DISSERTATION / DOCTORAL THESIS

Titel der Dissertation / Title of the Doctoral Thesis

„Design of Metamodels for Domain-Specific Modelling Methods using Conceptual Structures“

verfasst von / submitted by

Mag. Wilfrid Utz

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Doktor der Wirtschaftswissenschaften (Dr.rer.oec.)

Wien, 2020 / Vienna, 2020

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	UA 796 305 175
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Wirtschaftsinformatik
Betreut von / Supervisor:	o. Univ.-Prof. Prof.h.c. Dr. Dimitris Karagiannis

Declaration of Authorship

Abstract

Digital transformation has become the leading topic in recent years to re-invent and re-structure enterprises. The need for transformation is attributed to the availability of novel and innovative technologies, defining the abilities (internal or external) an organisation can build upon in order to elevate/align its offerings, adapt its organisational structure and quickly respond to changing market, legal or technological trends. Agility in these transformation and innovation process is regarded as a key organisational capability. Stakeholders need to trace and understand design decisions taken, evaluate and assess potential business cases based on common artefacts developed and capture the innovation process that has led to its development. Conceptual models and their underlying metamodels are considered the foundational building blocks for these intelligence considerations.

Within this thesis, the conceptual approach and technological realisation to support the harmonisation and alignment of metamodels applied during these transformative processes is discussed. Based on the assumption that stakeholders from different backgrounds require varying domain-specific expressiveness to contribute and collaborate, the need to align and couple these models and provide intelligence functionalities individually or on a global scope is manifested. The contribution targets these needs by defining a formal representation of metamodels using conceptual structures. A graph-based representation is proposed that defines a common, abstract vocabulary for metamodels and an extendible framework for syntactic and semantic knowledge operation to support the

– *Harmonisation of Metamodels* within distributed modelling ecosystems applying similarity matching techniques and to identify virtual relation between the nodes of the environment,

– *Alignment o f Intelligence Capabilities* to dynamically attach model processing functionalities to the participating metamodels and their harmonisation links.

The conceptual approach developed has been evaluated in the context of the OMiLAB environment for completeness and adequacy, accompanied by a prototypical implementation coined DeMoMa for domain-specific metamodel design, harmonisation and alignment of functionality.

Zusammenfassung

Der Begriff "Digitale Transformation" hat sich in den letzten Jahren zum bestimmenden Element in der strategischen Ausrichtung von Unternehmen entwickelt. Technologische, organisatorische und rechtliche Rahmenbedingungen, die von innen und aussen auf ein Unternehmen wirken, fordern eine kontinuierliche Evaluation der strategischen Ausrichtung. Die Art und Weise wie Produkte und Dienstleistungen in einem globalen Umfeld angeboten werden, ist einem konstanten Wandel unterworfen. Agilität, in allen funktionalen Ebenen einer Organisation, hat sich als bestimmende Notwendigkeit entwickelt um eine Neuausrichtung systematisch zu bewerkstelligen. Das Thema "Innovation", als organisatorische Fähigkeit auf Veränderungen vorausschauen zu reagieren bzw. vorwegzunehmen, hast sich von einer exotischen Randerscheinung zu einem bestimmenden Instrument für alle Geschäftsbereiche positioniert. Im Rahmen der hier präsentierten Forschungsarbeit wird auf diese Herausforderung eingegangen: basierend auf der Annahme, dass Innovationsprozesse durch Ansätze der Modellierung unterstützt werden können (im Sinne einer Externalisieren von Wissen), stellt sich diese Arbeit der Frage, wie unterschiedliche Modellierungsansätze, repräsentiert durch ihre Metamodelle, kombiniert und als gemeinsame Wissensbasis für die Innovation verstanden werden können.

Der Beitrag dieser Arbeit baut auf Erkenntnissen der formalen Wissensrepräsentation auf. Als Grundlage werden "Conceptual Structures" herangzogen und für die Entwick von Metamodelle entwickelt. Dieser formale Repräsentation ermöglicht einen systematischen Designprozess mit der Zielsetzung der:

– *Harmonisierung von Metamodellen* um verteilte Modellierungssysteme zu entwickeln, und

– *"Intelligence" Funktionalität*, die auf Basis der Resultate der Harmonisierung, die Zusammenhänge in komplexen Systemen verständlich und nachvollziehbar machen.

Das entwickelte, konzeptionelle Ansatz und deren technologische Realisierung wurde im Rahmen der Ergebnisse des OMiLABs auf Adäquanz und Vollständigkeit evaluiert.

Acknowledgement

This doctoral project has been supervised by o. Univ.-Prof. Prof.h.c. Dr. Dimitris Karagiannis, head of the research group Knowledge Engineering and the Open Models Initiative Laboratory (OMiLAB) at the Faculty of Computer Science, University of Vienna. I would like to thank Prof. Karagiannis for his patience, guidance and willingness to discuss, support and enable the research work throughout the whole period of problem identification, conceptualisation, development and evaluation in an inspiring and motivational manner. Without his ideas, guidance, freedom/liberty and possibilities to "work" and "concentrate", focus and participate in scientific events, projects and conference, this research work and the writing of the thesis would not have been possible.

In addition to the guidance received by my supervisor, I would like to thank the team at the Research Group Knowledge Engineering, my professional environment at BOC and OMiLAB as well as the ADOxx.org community for their support (implicit or explicit) in the development of this research project. Observations, discussions, feedback received and collaborative work performed with Dr. Robert Woitsch and team in Bäckerstraße, Operngasse and now Faulmanngasse have impacted the understanding of the problem space and guided the results established. Insights and viewpoints from the ADOxx.org community, documented in various interactions online and offline, refined and developed the understanding of the necessity of conceptual modelling, the importance of metamodels and digital intelligence thought after in this research project.

Last but not least, I thank my family and friends, who provided the economical and emotional support to follow my academic studies from the early days, already during my master's degree and the years following thereafter. This includes my parents who supported and educated me to follow my own path, to think freely and openly, and my family, that has been understanding and supportive throughout the whole period of research, and especially during the final phase of writing this thesis.

I have no special talents. I am only passionately curious.

Albert Einstein

Contents

List of Figures

List of Tables

List of Definitions

List of Abbreviations

ABL ADOxx Library Language (binary format).

ADL ADOxx Definition Language.

AI Artificial Intelligence.

ALL ADOxx Library Language.

AMME Agile Modelling Method Engineering.

API Application Programming Interfaces.

BG Basic Graph.

BPMN Business Process Model and Notation.

CD Continuous Deployment.

CG Conceptual Graph.

CGDF Conceptual Graph Display Format.

CGIF Conceptual Graph Interchange Format.

CI Continuous Integration.

CoChaCo Concept-Characteristic-Connector.

CPS Cyber-Physical System.

DeMoMa Design-Model-Make.

DeMoMa-(*) Design-Model-Make Analysis Framework.

DSML Domain-Specific Modelling Language.

EAM Enterprise Architecture Management.

EBNF Extended Backus–Naur form.

EG Existential Graphs.

EIS	Executive Information Systems.
FOL	First Order Logic.
GPML	General Purpose Modelling Language.
IBPM	Industrial Business Process Management.
ICT	Information and Communication Technology.
IDE	Integrated Development Environment.
IOT	Internet of Things.
IT	Information Technology.
JSON	JavaScript Object Notation.
MDD	Model-Drive Development.
MEMO	Multi-perspective Enterprise MOdeling.
MFG	Metamodel Foundation Graph.
MIS	Management Information Systems.
MOF	Meta-Object Facility.
OMG	Object Management Group.
OMiLAB	Open Models Initiative Laboratory.
PIM	Platform Independent Model.
PSM	Platform Specific Model.
RDF	Resource Description Framework.
ReST	Representational State Transfer.
SOA	Service-Oriented Architecture.
SuS	System-Under-Study.

UI User Interface.

UML Unified Modelling Language.

XMI Extensible Markup Language (XML) Metadata Interchange.

XML Extensible Markup Language.

YAML YAML Ain't Markup Language.

- This page is intentionally left blank -

Chapter 1

Introduction

Digital transformation initiatives have become a crucial element in the strategy definition of enterprises in almost all industries (Andal-Ancion et al., 2003). In the era of "constant customer connectivity" (Berman, 2012), organisations need to restructure or re-invent themselves constantly, supported by and resulting from the availability of digital technologies. According to (Petry, 2019), an intelligent enterprise needs to focus on three complementary activities:

(a) define *Intelligent Offerings* for the market that create customer and business value as a combination of digital and physical products or services capabilities,

(b) enable *Intelligent Customer Interactions*, considering location and time, collaborative relations with the client, and

(c) establish *Intelligent Processes*, as a transformative activities within the organisation to handle value-adding operations and adapt these model to domain-specific needs.

As stated by (Matt et al., 2015), such activities involve a transformation of "key business operations and affects products and processes, as well as organisational structures and management concepts; digital transformation strategies need to be formulated and aligned with corporate management practices to govern these complex transformations". Consequently, enterprises need to develop a new "portfolio of capabilities for flexibility and responsiveness to fast-changing [customer] requirements" (Berman, 2012) that are defined as organisational *digital innovation* competences.

In the scope of the research work performed, the above mentioned transformative activities and digital innovation capabilities are assessed in the context of information systems analysis and design. Building upon the definition of (Wand et al., 1995, p. 286), information systems are "viewed as a representation, or a model of another system". These representations needs to evolve and consider the impact of digital innovation already on model and metamodel level used. Within the research work performed, temporal and evolutionary aspects of these models and underlying metamodels are considered. Organisational aspects define the way knowledge is represented and to what extend a combination of structures is applicable for specific cases.

The contribution resulting from this research work is within the field of design, conceptualising a metamodelling environment to support temporal and organisational dynamics. The environment is capable to support a *federated design and application of metamodels* building on conceptual structures as a formal knowledge representation. Global functionalities that span across the boundaries of a specific metamodel become feasible using abstract patterns of metamodel structures. As a contribution to the field of digital innovation, a layered approach has been developed based using the categorisation of modelling techniques introduced by Walch in (Walch and Karagiannis, 2019, p. 7244). These layer are applicable as a means to define abstraction/decomposition of artefacts.

In the following section the motivation, as introduced above, is detailed. Observations on current trends and developments are discussed and their impact on the research scope is clarified.

1.1 Motivation

Enterprise operate today in fast changing environments. External factors influence and disrupt business models and objectives of enterprises. As technology advances at a rapid pace and changes the way organisations offer their products and services, legal/regulatory requirements need to be reviewed continuously and respected (Utz, 2018b, p. 47), agility already during design is a necessity. The environmental agility stems from trends how technological developments impact society in general, and enterprises and organisations in a narrow sense. Observations are introduced in the following section, motivating the research on the conceptual modelling framework for digital innovation.

1.2 Observations

Digital Economy and Innovation. At this stage, it is not necessary to question whether digital transformation will impact us or not, as developments within this field have significantly changed our life and the environment surrounding us. Statistical reports show that the economical footprint of the digital economy is increasing year by year and according to the World Economic Forum "by 2022, over 60% of global GDP will be digitized" (World Economic Forum, 2019). This development is attributed to the fact, that digital transformation can not be assigned to a specific sector but impacts all industries in developed and developing countries alike (UNCTAD, 2019). Evidence for these statements can be found in statistical reports on economic figures, that show the current state of the transformation: according to a study by PriceWaterhouseCoopers in (PriceWaterhouseCooper, 2019, p. 23), seven of the ten most valuable companies are directly related to the digital sector. In August 2019, Apple was the first company to reach a market capitalisation of 1 trillion US dollar (Davies, 2018).

Considering the societal dimension, the OECD report on digital economy argues a potential that "digital technologies can democratise innovation" (OECD, 2019, p. 15). Their assessment is based on the consideration, that the reduced cost of information and communication technologies now enable small and credit-constrained firms to develop innovative and novel business models, products, services and processes employing technology.

Observation 1
Digital innovation transforms whole economic sectors and disrupts the way enterprise operate, independent of industrial domain, maturity, age, or origin. Information systems analysis and design methodologies are a relevant and adequate technique to organise transformative change systematically, but need to consider the characteristics of convergence and generativity (Yoo, Boland et al., 2012) as innovation principles.

Digital innovation, its relation to digital transformation and characteristics are discussed from an information systems perspective in (Ciriello et al., 2018). The authors define the term as "innovating products, processes, or business models using digital technology platforms as a means or end within and across organisations". The definition shows the relevance for the field of information systems analysis and design. The characteristic of convergence is directly related to the use of pervasive digital technologies. As product and service artefacts are digitised, they can be combined and reprogrammed flexibly, building on homogenisation of data and self-referential nature (Yoo, Henfridsson et al., 2010, p. 4). New and intelligent product/service offerings can be realised without any limiting factors related to location and time. Generativity as a complementary characteristic "points to the fact that digital technologies are inherently dynamic, extensible, and malleable" (Ciriello et al., 2018, p. 564).

