Kostas Vergidis, Ashutosh Tiwari, and Basim Majeed. Business process analysis and optimization: Beyond reengineering. *Transactions on Systems, Man, and Cybernetics (SMC), Part C*, 38(1):69–82, 2008. (cit. on pp. 136 and 139)
- [VVK09] Jussi Vanhatalo, Hagen Völzer, and Jana Koehler. The refined process structure tree. *Data & Knowledge Engineering*, 68(9):793–818, 2009. (cit. on p. 161)
- [W3C01] W3C. Xquery: A query language for XML. <http://www.w3.org/TR/xquery>, 2001. (cit. on p. 253)
- [Wal92] Gregory K Wallace. The JPEG still picture compression standard. *IEEE transactions on consumer electronics*, 38(1):xviii–xxxiv, 1992. (cit. on p. 249)
- [WB97] John A Watlington and V Michael Bove. A system for parallel media processing. *Parallel Computing*, 23(12):1793–1809, 1997. (cit. on pp. 246 and 273)
- [WB02] Bobby Woolf and Kyle Brown. Patterns of system integration with enterprise messaging. In *Conference on Pattern Language of Programs (PLoP)*. Citeseer, 2002. (cit. on pp. 2, 6, 19, 20, 21, and 26)
- [WC96] Jennifer Widom and Stefano Ceri. *Active database systems: Triggers and rules for advanced database processing*. Morgan Kaufmann, 1996. (cit. on pp. 204 and 205)
- [WCZM07] Fei-Yue Wang, Kathleen M Carley, Daniel Zeng, and Wenji Mao. Social computing: From social informatics to social intelligence. *IEEE Intelligent systems*, 22(2):79–83, 2007. (cit. on p. 1)
- [WDOV08] Kenneth Wang, Marlon Dumas, Chun Ouyang, and Julien Vayssière. The service adaptation machine. In *European Conference on Web Services*, pages 145–154, 2008. (cit. on p. 31)
- [WF06] Phillip Ian Wilson and John Fernandez. Facial feature detection using Haar classifiers. *J. Comput. Sci. Coll.*, 21(4):127–133, April 2006. (cit. on p. 264)
- [WF12] Tim Wellhausen and Andreas Fiesser. How to write a pattern?: A rough guide for first-time pattern authors. In *European Conference on Pattern Languages of Programs (EuroPLoP)*, page 5. ACM, 2012. (cit. on p. 45)
- [Whi09] Cynthia Whissell. Using the revised dictionary of affect in language to quantify the emotional undertones of samples of natural language. *Psychological Reports*, 105(2):509–521, 2009. PMID: 19928612. (cit. on p. 252)
- [WHL98] Chris Wright, Hugh Hunston, and Anthony Lewis. *Automotive Logistics: Optimising supply chain efficiency*. FT Automotive, 1998. (cit. on p. 1)
- [Wie14] Roel Wieringa. *Design Science Methodology for Information Systems and Software Engineering*. Springer, 2014. (cit. on p. 11)
- [Wit13] Ludwig Wittgenstein. *Tractatus logico-philosophicus*. Routledge, 2013. (cit. on pp. 19 and 20)
- [WTA10] Louis Woods, Jens Teubner, and Gustavo Alonso. Complex event detection at wire speed with FPGAs. *International Conference on Very Large Data Bases (VLDB)*, 3(1):660–669, 2010. (cit. on pp. 222, 223, 235, and 271)