Disruption and Agility. Disruptive innovation, as a process,plays an important role in today's digital business model transformations. In contrast to continuous improvement or sustaining innovation strategies, that perform small, incremental changes in an environment that is well understood, disruptive innovation takes a radical step in an effort to promote change and develop market opportunities/organisational capabilities unknown before (Haemisch, 2013).

The term has been defined initially in (Bower and C. M. Christensen, 1995) as "disruptive technologies" and is closely related to the dynamics of new market entrants and their capabilities to adapt and respond to customer requirements at a faster pace than large cooperations, even though they might lack the organisational capabilities or resources to compete in a classical model.

Observation 2
In the context of information systems analysis and design methodologies, disruptive innovation impacts the way modelling techniques are employed in design phases. To design business models that are radically different, a design methodology is required that supports flexibility on a high abstraction level to a) involve stakeholders from any background, b) express non-conformant ways an enterprise could operate, and c) can quickly provide analytical feedback.

Transforming an existing organisation by applying disruptive innovation strategies requires a radical organisational separation as introduced in

(Clayton M. Christensen et al., 2016). This separation is required to develop novel business models, unbiased from existing sustainable innovation processes that follow a systematic approach.

Exploration not Exploitation. Innovation practices are typically run in a multidisciplinary setting that involve stakeholders from different domains and backgrounds. Exploitation considerations to incrementally improve existing systems and operations are replaced by exploration approaches.

Observation 3
Exploration and experimentation establish the feasibility layer of an information systems design that supports and enables digital innovation. The feasibility layer provides the technical environment and Cyber-Physical System (CPS)s and its devices to evaluate whether a design is effective from an operational perspective. For this purpose it is required to establish abstraction of technical capabilities as a creativity-supporting mitigation principle between the layers defined in (Walch, 2019).

Exploration encompasses the assessment of new design results in the context of the innovation processes, looking at feasibility from a business, organisational and technological standpoint. This tension between exploration and exploitation as a phenomena is discussed in (Ciriello et al., 2018, p. 567).

Conceptual Model Awareness at Run-Time. The role of conceptual modelling has shifted from a documentation and communication purpose, as discussed in (Mylopoulos, 1992, p. 2) to enable "human communication and understanding", towards machine-interpretable knowledge representations.

Observation 4
Metamodels can enable run-time awareness of conceptual models as they make sure that sufficient (machine-readable) richness and granularity in their knowledge representation is exposed to adapt and perform the behaviour accordingly (Karagiannis, 2015).

This observation is in close relation with the view on exploration and feasibility assessment, as the run-time system interprets the knowledge representation embodied in the conceptual models and adapts the behaviour accordingly. This observation needs to be reflected when designing underlying metamodels and their capabilities (structure/behaviour).

Sharing and Re-use of Concepts. Re-use is a prominent concept in software engineering. Software components are not developed from scratch anymore but composed using openly available subcomponents and libraries, resulting in an increased productivity and (debatable) software quality level. Software reuse patterns have been employed since the beginning of programming. (Frakes and Kang, 2005) discusses reuse patterns and architecture to enable the characteristics of collaboration, sharing, software reliability, maintainability and support of domain specific analysis. The open source community has been built upon this characteristics as a movement that gained importance since the end of the 1990s. On GitHub, the self-proclaimed "leading software development platform", 40 million users have been active, resulting in 44 million repositories available publicly (GitHub, 2019).

In the field of conceptual modelling, the re-use pattern has evolved historically from sharing of content as reference models (Koch et al., 2006), collaboration processes in conceptual modelling (Vom Brocke, 2004) towards the open development of modelling methods, metamodels, model processing techniques and their operationalisation (Götzinger et al., 2016).

Observation 5
The re-use pattern is a relevant concept in the design of metamodels that are adequate for a specific domain purpose. Metamodel structures, processing techniques and procedures are considered open artefacts that can be composed dynamically and specialised to the design and analysis requirements of enterprises. As a collaborative effort, conceptual modelling needs to provide means to overcome limitations in semantic distance and provide re-use on an abstract level.

The above observation have contributed the formulation of the problem statement and establish initial requirements for the conceptualisation of the envisioned metamodelling environment. The observations establish the broad characteristics as:

- *Relevance*: the observation on digital innovation processes has reconfirmed the research direction and its relevance. The domain-specific requirements on information systems design need to be reflected in the metamodelling system as an outcome.
- *Responsiveness and Agility*: derived from the observation on disruptive innovation processes, the responsiveness to change of the envisioned environment is crucial. Agile method engineering processes encompass that a) the conceptual modelling approach is dynamic in a sense that it adapt, resulting in b) that the validity of results is time-restricted and only applicable for the specific project/phase or situation. Situational method engineering approaches and mechanisms introduced in (Becker et al., 2007) influence the conceptualisation. Re-use patterns in metamodel engineering support on one hand the collaborative processes and enable the specialisation of pre-exiting artefacts for domain-specific needs.
- *Global and Local Model Processing*: the involvement of different stakeholders is important. They require specialised representation formats and collaboration processes on their specific level of abstraction of the domain, nevertheless a harmonisation needs to be support to realise global and integrated capabilities, vertically across and horizontally among the artefacts created.
- *Exploration of Feasibility*: early testing and evaluation of results needs to be supported. Such feasibility assessment contribute to the explorative nature of innovation processes. It is not sufficient to assess a system after it has put in operation, but different scenarios need to be explored during design and prototypical operation to find innovative solutions for a problem unknown in advance.
- *Visualisation and Software Analysis of Metamodel Interactions*: visualisation of the behaviour of run-time aware conceptual models provides an analytical input for the system to be implemented and deployed. Interactions between heterogenous metamodels need to become transparent to understand current implicit (e.g. correlation between a business model representation and the operational process) links between those models.

In the following section, the problem statement is described, deriving the research gap from the observations discussed above. A motivational case acts as the guiding example for the problem definition.

1.3 Problem Statement

Information systems design and analysis methodologies are characterised by its knowledge intense nature: analysts review, assess and analyse enterprise following defined perspectives, designers build on this analysis to create novel solutions and evaluate the impact of change required. Implementation experts transform designs into operation on an organisational or technological level. The foundation for these tasks relates to the available, externalised knowledge of the System-Under-Study (SuS). The quality of a new design is directly correlated to the availability of these knowledge artefacts, expressiveness and interrelations. Conceptual modelling is an accepted approach in literature to perform the externalisation task.

Current practices in the field show, that integration efforts are an essential component, already discussed in (Olle et al., 1988, p. 36). Submodels created during the planning, analysis and design phases may be fully or loosely integrated with each other to impact an efficient and holistic information systems management strategy. Such integration pose a critical challenge for the evolution of information systems management environment as, in times of digital innovation transformation, they are exposed to uncertainty with respect to the SuS, the stakeholder's expertise and involvement as well as combinational complexity of arbitrary modelling techniques selected and used.

Fig. 1.1 shows graphically the activities, roles and artefacts involved in information systems analysis/design, highlighting the research challenges on three distinct dimensions, introduced below.

1. *Adequate Metamodels*: the challenge relates to the identification or design process of adequate metamodels for a specific task. As shown in Fig. 1.1, this challenge is regarded a cross-layered issue independent of abstraction considerations. The gap in research relates to the availability/accessibility of repositories and knowledge representation formats to identify and match requirements with available results and support a design process of a domain-specific realisation/instantiations.

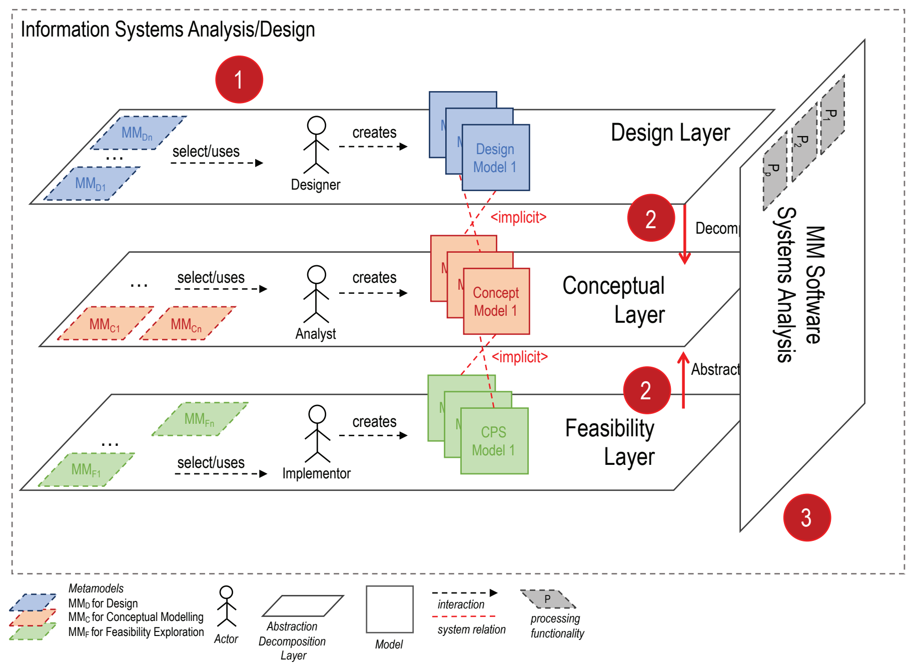


Figure 1.1: Problem Overview: Information Systems Modelling (inspired by (Olle et al., 1988, p. 36)

2. *Interaction within heterogenous metamodel environments*: differing abstraction requirements result in a set of metamodels that are, without considering their cross-layer dependencies, relevant for the design and analysis. A combination is needed as dependencies need to be resolved to provide interrelated structures that can be assessed and queried. Following the work of (Walch and Karagiannis, 2019), a layered approach is introduced that distinguishes design, conceptual and feasibility level interactions. Integration and combinations of metamodels (full or loosely) has been considered following the discussion in (Olle et al., 1988). In light of a responsive adaptation and agility as a novel concept is required that supports federated functionalities in heterogenous settings.

3. *Metamodel Software System Analysis*: the analysis of models at runtime and their interrelation is considered a challenge in current information system practices. Technology to visualise data interchange streams, transformations, Application Programming Interfaces (API) calls or model evolution are only available implicitly through the embedded knowledge of the model and metamodel engineer.

A detailed discussion of these research challenges is provided in the following subsections (considering the relevant background and identified research gap) as input for the definition of research objectives.

1.3.1 Challenge 1: Adequate Metamodels

Background. As metamodelling concepts and technologies have evolved to an established approach to realise domain-specific modelling methods, a plethora of approaches, frameworks and implementations exist for various application and industrial fields. (Karagiannis, Mayr et al., 2016) present the research results in this field within a global community. The publication introduces 24 modelling methods that are individually applicable for a specific purpose, reflect the requirements of stakeholders and provide the needed expressiveness on a certain, well-defined abstraction level. Modelling methods and underlying metamodels cover aspects such as design thinking, security, creative service design, production process planning, enterprise modelling, and many more.

Broadening the scope of the challenge, from Domain-Specific Modelling Language (DSML) towards General Purpose Modelling Language (GPML), industrial initiatives can be identified as relevant. These initiatives follow a standardisation and unification approach. The most prominent example is Unified Modelling Language (UML) that aims to be recognised as a language that "provide system architects, software engineers, and software developers with tools for analysis, design, and implementation of software-based systems as well as for modelling business and similar processes" (Object Management Group (OMG), 2011b). The example of UML shows the evolution of a dedicated domain-environment for software engineers to a unifying approach for information systems design. In the current version 2.5, 14 diagram types are provided, categorised in a structural and behavioural views of a software-based system. The same logic can be observed in the field of business process modelling: Business Process Model and Notation (BPMN) provides not only diagrammatic support for processes, but nowadays integrates operational aspects for workflow design, choreography and orchestration.

Gap: Metamodel Selection. Considering the role of a designer, analyst or implementation expert of an information system, in an initial phase (and then re-iterating in an agile manner), adequate metamodels have to be selected to from available sources. The selection criteria should relate to the expressiveness for the given domain of the SuS and how it supports the purpose and goal defined by involved stakeholders. In contrast to quality consideration as discussed in (Ma et al., 2013; Lopez-Fernandez et al., 2014), that consider metamodels from the software engineering domain, a semantic matching approach for domain requirements to capabilities of a metamodel is currently not available.

For the example shown in Fig. 1.2, it is assumed that the modeller (as a designer, analyst or system implementation expert) identifies the requirements of the SuS from a semantic, application and technology perspective. In the motivation example provided, the domain is "Telecommunication", the application is "Process Architecture Design and Optimization" and as functionalities "Simulation" techniques are to be used. These criteria define the search and matching strategy for adequate metamodels. Candidate metamodels are retrieved and evaluated. Based on the selection, it is applied on the SuS. In case specific requirements can not be satisfied, they are realised explicitly through customisation (adaptation of the metamodel) or implicitly through model fitting (semantic alignment on interpretation level).

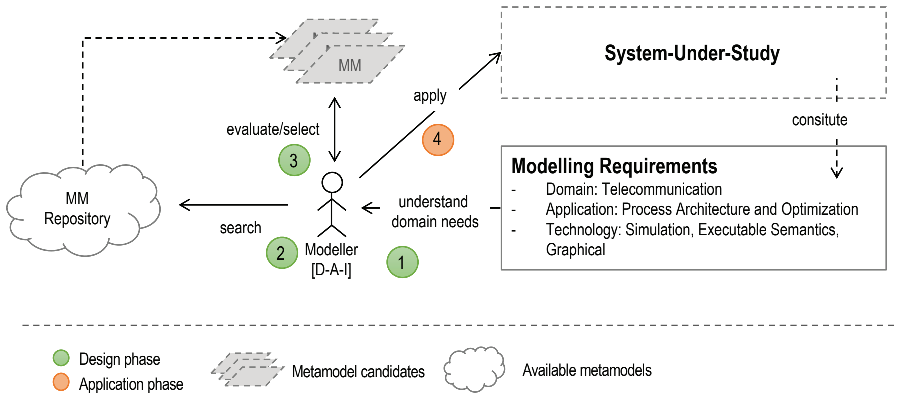


Figure 1.2: Challenge: Metamodel Selection

As we can see from the example in Fig. 1.2, this selection process is a knowledge intense task. The modeller needs to be aware of available results (sources), have the capability to understand the impact of a decision and needs to align with the domain in focus. As an analogy to open source software development processes, semantically rich repositories are required that can be queried using similarity matching mechanisms to recommend adequate metamodels or metamodel fragments/patterns.

Gap: Metamodel Design. The design process of metamodels, as the core element of a modelling method, is considered a challenging issue: the task to derive a consistent metamodel that is adequate (see above) for a specific domain and application, enables model value and provides interpretation support for stakeholders involved, requires conceptual, technological and domain-specific knowledge as a foundation for design decisions (Utz, 2018b).

Design patterns influence the process (e.g. design from scratch, semantic elevation and appropriation, alignment, composition). At its core, the design of a novel metamodel is considered a *knowledge challenge*. Results achieved strongly depend on the experience and knowledge of the metamodel engineer. Reviewing the current practices in the field of metamodel design and engineering, the following phases and steps can be observed, graphically shown in Fig. 1.3. In the discussion a focus is set on design activities.

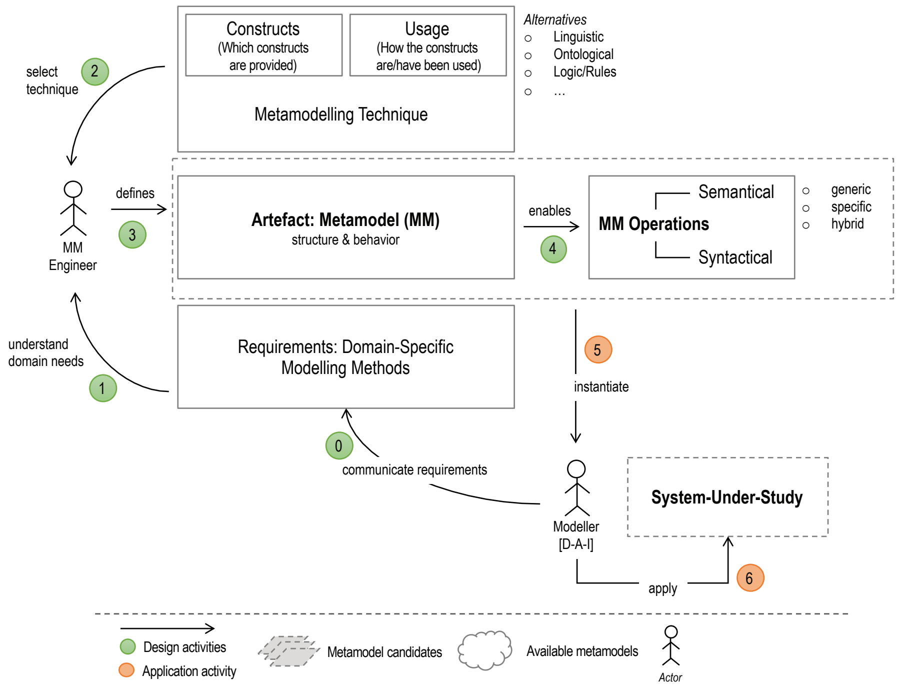


Figure 1.3: Challenge: Metamodel Design/Creation

1. *Understand requirements:* similar to the selection process, in order to develop an adequate and useful metamodel, the requirements of the domain, application and technology have to be captured and understood. Knowledge on the application and industrial domain is required to

trigger the design process, e.g. simulation of manufacturing processes, dependency analysis of community networks, verification of deadlocks in automated workflows, integration of metamodel on different formalisation levels into a consistent state for enterprise architecture management in the domain of trabsportation.

2. *Select a meta-modelling technique:* based on these requirements, the appropriate meta-modelling technique has to be selected. The technique provides generic constructs to develop a metamodel. Construction principle and usage patterns define functionalities e.g. code generation, reasoning techniques on ontological metamodels, deduction in logic based environments. The functionalities offered by the technique are mapped to the requirement of the domain. Knowledge on metamodelling techniques applicable is needed.

3. *Design the metamodel:* having accomplished these 2 prerequisites, the metamodel is designed in conformance with the selected metamodelling approach. The design techniques are typically adjustment, mapping or re-use patterns. The resulting metamodel is strongly influenced by the expertise and creativity of the metamodel engineer.

4. *Enable metamodel operations:* knowledge operations realise the value of models supported/enabled by the metamodel. These operations are functionalities like composition (e.g. in case of sliced metamodel as discussed in (Bork, Karagiannis et al., 2018)), binding of model processing algorithms through reference alignment, transformation and semantic lifting (Hrgovcic, Karagiannis et al., 2013) and operate on the structure and semantics provided by the metamodel. This steps verifies the applicability of the metamodel.

The knowledge-based challenges introduced above, strongly influences the efficiency during the design of an adequate metamodel. The selection of a meta-modelling technique is currently driven by the metamodel engineers knowledge on a specific approach rather than the requirements elicited; during this phase adjustments are performed to overcome potential mis-selection as workarounds. It is assumed that these adjustment have an impact on the usefulness of the resulting metamodel as well as its efficiency. As a description framework of specific, existing metamodels or its abstract fragments is currently not available, re-use of results is limited. As a consequence, it results in a re-implementation of similar structures (using the same or a different technique), demonstrated below for the implementation of the BPMN 2.0 metamodel, graphically shown in Fig. 1.4 as an example. The metamodel has been selected as it is extensively discussed in literature in the past years and demonstrates how different metamodelling techniques impact conceptualisation results.

The representation shows the aspect of "sequence flows" (highlighted in red, dashed line) of the BPMN metamodel and how it has been designed using three different meta-modelling techniques: a) the formal specification from the BPMN 2.0 standardisation deliverable using class diagrams (Object Management Group (OMG), 2011a, p. 144), b) an ontology-based approach to conceptualise the specification (Rospocher et al., 2014, p. 140) and c) a logic-based representation using rules mapped to petri net constructs (introduced in (Jeusfeld, 2016, p. 51) extended in (Jeusfeld, 2017)).

1. *Metamodel using UML Class Diagrams*: intends a formal representation that can be used to generate code as it can be mapped to elements of object-oriented programming languages. This is in line with the BPMN 2.0 specification, that defines that "execution semantics have been fully formalised" (Object Management Group (OMG), 2011a, p. 10) and allow conformance verification as well as runtime interpretation.

2. *Metamodel using OWL Representation*: using ontology concepts enables verification and checks of model artefacts e.g. by "checking the compliance of a process against the BPMN specification" (Rospocher et al., 2014, p. 136), reasoning and detection mechanisms. The sequence logic is implemented as object properties in the ontology.

3. *Metamodel using TELOS*: using logic and rules that are bound to the petri net meta-structure provide simulation capabilities as the behaviour is described already in the abstract meta-model. The sequencing possibilities are represented as a self-referential loop on the root element.

The selected meta-modelling technique does not only impact the way concepts are represented, but is also related to the functionality required and supported by the approach. Currently the intrinsic capabilities of multiple

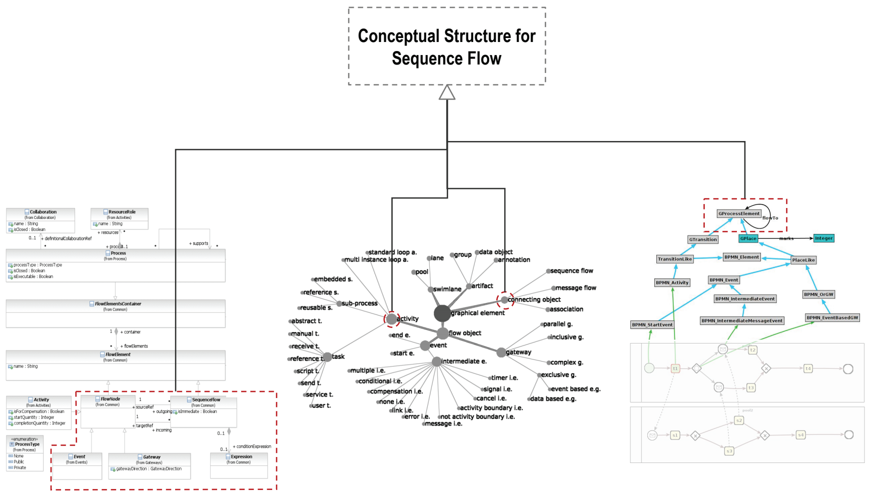


Figure 1.4: Motivational Example: BPMN 2.0 Process Metamodel represented as a) UML 2.0 Class Diagram, b) OWL Ontology, c) Logic/Rules using TELOS(Mylopoulos et al., 1990)

metamodelling techniques are not externalised in a way that non-expert users can identify which approach to choose for a given challenge. It is therefore required to clearly align the knowledge representation on metamodel level with the metamodelling technique that was used to realise them.

1.3.2 Challenge 2: Heterogenous Modelling Environments

Background. The combination of models between different abstraction is an essential element in a comprehensive analysis of an information systems. The elements are related to operational and organisation aspects, and further decomposed on the feasibility layer, that connects with APIs exposed by physical elements. Such an approach manifests a top-down methodology where design influences and controls operation. A reverse viewpoint is also applicable where physical/device capabilities are abstracted from the feasibility layer and the conceptual modelling layer as shown in Fig. 1.1 acts as a mitigation layer between the execution semantics and the abstract design artefacts. Considering multiple instantiations per layer, the complexity of combination is further exaggerated as each composition or integration (independent of the strategy) needs to cope with an any-to-any interfacing issue.

Gap. Current approaches for the operationalisation of heterogenous metamodel environment follow either a) an integration and composition technique (combining all metamodels into a common superstructure) as discussed in (Živkovic and Karagiannis, 2015), b) a semantic lifting approach of relevant fragments (Hrgovcic, Karagiannis et al., 2013), c) model transformation techniques prominently discussed for the software engineering domain in (Mens and Van Gorp, 2006; Czarnecki and Helsen, 2003) (specifically focusing on semantic loss) or d) a combination of the previously mentioned techniques, classified as a hybrid integration. A common denominator for all techniques is that the control of interaction is assigned to a specific element within the system that binds, synchronises, retrieves or calls functionalities of the opponent (e.g. in case of transformation from a source to a target, for communication, push and pull mechanisms).

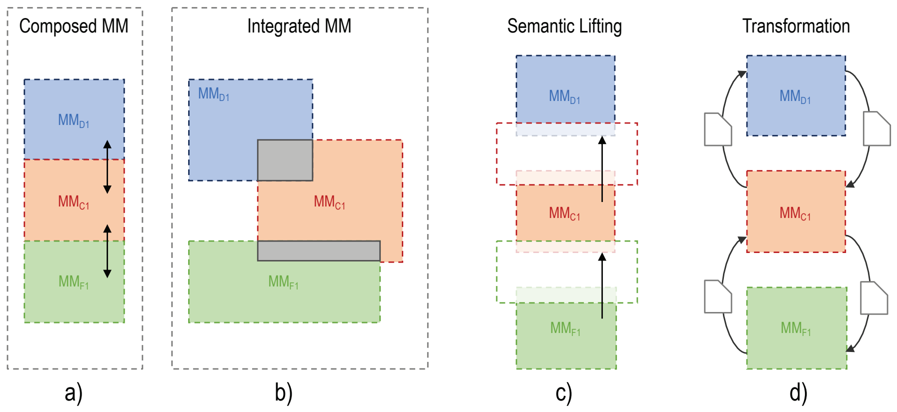


Figure 1.5: Challenge: Interaction in Heterogenous Metamodel Environments

Fig. 1.5 shows abstract examples of the combination approaches reviewed, considering the layers introduced earlier: a) a composed metamodel is constructed in a modular fashion. Pointing concepts establish the interrelations between the modules. The prerequisite for such a modular composition is that the concerned metamodels are semantically close to each other; b) integration of metamodels where overlaps and extensions act as the binding parts. For such an integration, available metamodels need to be assessed for overlapping, similar meaning to create strong integration links; c) semantic lifting supports the aforementioned approaches and in addition allows for elevation of domain semantics to create e.g. tunnel/umbrella structures for integration. The main difference to a) and b) is the distributed nature in such a case; d) model transformation (uni-directional or bi-directional) support the transformation of one layer to the other. The semantic distance has to be bridged within the mapping rules and strategies to handle semantic loss need to be employied.