- [WWES92] Joseph G. Walls, George R. Widmeyer, and Omar A. El Sawy. Building an information system design theory for vigilant EIS. *Information Systems Research*, 3(1):36–59, March 1992. (cit. on p. 12)
- [YPL⁺00] CC Yang, SO Prasher, JA Landry, J Perret, and HS Ramaswamy. Recognition of weeds with image processing and their use with fuzzy logic for precision farming. *Canadian Agricultural Engineering*, 42(4):195–200, 2000. (cit. on pp. 246, 255, and 256)
- [Zal09] J. Zaleski. *Integrating Device Data Into the Electronic Medical Record: A Developer’s Guide to Design and a Practitioner’s Guide to Application*. Wiley, 2009. (cit. on p. 179)
- [Zap17] Zapier. Zapier v2 Documentation. <https://zapier.com/developer/documentation/v2/reference/>, 2017. (cit. on pp. 38, 39, 47, and 48)
- [ZB74] Marvin V Zelkowitz and William G Bail. Optimization of structured programs. *Software: Practice and Experience*, 4(1):51–57, 1974. (cit. on p. 161)
- [ZB10] Hong Zhu and Ian Bayley. Laws of pattern composition. In *International Conference on Formal Engineering Methods*, pages 630–645. Springer, 2010. (cit. on p. 131)
- [ZCF⁺10] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. *HotCloud*, 10(10-10):95, 2010. (cit. on pp. 170, 173, and 204)
- [ZCJ11] Yan Zheng, Hongming Cai, and Lihong Jiang. Application integration patterns based on open resource-based integrated process platform. In *International Conference on Information Computing and Applications*, pages 577–584, 2011. (cit. on p. 33)
- [Zdu07] Uwe Zdun. Systematic pattern selection using pattern language grammars and design space analysis. *Software: Practice and Experience*, 37(9):983–1016, 2007. (cit. on p. 21)
- [Zdu08] Uwe Zdun. Pattern-based design of a service-oriented middleware for remote object federations. *ACM Transactions on Internet Technology (TOIT)*, 8(3):15, 2008. (cit. on pp. 31 and 32)
- [ZDW05] Zlatko Zlatev, Maya Daneva, and Roel Wieringa. Multi-perspective requirements engineering for networked business systems: A framework for pattern composition. In *WER*, pages 26–37, 2005. (cit. on p. 132)
- [Zen85] Alexandre Zenie. Colored stochastic Petri nets. In *International Workshop on Timed Petri Nets*, pages 262–271, 1985. (cit. on pp. 79, 81, and 130)
- [ZH02] Hong Zhu and Xudong He. A methodology of testing high-level Petri nets. *Information and Software Technology*, 44(8):473–489, 2002. (cit. on p. 85)
- [ZHVDA06] Uwe Zdun, Carsten Hentrich, and Wil MP Van Der Aalst. A survey of patterns for service-oriented architectures. *International journal of Internet protocol technology*, 1(3):132–143, 2006. (cit. on pp. 31 and 32)

- [ZHZW12] Peng Zhang, Yanbo Han, Zhuofeng Zhao, and Guiling Wang. Cost optimization of cloud-based data integration system. In *Web Information Systems and Applications Conference (WISA)*, pages 183–188, 2012. (cit. on pp. [139](#), [140](#), and [141](#))
- [ZLM⁺16] Wei Zhang, Guyue Liu, Ali Mohammadkhan, Jinho Hwang, K. K. Ramakrishnan, and Timothy Wood. SDNFV: flexible and dynamic software defined control of an application- and flow-aware data plane. In *International Middleware Conference (Middleware)*, page 2, 2016. (cit. on p. [278](#))
- [ZPHW16] Olaf Zimmermann, Cesare Pautasso, Gregor Hohpe, and Bobby Woolf. A decade of enterprise integration patterns: A conversation with the authors. *IEEE Software*, 33(1):13–19, 2016. (cit. on pp. [2](#), [4](#), [6](#), [9](#), [11](#), [19](#), [20](#), [21](#), [24](#), [26](#), [31](#), [37](#), [38](#), [41](#), [43](#), [44](#), [45](#), [46](#), [49](#), [58](#), [62](#), [272](#), [276](#), and [282](#))
- [ZSSZ14] Hao-Dong Zhu, Zhen Shen, Li Shang, and Xiaoping Zhang. Parallel image texture feature extraction under hadoop cloud platform. In *International Conference on Intelligent Computing (ICIC)*, pages 459–465, 2014. (cit. on p. [274](#))
- [Zub87] Wlodzimierz M. Zuberek. D-timed Petri nets and modeling of timeouts and protocols. *Trans. Soc. Comp. Simul.*, 4(4):331–357, 1987. (cit. on pp. [67](#), [73](#), and [130](#))

Appendix A

Correctness-Preserving Reduce Interaction Optimizations

The catalog of optimization strategies and their optimizations in Chapter 4 evolves with new optimizations added to existing or new strategy categories, collected in [RFRM19]. For example, for a more efficient and resilient processing, pattern compositions strive to reduce interactions with external participants (OS-4). Common optimizations of this strategy are *ignore failing endpoints* and *reduce requests*. To illustrate that our formalism from Chapter 4 is suitable for the definition of new optimizations, the two optimizations are subsequently specified accordingly and their correctness is shown down to the execution semantics. Similarly, other correctness-preserving optimizations on our pattern compositions can be defined.