Any of these approaches result in a complex modelling environment that is inefficient with respect to responsiveness to change and therefore lack the required flexibility to exchange elements, extend and modify structures and/or functionality.

1.3.3 Challenge 3: Digital Intelligence

Background. As outlined above, the multi-disciplinary nature of information systems design processes result in a disperse landscape of modelling approaches, techniques, metamodels and tools. Knowledge operations for analytical purposes are well-establish on local instances within a specific layer, but global views across layers and metamodels require complex mapping/translation mechanisms. This observation is specifically valid for operationalisation where various software artefacts are made available, interfacing technologies are used to realise combinations and user interactions become transparent, independent of the underlying complex stack of services and tools.

This hinders the externalisation of knowledge within the composed nature of such systems. (Bihanic and Polacsek, 2012) discuss introduce information systems as "heterogeneous and interrelated subsystems including interactions" and introduce visualisation options as viewpoints that allow for a change in granularity and also a "shift in semantic universe" (Bihanic and Polacsek, 2012, p. 131). In the paper, the authors introduce visualisation options for such systems and discuss three representation patterns: tree-based for hierarchical representations, maps for reticular representation of e.g. networks and landscapes that support "multi-scale representations of planar shapes"; these patterns are deemed relevant for the visualisation of the structural aspects of a complex, software-based system.

(Pacione et al., 2004) discuss the term "software comprehension" and introduce a multi-faceted, three-dimensional abstraction model for the visualisation of software systems, in line with the classification performed in (Petre and Quincey, 2006). The authors argue, that different tasks require information and visualisation within the software development lifecycle, ranging from design, comprehension of large systems, performance tuning or debugging. A relevant consideration in their publication relates to the observation with respect to visualisation. Visualisation target "multiple, linked, dynamic visualisations of the interaction of system components at runtime"(Petre and Quincey, 2006, p. 1) as an elevation of human understanding and effective use of technology.

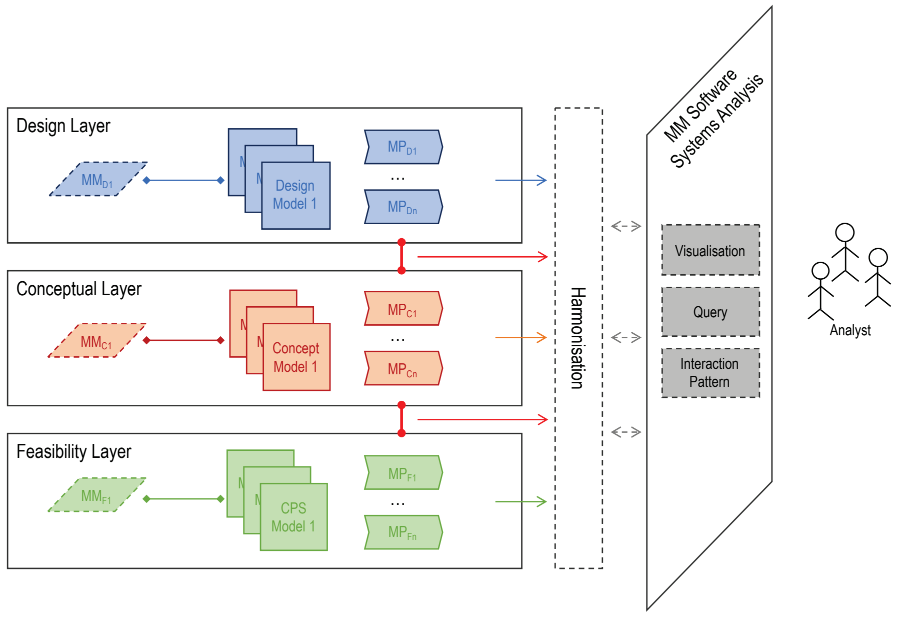


Figure 1.6: Challenge: Metamodel Software System Analysis

Gap. The techniques in software engineering for visualisation and analysis of complex systems have been considered in research but only partially are considered in application. As a scientific challenge, the knowledge to enable analytical viewpoints on a runtime system are still implicit to a large extend i.e. the software engineer is well aware of the interrelation of the system, its dependencies and interaction path but does not externalise these trajectories in an analytical way. This is specifically true for information systems as the SuS is typically represented in a visual manner applying diagrammatic/graphical modelling techniques but the visualisation of interdependencies and runtime characteristics, especially across different formalisms and abstraction layers is limited. The analysis and visualisation needs to support real-time visualisation of interaction streams on metamodel level to follow the responsiveness and intelligence requirements.

The challenges and gaps outlined above contribute to the formulation of the objectives of the research performed and contribution made to the field of conceptual modelling in information systems design, specifically in digital innovation settings. They are summarised as:

– Knowledge externalisation support is required in the field of metamodelling to design and implement adequate modelling environments, whereas the focus of the contribution is on design techniques,

– Homogenisation of interaction on metamodel level is considered as an aspect to support analytical algorithms. For this challenge, the proposed knowledge representation needs to support human and machine interpretation alike,

– Visualisation of information systems at run-time is regarded a building block to enable transparent analytical capabilities and bridge the gap between high level designs and analytical levels.

A flexible toolbox of design techniques, patterns and fragments supports the metamodel engineer to overcome the above challenges in an efficient way. The flexibility characteristic is positioned as a requirement derived from the agile nature of the SuS in times of (digital) transformation. Agile approaches for metamodelling have been initially discussed in (Karagiannis, 2015) as the Agile Modelling Method Engineering (AMME) framework considered as a guiding procedure for the concept development phase of this thesis.

1.4 Research Objective

Based on the challenges introduced in the previous section, the research objectives are outline. In current practices (design, selection and (continuous) adaptation/evolutions of metamodels) are a tedious, inefficient and error-prone task that requires expertise on meta-modelling techniques, domain knowledge, an understanding of existing results and creativity to perform the adjustment and interpretation of requirements to an evolving/changing metamodels stack. To address this gap, it is proposed to extend the notion of metamodels towards conceptual structures as a formal knowledge representation approach and its graph representation to define and describe metamodels independent of the technique used and collect these conceptual structures for re-use in the engineering process of metamodels in an open repository as input for a) the elevation of local modelling method mechanisms and algorithms to a global level and support overarching analytical capabilities (including an assessment of interrelation at runtime), and b) enable the visualisation of these interaction in a common framework.

The need for adequate, useful and purposeful metamodels on different levels and during different phases of digital innovation/transformation influence the research objectives.

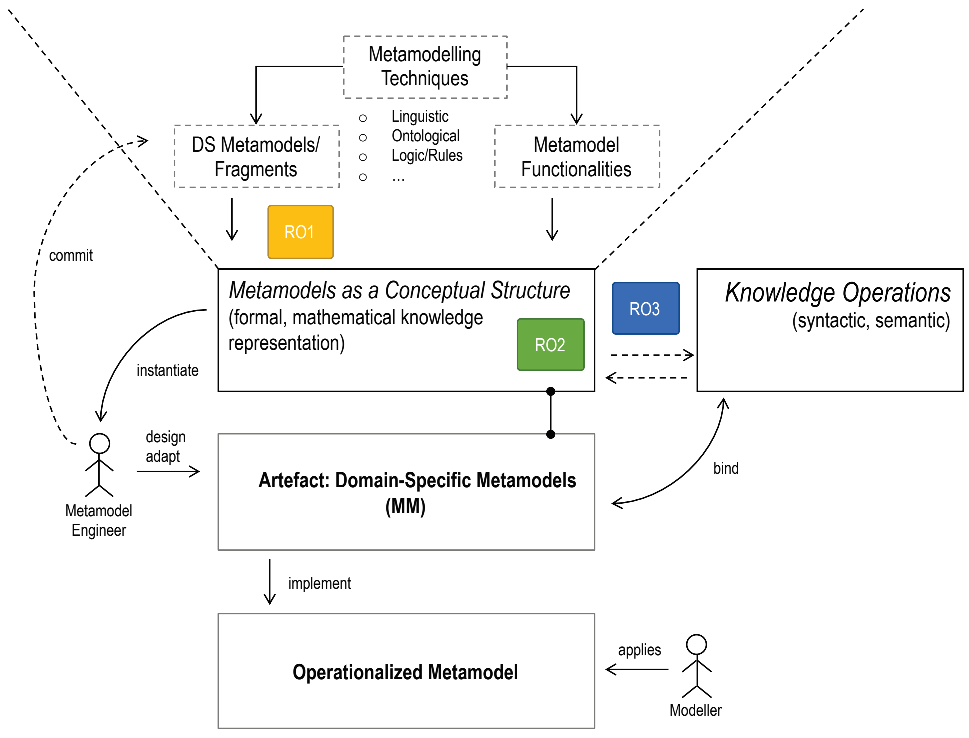


Figure 1.7: Research Objectives:
Metamodels as Conceptual Structures

This section introduces the research objectives based on hypothesis and assumptions (derived from the observations, challenges and gaps) and defines the intended contribution in the field of conceptual modelling.

Fig. 1.7 shows graphically the relation between objectives and the research topic. Following the broad theme of "Metamodels as Conceptual Structures", three research objectives have been formulated that guided the work performed[1] Underlying hypothesis and assumptions for the formulation are introduced.

Hypothesis 1
A metamodel is a knowledge representation of a specific industrial or application domain on type level. A formalisation of this knowledge for machine interpretation supports the operationalisation of the metamodel. Intelligent support mechanisms are required to design or align metamodels to these semantics.

Initial evidence for Hypothesis 1 is available in the engineering approaches for modelling tools as a potential operationalisation of metamodels. For the realisation of tools, a transformation of the conceptual representation of a metamodel into its formal, operationalised format is typically performed by an experienced metamodel engineer.

Software engineering processes support the transformation from a Platform Independent Model (PIM) into a Platform Specific Model (PSM). Based on design decisions a mapping of requirements to platform specific characteristics is performed. As such, the formalisation process and its results are an implicit artefact. Consequently, re-implementations (of the same, or slightly different aspects) happen instead of re-use, as the interaction between stakeholders happen on source code level rather than on a (formal) conceptual representation.

Literature that supports this hypothesis conisders metamodelling as an engineering discipline. Various approaches are introduced in (Bork, Karagiannis et al., 2018); an evaluated technique established in the context of the OMiLAB is AMME, discussed in detail in (Karagiannis, Burzynski et al., 2019), graphically introduced in Fig. 2.4. AMME defines as an engineering lifecycle iterative engineering phases, that enforces formalisation, development and deployment of smart models based on the outcomes of creative phases.

[1]As a design science research approach is followed (introduced in detail in section 1.5), feedback received during the presentation of (Utz, 2018b) has been reflected to formulate research objectives as broad themes rather than concrete research questions.

Hypothesis 2
A single metamodel cannot fulfil the requirements of a multi-layered approach for digital innovation as different foci of stakeholders have to be considered. Dynamics in the domain are evident and as a response, dispersed, heterogenous modelling environments are evolving.

Standardisation efforts follow a unification and continuous extension strategy. This means that various levels of abstraction are integrated, combined and made available. Examples for this observation can be found in e.g. in the field of enterprise architecture modelling that spans multiple dimensions from strategy to infrastructure (Gill, 2015), the usage dimensions of UML as discussed in (Dobing and Parsons, 2011) or scientific results such as MEMO (Frank, 2014; Bock and Frank, 2016).

Hypothesis 3
Harmonisation in a heterogenous information system landscape is a critical factor in in today's digital economy. Concepts to realise such harmonisation attempts are well-established on layers of low semantic expressiveness (e.g. data transformation). The need for feasibility assessment/experimental exploration is required, visualisation of interactions of domain-specific information systems components contribute to this challenge.

For data-level harmonisation approaches, research is well advanced. Concepts have been established e.g. for interoperability of autonomous databases in (Litwin et al., 1990) on syntax and format level, to schema evolution as discussed in (Roussopoulos and Karagiannis, 2009) that introduces the concept of "schema-during" as a bridging element between conceptual modelling and data collection and manipulation.

In the field of model-driven approaches in general, and conceptual modelling in particular, harmonisation techniques within distributed, heterogenous systems are addressed by integration, transformation or composition techniques, mainly concerned with interoperability aspects. (Woitsch, 2013) discusses this aspect by reviewing metamodel characteristics from (a) a domain perspective, (b) the level of technical granularity, (c) the degree of formalisation and finally (d) the cultural dependency of the applying community. A review of potential patterns for merging metamodels are introduced: (a) loose integration of meta models, (b) strong integration and (c) hybrid integration based on (Kühn, 2004). The argumentation presented as "loose coupling is flexible, whereas tight coupling enables the realisation of additional functionality" (Woitsch, 2013, p. 300) contributes to the research objectives of this thesis.

Based in the hypothesis above, the research question has been defined.

Which form of knowledge representation is appropriate and can be identified to support the analysis of information systems defined using metamodels?

The breakdown of the question, in the context of metamodel design and operationalisation of heterogenous modelling environments, is identified as

– Which formalisation technique is appropriate for metamodels pertaining different abstraction levels? Which patterns can be imposed on existing ones and support the design of new metamodel or their evolution?

  Research themes: integration of metamodels through alignment of metamodel capabilities, re-use of metamodel fragments and their adaptation, input for "from-scratch" developments, design patterns

– To what extend is the operationalisation supported in the sense of model value processing algorithms and mechanisms?

  Research themes: operationalisation and usage, instantiation of the AMME framework for re-use on implementation level, requirements from the analysis viewpoint, homogenisation strategies and operationalisation patterns

– Which analysis techniques are required to assess interrelation on software level and the feasibility on software/tool level?

  Research themes: visualisation techniques for conceptual modelling environment, software-based analysis (exploration during run-time)

As an outcome of the research work, a conceptual design environment is proposed as a contribution to the field of domain-specific modelling method engineering, more specifically to the design of metamodels. The notion of metamodel is extended and elevated towards conceptual structures. The environment support the method engineering in the harmonisation of heterogenous modelling systmes and its operationalisation, embedded in a defined, agile development approach based on the AMME framework. The characteristics and advantages of the design framework build on the assumptions that an increased productivity and efficiency can be realised as a high expressiveness and narrowed down design with less model fitting is created.

Objective 1
A design environment for metamodels is conceptually defined and builds on re-use and combination patterns, supported by conceptual structures as a knowledge representation format. Conceptual structures, due to their close relation to linguistics and available knowledge operation via their formal representation as conceptual graphs are validated for their applicability to support design operations.

Research objective 1 aims to develop an environment for knowledge representation as conceptual structures that is adequate to describe metamodels. The underlying assumption is that metamodels can be represented as conceptual graphs. As sources and evidence for this assumption a) the definition of the term "metamodel" in the scientific community, b) existing metamodels from academia and industry and their characteristics and construction principles, c) meta-modelling techniques and patterns that are currently applied, and d) knowledge operation as requirements of metamodels are considered. The assessment of the foundation has resulted in the viewpoint that multiple *dimensions* can be identified on syntactical, semantical, behavioural and operational level that are applicable to categorise and structure metamodel patterns.

Objective 2
Based on the challenge of harmonisation and responsiveness to change, visualisation techniques are sought after that are useful to support an explorative experimentation in digital innovation strategies/processes. This objective targets the *usefulness* of metamodels for digital innovation: as discussed above, it can be observed that interrelation between models are mostly implicit and software artefacts such as API interaction are not transparent within heterogenous systems. Visualisation techniques support this transparency consideration in a way that any involved stakeholder is aware of the interaction a system needs to establish early in the development process during experiments.

Research objective 2 targets harmonisation and transparency needs required to a) describe and b) collect/support re-use of metamodels as conceptual structure within heterogenous modelling systems. Visualisation is needed as a principle to externalise knowledge and provide an assessment of interrelations among model artefacts, whereas the concept is realised on metamodel level.

As a "summarising" objective, the analysis results and their evaluation constitute value for the metamodelling community and therefore a collection of metamodels as libraries within a common repository is suggested.

Objective 3
Metamodels analysed as part of this thesis are categorised according to their characterising dimensions using the formal representation proposed in objective 1. The collection is made available to the community for use and further extension. The necessary interaction capabilities with such a metamodel repository are defined.

Having the motivational example as shown in Fig. 1.4 in mind, objective 3 targets the identification of patterns based on the derived dimensions and their exposure. Metamodels developed using modelling techniques of similar or varying expressiveness and functionality have been reviewed to identify techniques to describe patterns and provide means for e.g. generalisation/-abstraction of these patterns or fragments.

The thesis focuses on domain-specific modelling methods, more specifically on their metamodels as they constitute the domain-specific knowledge involved. The challenge of designing "adequate" metamodels to support dynamic (in the sense of quickly evolving and responsive) and complex (vertically and horizontally integrated) environments is considered.

Based on an initial understanding of "adequate" in relation with a domain-specific metamodels as derived through observation and discussion in (Karagiannis, 2015), where usefulness defines the scope of the abstraction performed in the design process (Karagiannis, 2015, p. 5). Requirements elicited during the design process (influenced by the metamodel designers knowledge and experience) and runtime aspects influence the process.

As a theoretical foundations, conceptual structures are applied. Within this thesis, the term conceptual structure is understood and closely linked to the work performed by Sowa in (Sowa, 1984) as a logic-based knowledge representation technique. Derived from linguistics, a conceptual structure allows the representation of concepts in terms of a small number of conceptual primitives ('Conceptual Structure' 2019) that can be expressed mathematically as a conceptual graph, following the discussion in (Jackendoff, 1992).

1.5 Research Methodology

The research methodology is developed in accordance with the memorandum for design-driven business informatics in (Österle et al., 2010). The memorandum enforces research work as a design process/phases. Relevance and rigor of results are considered as foundational elements in the research process. For the work performed in this thesis, this memorandum is applicable, detailed as the research framework developed and continuously refined. The framework considers the research process, the definition of the artefact and its evaluation criteria as well as diffusion and dissemination channels and considers the methodology discussed in (A. Hevner and Chatterjee, 2010) as a concretisation.

Fig. 1.8 presents the research framework as an instantiation of the above principles, applicable for the work presented.

Artefact. The Design-Model-Make Analysis Framework (DeMoMa-(*)) developed in this thesis as an artefact is characterised by its knowledge representation formalism for heterogenous modelling environments in information systems design and visualisation of interrelation of metamodels on operational level. The artefact is considered on a conceptual level as a construct and a method that aims to establish re-use patterns in the design of metamodels and visualisation techniques, and its applicability for design processes.

Research Process. The research process is discussed in 4 distinct phases. The research process follows the design guidelines articulated by Hevner et.al. in (A. R. Hevner et al., 2004, p. 83), condensed in (Österle et al., 2010, p. 4-5).

1. *Analysis*: the trigger and motivation for the work performed is related to the practices of digital innovation processes of information systems in industrial and academic settings and has been discussed in sections

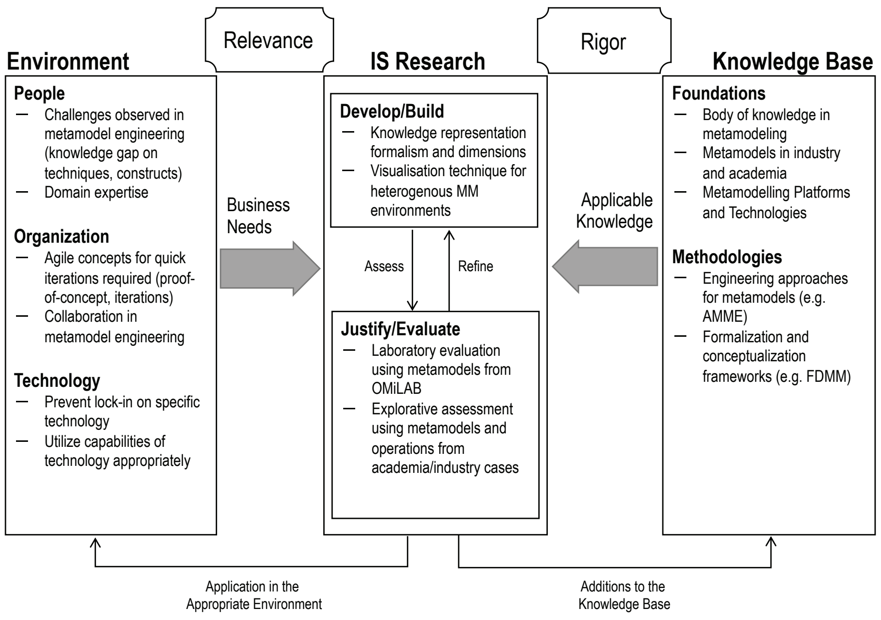

Relevance
Rigor
Environment
IS Research
Knowledge Base
People
— Challenges observed in metamodel engineering (knowledge gap on techniques, constructs)
— Domain expertise
Organization
— Agile concepts for quick iterations required (proof-of-concept, iterations)
— Collaboration in metamodel engineering
Technology
— Prevent lock-in on specific technology
— Utilize capabilities of technology appropriately
Business Needs
Develop/Build
— Knowledge representation formalism and dimensions
— Visualisation technique for heterogenous MM environments
Assess
Refine
Justify/Evaluate
— Laboratory evaluation using metamodels from OMiLAB
— Explorative assessment using metamodels and operations from academia/industry cases
Applicable Knowledge
Foundations
— Body of knowledge in metamodeling
— Metamodels in industry and academia
— Metamodelling Platforms and Technologies
Methodologies
— Engineering approaches for metamodels (e.g. AMME)
— Formalization and conceptualization frameworks (e.g. FDMM)
Application in the Appropriate Environment
Additions to the Knowledge Base

Figure 1.8: Design-Science Research Methodology based on (A. R. Hevner et al., 2004, p. 80)

1.1 and 1.2 defining the relevance for the work performed. Characteristics of agility, re-use and responsiveness to change are considered. The gap in current practices has been identified and is related to research objectives and directions.

2. *Design*: the design phase covers the construction of the artefact on a conceptual level. The knowledge base relevant for the design is defined as its foundation and influences the development process. Iterative cycles adapt the specification of the concept.

3. *Evaluation*: the artefact is evaluated using prototyping as a method. The applicability of the prototype is evaluated in an experimental setting assessing whether the artefact is adequate for the specific purpose of designing digital innovation and amalgamating heterogenous modelling system components.

4. *Diffusion*: Results of the iterative design and evaluation cycle are reported in relevant conferences and journals; the metamodelling community is addressed by the contribution made.

Evaluation methods. Evaluation methods and criteria for DeMoMa-(*)

have been defined in advance to assess the applicability of the result in design-driven innovation scenarios. The following criteria are selected, criteria are defined in accordance with (A. R. Hevner et al., 2004, p. 86)

– Analytical Evaluation: an assessment of the artefact is performed applying "static analysis". The criteria associated with the method is complexity to design and operationalise the a heterogenous modelling environment. The fitness to embed the artefact into existing technical information system architecture is assessed following an "Architecture Analysis".

– Experimental Evaluation: the usability of the artefact is demonstrated in the experimental, laboratory setting of the OMiLAB. Arbitrary modelling methods and their metamodels are selected as elements of a heterogenous modelling environment and extended with the concept of visualisation.

– Observational Evaluation: The applicability and relevance of the artefact is evaluated by applying it to a real business environment in the context of a collaborative research project.

Concrete evaluation criteria are defined in the evaluation chapter 7, following and scoping the work of (Prat et al., 2014) in the context of the developed artefact.

Dissemination. The result achieved during the different phases of the research process have been published in relevant conferences to disseminate the artefact, refine and integrate feedback in the iterative design - develop - evaluate phases.

An indicative list of selected publications is provided below.

Problem Relevance: Publications related to the exploration of the problem relevance.

– Wilfrid Utz and Dimitris Karagiannis (2009). 'Towards business and it alignment in the future internet; managing complexity in e-business'. In: *Proceedings - 2009 1st International Conference on Advances in Future Internet, AFIN 2009*

– Wilfrid Utz, Robert Woitsch and Dimitris Karagiannis (2011). 'Conceptualisation of hybrid service models: An open models approach'.

In: *Proceedings - International Computer Software and Applications Conference*

– Vedran Hrgovcic, Wilfrid Utz and Dimitris Karagiannis (2011). ‘Service modeling: A model based approach for business and IT alignment’. In: *Proceedings - International Computer Software and Applications Conference*

– Wilfrid Utz, Peter Reimann and Dimitris Karagiannis (2014). ‘Capturing learning activities in heterogeneous environments: A model-based approach for data marshalling’. In: *Proceedings - IEEE 14th International Conference on Advanced Learning Technologies, ICALT 2014*

– Wilfrid Utz, Robert Woitsch and Zbigniew Misiak (Nov. 2016). ‘Planning for integration: A meta-modelling approach using ADOxx’. In: *Measuring and Visualizing Learning in the Information-Rich Classroom.* Routledge, pp. 183–195

– Robert Woitsch and Wilfrid Utz (Feb. 2016). ‘Business Process as a Service: Model Based Business and IT Cloud Alignment as a Cloud Offering’. In: *Proceedings - 2015 3rd International Conference on Enterprise Systems, ES 2015.* Institute of Electrical and Electronics Engineers Inc., pp. 121–130

Design as an Artefact: Publications related to the ideation and design of the artefact.