A.1 Ignore Failing Endpoints

When endpoints fail, different exceptional situations have to be handled on the caller side. In addition, this can come with long timeouts, which can block the caller and increase latency. Knowing that an endpoint is unreliable can speed up processing, by immediately falling back to an alternative.

Change primitives: The rule is given in Figure A.1(a), where SG_{ext} is a failing endpoint, SG_1 and SG_2 subgraphs, and P_1 is a service call or message send pattern with configuration cf . This specifies the collected number of subsequently failed delivery attempts to the endpoint

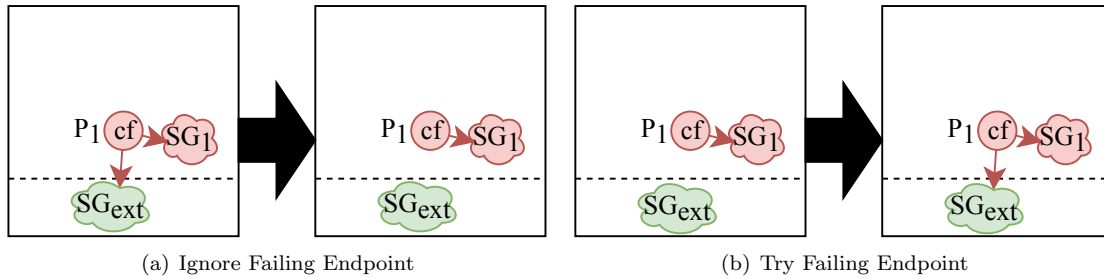


Figure A.1: Rules for ignore failing endpoints

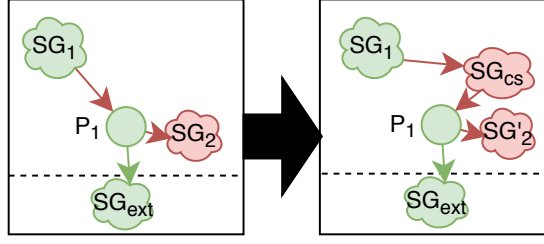


Figure A.2: Rules for reduce requests

or a configurable time interval. If one of these thresholds is reached, the process stops calling SG_{ext} and does not continue with the usual processing in SG_1 , however, invokes an alternative processing or exception handling in SG_2 .

Effect: Besides an improved latency (i.e., average time to response from endpoint in case of failure), the integration process behaves more stable due to immediate alternative processing. To not exclude the remote endpoint forever, the rule in Figure A.1(b) is scheduled for execution after a period of time to try whether the endpoint is still failing. If not, the configuration is updated to cf' to avoid the execution of Figure A.1(a). The retry time is adjusted depending on experienced values (e.g., endpoint is down every two hours for ten minutes).

A.2 Reduce Requests

A *message limited* endpoint, i.e., an endpoint that is not able to handle a high rate of requests, can get unresponsive or fail. To avoid this, the caller can notice this (e.g., by TCP back-pressure) and react by reducing the number or frequency of requests. This can be done by employing a throttling or even sampling patterns from Chapter 2, which reduces the number of messages sent per time or removes messages, respectively. An Aggregator can also help to combine messages to multi-messages Section 6.1.

Change primitives: The rewriting is given by the rule in Figure A.2, where P_1 is a service call or message send pattern, SG_{ext} a message limited external endpoint, SG_2 a subgraph with SG'_2 a re-configured copy of SG_2 (e.g., for vectorized message processing Sections 5.1.2 and 6.1), and SG_{cs} a subgraph that reduce the pace, or number of messages sent.

Effect: Latency and message throughput might improve, but this optimization mainly targets stability of communication. This is improved by configuring the caller to a message rate or number of requests that the receiver can handle.

A.3 Correctness Considerations

We recall the bisimilarity considerations from Section 4.2.4 for the subsequent optimization correctness proofs.

Ignore, try failing endpoints Suppose the left hand side of Figure A.1(a) takes a finite amount of steps to move a token to SG_1 , however, the transition to SG_{ext} does not produce a result due to an exceptional situation (i.e., no change of the marking in cf). Correspondingly, the right hand side moves the token, however, without the failing, and thus read-only transition to SG_{ext} , which ensures the equality of the resulting tokens on either side.

Under the same restriction that the no exception context is returned from SG_{ext} , the right hand side can simulate the left hand side accordingly.

Reduce requests Since the only difference between the left hand side and the right hand side is the slow-down due to the insertion of a throttler pattern SG_{CS} , and simulation does not take the age of messages into account, the left hand side can simulate the right hand side and vice versa.