– Nesat Efendioglu, Robert Woitsch and Wilfrid Utz (2016). ‘A toolbox supporting agile modelling method engineering: ADOxx.org modelling method conceptualization environment’. In: *Lecture Notes in Business Information Processing*

– Peter Reimann and Wilfrid Utz (2016). ‘Modeling learning data for feedback and assessment’. In: *Domain-Specific Conceptual Modeling: Concepts, Methods and Tools.* Cham: Springer International Publishing, pp. 555–574

– Wilfrid Utz and Robert Woitsch (2017). ‘A model-based environment for data services: Energy-aware behavioral triggering using ADOxx’. In: *IFIP Advances in Information and Communication Technology*

– Nesat Efendioglu, Robert Woitsch, Wilfrid Utz and Damiano Falcioni (2017). 'A Product-Service System Proposal for Agile Modelling Method Engineering on Demand: ADOxx . org'. In: *Digital Enterprise Computing 2017*, pp. 199–212

– Hisashi Masuda, Wilfrid Utz and Yoshinori Hara (2013). 'Context-Free and Context-Dependent Service Models Based on "Role Model" Concept for Utilizing Cultural Aspects'. In: *Proceedings - Knowledge Science, Engineering and Management. KSEM 2013.* Ed. by Mingzheng Wang. Springer Berlin Heidelberg, pp. 591–601

– Christoph Moser, Robert Andrei Buchmann, Wilfrid Utz and Dimitris Karagiannis (2017). 'CE-SIB: A modelling method plug-in for managing standards in enterprise architectures'. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*

Design Evaluation: Publications related to the evaluation of the design artefact.

– Wilfrid Utz and Moonkun Lee (July 2017). 'Industrial Business Process Management Using Adonis Towards a Modular Business Process Modelling Method for Zero-Defect-Manufacturing'. In: *2017 International Conference on Industrial Engineering, Management Science and Application, ICIMSA 2017.* Institute of Electrical and Electronics Engineers Inc.

– Wilfrid Utz (2018b). 'Design metamodels for domain-specific modelling methods using conceptual structures'. In: *CEUR Workshop Proceedings.* Vol. 2234, pp. 47–60

– Wilfrid Utz and Damiano Falcioni (Sept. 2018). 'Data Assets for Decision Support in Multi -Stage Production Systems Industrial Business Process Management using ADOxx'. In: *Proceedings - IEEE 16th International Conference on Industrial Informatics, INDIN 2018.* Institute of Electrical and Electronics Engineers Inc., pp. 809–814

– Dimitris Karagiannis, Dominik Bork and Wilfrid Utz (2019). 'Metamodels as a Conceptual Structure: Some Semantical and Syntactical Operations'. In: *The Art of Structuring.* Cham: Springer International Publishing, pp. 75–86

– Dominik Bork, Hans-Georg Fill, Dinitris Karagiannis and Wilfrid Utz (2018). ‘Simulation of Multi-Stage Industrial Business Process Using Mmetamodelling Building Blocks with ADOxx’. In: *Journal of Enterprise Modelling and Information Systems Architectures* 13.2, pp. 333–344

– Hisashi Masuda and Wilfrid Utz (2019). ‘Visualization of Customer Satisfaction Linked to Behavior Using a Process-Based Web Questionnaire’. In: *Proceedings - 2019 12th International Conference on Knowledge Science, Engineering and Management*, pp. 596–603

– Wilfrid Utz (2019a). ‘Design of a Domain-Specific Metamodel for Industrial Business Process Management’. In: *Proceedings - 2019 8th International Congress on Advanced Applied Informatics (IIAI-AAI)*. vol. 2019. Toyama: IIAI, pp. 821–826

– Wilfrid Utz (2019b). ‘Support of Collaborative Design Thinking using ADOxx’. In: *Proceedings - 2019 International Conference on Innovation and Management*. Hiroshima

– Dimitris Karagiannis, Patrik Burzynski, Wilfrid Utz and Robert Andrei Buchmann (2019). ‘A Metamodeling Approach to Support the Engineering of Modeling Method Requirements’. In: *Proceedings - 2019 27th IEEE International Requirements Engineering Conference*, pp. 199–210

As a summary, the characteristics of the artefact are introduced to provide the necessary context for the foundation and related work chapters. The DeMoMa-(*) artefact developed is considered as a concept to support the design and visualisation of model interrelation and functionality in heterogenous environment. As such it consists of the following elements:

– Abstract metamodel patterns for visualisation support as *constructs*: patterns considered elevate existing and support the design of new metamodels for homogenisation.

– A *method* for design and combination within heterogenous modelling environments, and

– *Instances* for an experimental evaluation according to the evaluation criteria of applicability, usability and fit for use in complex settings.

In the following section, the structure of the thesis is introduced to guide the reader along the "red-thread" of chapters and sections. Fig. 1.9 shows graphically the dependencies (inputs and outputs), including the design/evaluation cycle.

1.6 Structure

The thesis is structure in seven chapters following the research process defined previously. Chapter 1 introduces the observations made and formulates the problem statement. The problem statement is input to the relevance consideration of the research methodology. Chapter 2 establishes the rigor, reviewing and discussing the foundations and related work in the field.

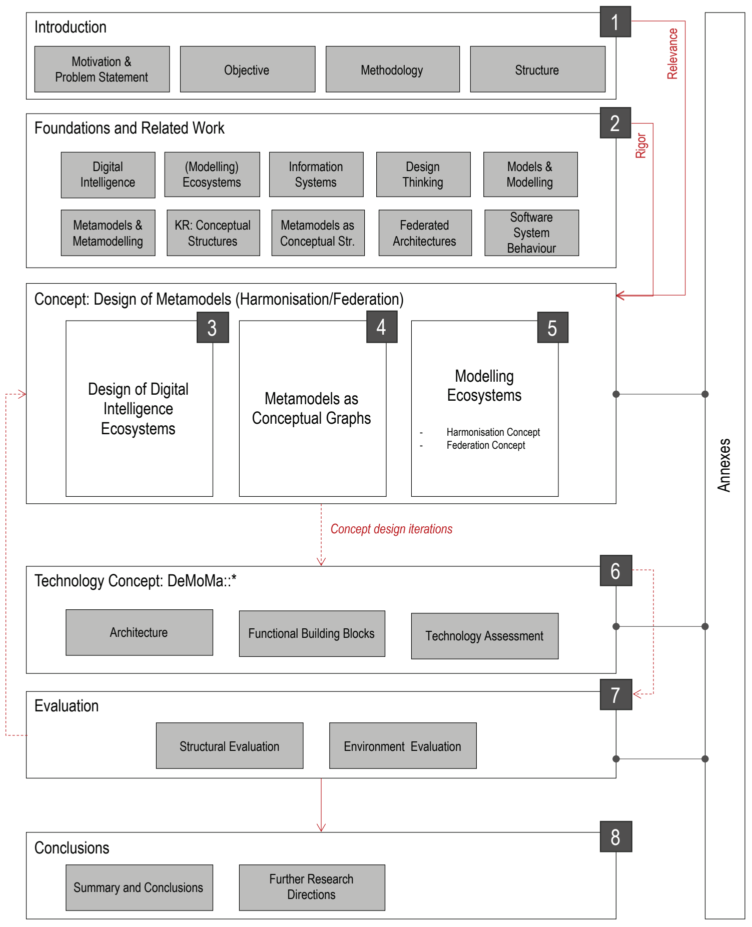

Introduction
1
Motivation & Problem Statement
Objective
Methodology
Structure
Relevance
Foundations and Related Work
2
Rigor
Digital Intelligence
(Modelling) Ecosystems
Information Systems
Design Thinking
Models & Modelling
Metamodels & Metamodelling
KR: Conceptual Structures
Metamodels as Conceptual Str.
Federated Architectures
Software System Behaviour
Concept: Design of Metamodels (Harmonisation/Federation)
3
Design of Digital Intelligence Ecosystems
4
Metamodels as Conceptual Graphs
5
Modelling Ecosystems
- Harmonisation Concept
- Federation Concept
Concept design iterations
Technology Concept: DeMoMa::*
6
Architecture
Functional Building Blocks
Technology Assessment
Evaluation
7
Structural Evaluation
Environment Evaluation
Conclusions
8
Summary and Conclusions
Further Research Directions
Annexes

Figure 1.9: Roadmap of the Thesis

Both chapters (motivation and related work) are input to the artefact/-concept development chapters. Chapter 3 develops the artefacts as a conceptualisation of design techniques of metamodels considering harmonisation and federated functionality, iteratively defining the conceptual structure, patterns, interaction behaviour and development methods, whereas chapter 6 transforms the results into a technological realisation concept as input for evaluation.

The evaluation chapter introduces in its subsections the different types of evaluation performed (prototype feasibility, laboratory assessment and application on a real-world case in the field of Industrial Business Process Management (IBPM)). Feedback loops to the conceptualisation phase stemming from prototypical cases and their evaluation are cross-referenced in chapter 7.

Summary and concluding remarks are discussed in chapter 8, defining further research fields and directions based on the observations during the evaluation phase.

Chapter 2

Foundations and Related Work

The chapter on foundations and related work discusses the relevant body of knowledge for the research work presented. A literature review of related work has been performed for the artefacts of digital ecosystems, intelligence information systems, conceptual models, metamodels, knowledge representation, visualisation techniques of software-based systems and federation concepts in computer science. Each subsection is divided into two parts:

1. Foundations: for the specific field definitions are introduced in light of the observations and the problem statement. The technical and conceptual perspectives are considered.
2. Related Work is discussed for each thematic field identified. The related work is reviewed for pre-existing knowledge as input for the concept development/design of the artefact. The discussion of related work introduces the building blocks of the design artefacts, its reception in scientific literature and industry.

The thematic fields discussed below, have been derived from the research objective to develop an artefact that is adequate to a) support information systems design, using b) metamodels represented as conceptual structures for the purpose of explicating interrelations and interactions in heterogenous, complex systems for the purpose of c) digital intelligence within modelling ecosystems.

2.1 Digital Intelligence

Foundation. The term "digital intelligence" has become popular in recent years, describing the literacy requirements of human actors in a digital world. It stems from the field of digital education, summarised by the DQ Institute as "a comprehensive set of technical, cognitive, meta-cognitive, and socio-emotional competencies that are grounded in universal moral values and that enable individuals to face the challenges and harness the opportunities of digital life" (DQ Institute, 2020). The educational aspect of the above citation demonstrates the multi-dimensional aspects considered within the term. This is also articulated in the publications of (Adams, 2010; Adams, 2004) who relates the technology advances with the need to grasp and understand the underlying systems cognitively.

This view of cognitive understanding of a system, its structure and behaviour is applicable for the work performed in this thesis: information systems are not a monolithic environment, handled by experts in close collaboration with business experts, but continuously evolve beyond the controlled boundaries of an enterprise. Systems are directly used for a single, temporal purpose impacting related fields. The term intelligence is therefore used in a broad sense, denoting the capability to understand, relate and interact with systems/actors in a heterogenous setting, but maintaining the syntactical and semantic links between the artefacts.

Related Work. Related work in the field of information systems and modelling is can be retrieved for the development of intelligent systems and services. The related term is "Business Intelligence" that has been defined in (Negash and P. Gray, 2008) as an emphasis on the "analysis of large volumes of data about the firm and its operations"(Negash and P. Gray, 2008, p. 175) as input for decision support systems. This viewpoint is too narrow for the cognitive aspect introduced above: to understand/assess a complex, connected system and provide functionality to grasp the meaning of these interactions of these systems rather than on analysing the historical perspective of events and actions that have already taken place.

Definition 2.1: Digital Intelligence
Digital intelligence is defined as a cognitive function on an information system that supports a systematic approach to enable human actors to understand the impact of technological advances within transformative innovation processes. The explorative nature of these intelligence function is considered a building block in the development of digital literacy skills and competences (intelligence as a cause and effect relation within an information system).

Digital intelligence platforms as assessed by Forrester in (Mccormick et al., 2019) follow the technology-centred viewpoint. These platforms provide essentially functionalities to transform data into actionable, customer-centric insights - a marketing-oriented offering on digital data management, analytics, experience optimisation, Artificial Intelligence (AI)/machine learning and integration capabilities. The 2019 leader in the report has been Adobe, interestingly without any domain-specific assessment of the enterprise's offering.

In the context of this work, and in line with the definition above, the term "digital intelligence" is understood in a broad sense, as discussed in (Adams, 2004), "resulting from human interaction with digital computers" (Adams, 2004, p. 94). Based on Gardners, seven theories and classification of intelligence, initially defined in (Gardner, 1993) and extended in (Gardner, 1999) are applicable. The author claims that the combination of these categories result in the emergence of a novel, meta-intelligence. It is not sufficient to understand the functional aspects of a program or technology service, but assess and evaluate continuously the "hidden" elements dynamically composing the behaviour of socio-technical systems in a digital world.

2.2 Digital Ecosystems

The term ecosystem, stemming from the discipline of biology, is applied as an analogy to the distributed manner and their dependencies among and contributions of participants towards a common objective/offering. Nodes in the ecosystem act according to their hierarchical position and cause-effect relations towards sustaining the overall system. Translating this viewpoint to the digital domain (and excluding business ecosystem of technological offerings such as the famously cited ecosystem-based business models of IT giants), a focus for the review of related work is set on the functional aspect in co-creation of capabilities in a digital world.

As such, digital ecosystems are considered in literature as an evolution of the service-oriented paradigm: distributed functionalities (exposed as services) act autonomously, harmonisation and integration is based on the semantic annotation of service functionalities (as exposed by their APIs). Abstracting from this technical viewpoint, enterprises that act in a collaborative fashion, need to clarify how such interaction (on strategical, tactical and operational level) influence their operation and which role they play at a given point in time. Understanding these relations from an internal and external perspective is essential and in-line with the discussion on intelligence capabilities of any enterprise.

Foundation. (Briscoe and De Wilde, 2006) discuss the above observation. By adding evolutionary computing concepts to Service-Oriented Architecture (SOA), self-organising systems are established. Services (as functionalities) in these system align themselves within the ecosystem, adapt to user interaction and behaviour aiming at "increasing the effectiveness of the user base"(Briscoe and De Wilde, 2006, p. 1) continuously. At the core, two optimisation schemes/levels for this re-adaptation are introduced - services understood as agents operate in a peer-to-peer collaborative manner and optimisation aims at the network structure and negotiation between nodes, whereas these inputs feed into the self-organising/evolutionary algorithm within each node.

This technology view is derived from the understanding that has evolved in the business world. The adoption of the term "ecosystem" is attributed to the work of James Moore on competition. In (Moore, 1993), which is referred to as the "birth of business ecosystem", Moore describes the evolutionary steps and phases of such an ecosystem in relation to its biological counterpart. Referring to "Gregory Bateson's definition of co- evolution in both natural and social systems" (Moore, 1993, p. 75), the organisational and structuring aspect are introduced. The leadership aspect are a crucial element in this self-organising process of an ecosystem.

Related Work. Related work in relation to ecosystems and the topic of this thesis is available with respect to the evolution of modelling ecosystem services. (Burkhard et al., 2013; Crossman et al., 2013) introduce the concept of modelling for ecosystem services. The modelling and mapping of these services is required to capture the value flows within the ecosystem and quantify those. Modelling is applicable to enable a dynamic and flexible assessment of the enterprises and organisations taking part in the system and therefore assess how value is distributed within the network of nodes. A core aspect within the papers is related to the domain-specifics of each service offering and how such aspects can be mapped. Assuming that an abstraction (via modelling) explicates the knowledge captured by the service, a domain-specific modelling technique is required.

Combining the technical viewpoint on SOA and its evolution towards ecosystems with the domain-specific aspects in business/enterprise terms, poses a semantic challenge. Modelling and mapping techniques are specific to the node type, industry background and technical abstraction. (Blanc et al., 2005) introduce a technical contribution towards this challenge: modelling services (in the field of software engineering) are combined and composed using a common model bus as an integration layer. Such a bus though, contradicts the idea of an ecosystem as a self-organising system as a controlling instance is introduced. Loose-coupling approaches have been introduced in (Woitsch, 2013), nevertheless these techniques build on the content (models) rather than on the metamodel representation and a conceptual alignment - a gap that the work presented in this thesis aims to narrow. Since metamodels constitute the knowledge (in abstract terms), modelling services need to expose these semantics for a semantically-rich combination and integration, resulting in the below definition

Definition 2.2: Digital Ecosystem
A digital ecosystem is defined as a network of nodes that are involved in the provision of value. Each of these nodes or entities affects or is affected by the behaviour of related nodes; specific characteristics and types of nodes in the system can be recognised. Due to these relation, an ecosystem has self-organisation and optimisation capabilities. Digitalisation reinforces the possibilities of interaction in the environment. Relations become location and time-independent.

Following the above definition, a specialisation has been derived for modelling ecosystem.

Definition 2.3: Modelling Ecosystem
The viewpoint is based on the assumption that it is possible to understand domain-specific aspects of modelling as nodes within an ecosystem of enterprise hierarchies and beyond its organisational boundaries. This means that each aspect that is relevant for intelligence and understanding are the result of a modelling services that interact autonomously with each other. Such a system is coined modelling ecosystem; the phases of "Birth", "Expansion", "Leadership", "Self-Renewal" as introduced in (Moore, 1993, p. 77) are applicable for individual nodes or modelling services, sets or the whole ecosystem.

The abstract view on digitalisation from an intelligence and ecosystem perspective is detailed, focusing on the purpose of information systems design and the interactions with such services.

2.3 Information Systems

The objective of this thesis is to support the metamodel engineer in defining an adequate environment for the design and analysis of an information system. Digital innovation trends disrupt the way the term information systems is understood.

Foundations. The Oxford Dictionary of Computer Science defines an information system as "the branch of knowledge concerning the purpose, design, uses, and effects of information systems in organisations" (Information Systems, 2016). The aspect of interdisciplinary is considered important as it draws, as a concept, from the field of computer science as a technical perspective and business/economical studies from an organisational viewpoint.

A similar definition of the term, also derived from the viewpoint of combination, is presented in (Alter, 1991), who characterises information systems as "a combination of work practices, information, people and information technologies organised to accomplish goals in an organisation" (Alter, 1991, p. 7). As such an information system is a socio-technical system that considers the collection, processing and interpretation of information by human and machine actors, "improving the effectiveness and efficiency of that organisation" (Silver et al., 1995).

A comprehensive definition, considering the information/knowledge intense nature of an information systems is presented the Encyclopaedia Britannica. The definition positions the term as an "integrated set of components for collecting, storing, and processing data and for providing information, knowledge, and digital products" (Zwass, 2017). The distributed nature becomes obvious from definition, as integration of systems on different levels of abstractions have to be considered and a single system supporting an information system holistically is not feasible (see (Vázquez, 2012, p. 21)).

Definition 2.4: Information System
An information system is understood as a collection of systems targeting specific needs of an enterprise with respect to information management considering the aspects of processes, information/data, people and technology in a holistic and integrated manner. Different types of information systems are distinguished in literature, traditionally ranging from Executive Information Systems (EIS), Decision Support Systems, Management Information Systems (MIS) to Transaction Processing systems, extended towards intelligence information systems that enable collaboration and interaction. It provides information about the organisation and its environment (Avison and Fitzgerald, 2006).

For this thesis a distinct understanding of an information system is applied, namely the knowledge management perspective of intelligent information systems to support innovation processes and strategies. These tasks are understood as knowledge intense (such as digital innovation processes and strategies) and require approaches for creativity, consistency, and interrelated evaluation.

Based on the observation on complexity as system-of-systems, the management of such systems is considered a challenge as the methodologies for design, analysis, specification and implementation have to be understood in the context of evolving environment and varying stakeholder needs.

Fig. 2.1 shows the phases/processes and artefacts in information systems design in relation to conceptual modelling. An enterprise is represented by its structure and semantics of the underlying information system. The design (forward-looking) and analysis (historical perspective) of an information system utilises information modelling that is understood as a subfield of conceptual modelling. Information models are conceptual models, that are created, employing various modelling methods (structured according to the fields and levels of the enterprise).

The challenge of human interpretation and combination is addressed in (Wand et al., 1995) as the authors investigate into three theories to support the idea on using human knowledge as the foundation for conceptual modelling in systems development: ontology, concept theory and speech-act theory. The discussion of these theories build on the assumption that con-

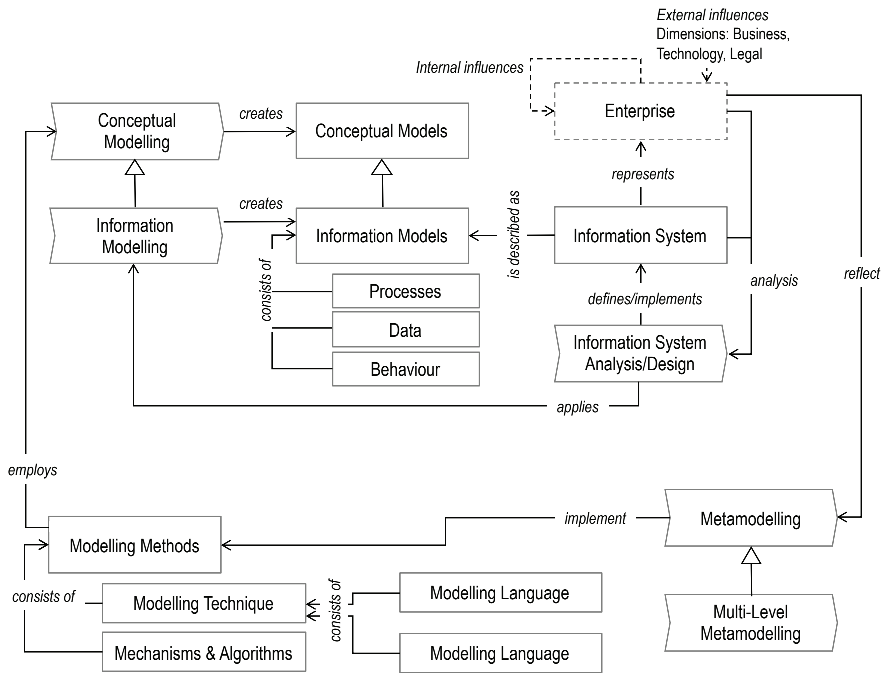


Figure 2.1: Relation Information Systems and Conceptual Modelling

ceptual models are an essential element in the design process to capture the knowledge and interactions required. Methodologies to support the process systematically are discussed in literature influenced by changing trends and influences. In the following the common characteristics of the methodologies are assessed and a definition applicable for this thesis is derived.

The traditional viewpoint of information systems methodologies is described in (Olle et al., 1988) as "a methodological approach to information systems planning, analysis and design". The term methodology is considered synonymously to method; the constituting elements of planning, analysis and design distinguish temporal aspect of the process from understanding of the current system, to detailing change requirements and specification towards implementation and operation. The requirements for methodologies has shifted from requirements and documentation support for a computer-based, digital system towards collaboration and comprehension of design decisions. This viewpoint is discussed already in (Avison and Fitzgerald, 2006) as a shift in roles of people the process. The authors discuss techniques, tools and methodologies that are applicable resulting in a "jungle" of methodologies. Therefore, the definition of the term is presented from an abstract standpoint. A common characteristic is the externalisation demand of knowledge during the phases of the methodology.

In relation with the research objectives defined, the design phase of information systems methodologies are in scope.

Definition 2.5: Information Systems Methodology
Information systems methodologies describe, in a broad sense, as a method the tasks and results for the analysis, design and specification of an information system. The methodologies vary depending on domain requirements and are adapted in the context of the environmental trends.

Related Work. The approach to use conceptual models for the development and management of information systems has already been articulate in (Olle et al., 1988) discussing information systems methodologies. The authors distinguish in their discussion between the analytical (looking at the existing systems) and design aspects (creating a new system or components). The combination of both parts, named "Design product" is considered the input for system specification and implementation.

From the background of innovation processes, the aspect of information systems design, especially in the digital transformation age, has been studied in (Delmond et al., 2017). The author argue that the notion of information systems has evolved in the past from "inter-organisational to value nets" to support processes carried out jointly and enable the sharing of knowledge.

In line with the value net considerations, (Kim et al., 2010) review the role of IT within business ecosystems and discuss the four characteristics of a) robustness, b) creativity, c) interoperability and d) productivity along the dimensions of knowledge intensity and environmental velocity. The impact of Information Technology (IT) as an "Intelligent Service Platform" is introduced, that has to "respond to internal and external stimulation", "senses external changes", enables actionability, "analyses intelligent information" and "protects the flow of health information and knowledge". These requirements on intelligence of information systems has been reflected historically in (Brodie, 1989), who argues in 1989 for a combined view on data and AI to support collaborative work in enterprises. Smart and intelligent systems are a trending topic in computer science in various domains and application fields (e.g. in (Meder et al., 2015) for an collection of articles in different domains).

As an interdisciplinary field, not only technological evolution impacts the way information systems are understood. The trend in business to follow a service logic in providing "intelligent offerings" has changed the way system support is required. (Bohmann et al., 2014) discuss the relevance to the field of information systems research, as service-dominant logic emphasises *collaboration* and *contextualisation*. The context-specific value is co-created by the value network. Such considerations impact the design and engineering of information systems. The authors discuss three dimensions to be considered in the systematic, knowledge intense design process of such information systems to define the a) service architectures, b) service systems interactions and c) resource mobility. Such aspects influence the way design methodologies are defined and applied.

The need for models and modelling is well established in literature. Methodologies, independent whether they are analytical, design or specification related, employ modelling as a tool and the resulting models are an essential building artefact for system comprehension. The requirements of collaboration, co-creation and contextualisation are reflected upon in the next section.

2.4 Design Thinking

Design thinking has gained in popularity in recent years as a set of methods and tools, inspired by product design and development, have evolved to support strategic alignment of enterprises towards market demands. From an organisational point of view, the involvement of stakeholders is regarded as a key aspect. Co-creation of innovative approaches in multi-disciplinary teams is, according to (Brown, 2008), a "principal source of differentiation and competitive advantage" and design thinking builds upon this need by matching methods of designer with needs of stakeholders.

From a temporal perspective, design thinking targets to develop future states and outcomes, building on the expertise and experience of involved participants. This includes the historical evolution implicitly.

Foundation. The foundation of design thinking approaches are prominently introduced in (Rowe, 1987). Rowe's work defines and accounts toward the systematic approach followed in design thinking to establish a novel and innovative artefact based on a well-defined problem space within the field of architectural design. A new take on design thinking, applicable for the business domain, is presented in (Brown, 2008), positioning the approach as an iterative process for innovation. The problem space is explored in co-creation manner, resulting in prototypes that can be tested, explored and its feasibility becomes assessable.

Related Work. Related work in the context of the research topic is concerned with the explorative nature to enable cognition and understanding of complex systems, their evolution and the manifestation of innovative ideas in a distributed and multi-disciplinary setting. (Plattner et al., 2011) discuss the applicability of design thinking in general, and introduce specific methods and tools for application domains such as co-evolution, collaboration and prototyping. At the core of the contribution, the exploration of a potentially complex problem space, following a systematic approach that spans across functional and organisational borders is positioned.

Definition 2.6: Design Thinking
Design thinking is positioned in the context of the work presented as an explorative method that enables the iterative systematic assessment of a system and its evolution involving the knowledge and expertise distributed in an enterprise ecosystem.

As it can be seen from the definition above, design thinking requires flexible tools and capabilities to adapt to the problem space and its specifics. Modelling is considered an approach that enables the externalisation of knowledge and expertise in a domain-specific manner, especially in the context of information systems design as discussed in (Wieringa, 2014). The theoretical foundations for modelling in information systems as discussed in (Wand et al., 1995) is therefore applicable.

2.5 Models and Modelling

The definition of the term "model" in computer science, has according to Stachowiak in (Stachowiak, 1973, p. 129) a double meaning: a model in its descriptive nature, representing a phenomena of the real world as well as a prescriptive meaning as a design artefact, defining how a system or environment could look like. The German terms "Abbild" and "Vorbild" describe this distinction of the term.

Modelling as an activity is concerned with the creation, analysis and assessment of models. In the following the relevant literature within the field of computer science is presented. Views on models and modelling in other fields such as economics, psychology, etc. are excluded from the discussion below and the foundation section focuses on conceptual models/modelling.

Foundation. According to (Thomas, 2005), four definition approaches can be observed in literature for the term model:

1. General definition: according to (Stachowiak, 1973), three generic characteristics are established. A model is a *representation* of a natural or artificial originals (which can be a model as well, therefore becomes a model of a model), models reduce complexity, commonly translated to "truncation" and follow a pragmatic approach in its construction and application, considering the subjective view of the modeller, the purpose for creation and temporal validity.

2. Model as representation: the view of a model as a representation has a long tradition in business/economics and business informatics. The assumption is that a model is an adequate representation of the original that exists in reality (in contrast to above, where representation is also considered for virtual/artificial objects). The definition assumes that reality already exists in a pre-structured way to be represented. According to (Thomas, 2005), this definition is subject to criticism as the value of observation and design are neglected.

3. Model as construction: In contrast to the representation definition, the models as constructions consider the complex construction and design process of models. The modeller acts as an observer of phenomena in reality as well as constructs/designs possible solutions as a model.

4. Axiomatic understanding: following the mathematical tradition in logic, a set of axioms construct an axiomatic system that, according to the definition of model theory in (Marker, 2002) can be understood as a model.

For the purpose of this thesis, the general approach is followed, elevated by considerations on models as constructions, further extended by the viewpoint on models in (Thalheim, 2011), who details the common characteristics of models as a "Purpose", "Mapping", its "Language as a Carrier", and "Value", resulting in additional characteristics for models as amplification, distortion and idealisation. Especially, the additions on language, to structure and also limits the expressiveness of models and value considerations impacted the research work presented in this thesis.

Definition 2.7: Conceptual Model
A model is defined as the subjective construction, that results from a specific/real or artificial phenomena, following a specific purpose and therefore reduces and scopes the complexity using a language for representation. A model pertains value based on its utility, capability and quality characteristic.

Modelling is the activity to create models, applying a specific modelling language. From a conceptual perspective, it is defined in literature as "the activity of formally describing some aspects of the physical and social world around us for the purposes of understanding and communication" (Mylopoulos, 1992, p. 3). The formal aspect of modelling in this definition relates to the application of a modelling language, represented as a metamodel.

Definition 2.8: Conceptual Modelling
Modelling as an embedded activity, within a method, is concerned with the creation of models of any kind of abstraction and is considered an analytical as well as design task. Modelling applies a modelling language that is appropriate for the specific need and creates models using the defined expressiveness of the language.

Each model created conforms formally to the language used and the underlying metamodel of the language. Historically, modelling in computer science has evolved from the design and specification of database systems towards software systems engineering and is now broadly understood and deemed essential in the field of information systems methodology. (Olle et al., 1988) describe modelling as an essential element within any methodology related to the analysis and design as a knowledge-intense task to externalise and structure the SuS. Fig. 2.2 shows graphically the differentiation between observation and design (called Construction or "Gestaltung" in the original).

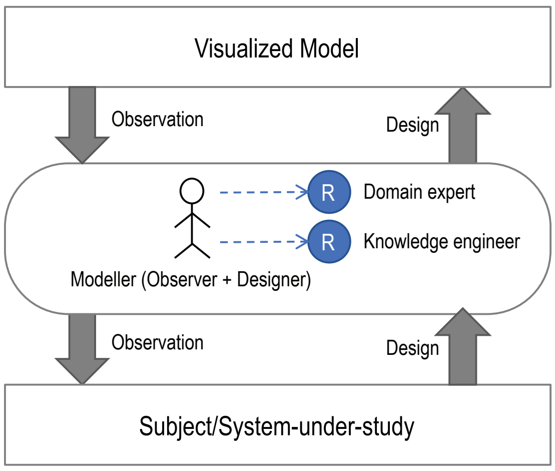


Figure 2.2: Conceptual Modelling: Observation and Design (translated and adapted from (Krcmar, 2015))

Related Work. Conceptual modelling, concerned with the process to develop a graphical, human interpretable graphical representation derived from the real-world phenomena has been characterised by the ability to abstract, truncate and structure relevant aspects observed. The design and planning aspect of the abstraction activity in the field is well-received in the scientific world and industrial applications.

A challenge observed relates to the development of adequate modelling languages to support the rigor required for implementing a system (Frank, 1999, p. 696), a challenge discussed in detail in section 2.6. Typically, such abstraction activities require the involvement of two roles: the domain expert, who possesses the knowledge of the field/SuS and the knowledge engineer in a supporting role to select and transform the knowledge into a representation that is appropriate for the purpose, respecting the formalisms decided for (Walch, 2019, p. 46). The level of knowledge engineering relates to the formality of the modelling language, the purpose of modelling (in relation with processing techniques that create value).

Considering different stakeholders involved in the process, that need to express their view on the system using differing formalities (e.g. a business model designer, describing a new way to offer services and products, a business analyst reviewing the operational practices and rules related to them, and an implementation expert defining the Information and Communication Technology (ICT) elements, the modelling process and models created vary in formality, expressiveness and value considerably. Creative approaches as stimulated by the design thinking community are contrasted by classical software engineering methods.

2.6 Metamodels and Metamodelling

This section covers the understanding of metamodels and their construction principles, identified as the activity of metamodelling.

Foundation. Metamodels define the domain-specific expressiveness of the models created with them. The metamodel can therefore be understood as a knowledge representation of the domain in focus, provides the formal structure as syntax, the meaning in the form of semantics - both relevant for processing capabilities - and notational aspects for comprehension. The literature on metamodels and metamodelling is diverse, in a sense that different fields in computer science have established distinct viewpoints and understanding on the topic.

These views are introduced below as the foundation of the definition presented.

Language-based Metamodels: Strahinger investigates in (Strahringer, 1995) the term "metamodel" and how it is used in scientific literature. 24 definitions have been assessed, showing the ambiguity of the term and its application in scientific literature of the domain. The author concludes by constructing the term using a language-oriented approach called "Metaisierungsprinzips", detailed in (Strahringer, 1998) as a generalisation.

The language-based understanding of a metamodel builds on the classification of models and specifically has linguistic models in graphical or verbal form in scope. Applying language-level theory or metalanguage theory, as introduced by Essler, it can be stated, that metamodels can be considered as a model of a model. This higher level model is a linguistic model, that represents the language of the lower level. As an abstraction, there is no limits to the hierarchical levels.

Definition 2.9: Metamodel

A metamodel is defined as a model of a model. Following a linguistic definition, the metamodel provides the language constructs that are available to construct the model. The hierarchisation of models to metamodel, meta^{2}models as an abstraction enables generalisation and specialisation of language constructs. Expressiveness of a metamodel is a result from the structural, syntactical definition and its semantic anchoring within the domain.

Situative Metamodels: The purpose considerations of metamodels have been discussed in (Brinkkemper, 1996), reflecting the method and tooling in information systems developments from a situational perspective. Specific needs of the development process require methods that are enabled by tools (in a broad sense, not only IT tools). The construction of tool support is demand-driven and needs to respect the specific situation, resulting in language elements and constructs that are applicable. Semantic soundness discussed as "ontological anchoring" of method fragments in (Brinkkemper et al., 1999) is required: "Method fragments should be anchored, i.e. described in terms of unambiguously defined concepts and, possibly, associations of an anchoring system". An extended view on situational method engineering that influenced this thesis is available in (Henderson-Sellers and Ralyté, 2010), and the explicated composition process in (Ralyté and Rolland, 2001).

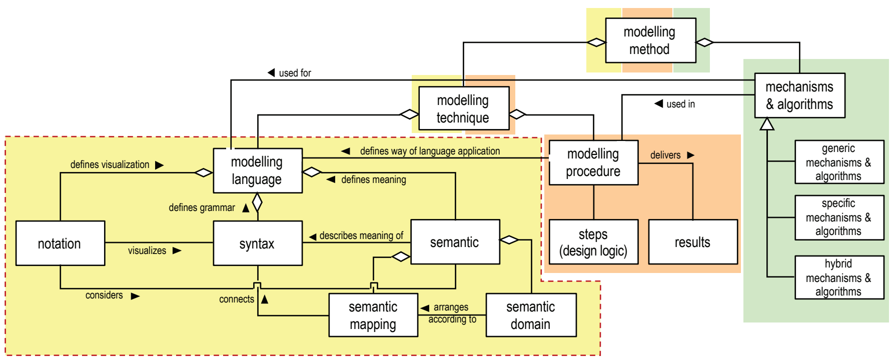


Figure 2.3: Generic Modelling Method Framework
(Karagiannis and Kühn, 2002)

Metamodels in Model-Driven Development Atkinson/Kühne review in (Atkinson and Kühne, 2003) types of metamodels used in Model-Drive Development (MDD). They identify abstraction techniques that are currently used and applied by software engineers to write higher-level code: traditional, OMG based modelling infrastructures such as the UML superstructure and Meta-Object Facility (MOF), linguistic metamodelling and ontological metamodelling are applicable. In the field of software engineering, metamodels are used to provide a) domain-specific languages that are closer to the domain (as DSML in contrast to GPML and b) establish the formality for model transformation, for software engineers, the generation of executable source code.

Ontological View on Metamodels: The ontological representation of metamodels has been extensively researched in the past following the objective to enrich metamodels (type semantics) and models (inherent semantics) with elevated expressiveness e.g. to cover semantic interoperability requirements (see (Höfferer, 2007)). The ontological representation is discusses as a synergy to metamodels as it is assumed that a formal foundation for a domain can be established (Wand, 1996). In general, these synergies are discussed in literature with ambiguity, as summarised in (Höfferer, 2007, p. 1624-1625) as some authors consider them equivalent and of the same nature, others argue for a distinction of syntax (within the metamodel) and semantics (within an ontological representation).

Definition 2.10: Metamodelling

Metamodelling is defined as the design and construction process of a metamodel that applies abstraction of concepts in a specific domain into appropriate meta-types, provided by the hierarchical meta-x level above. As such metamodelling considers the meta-x constructs, applies applicable abstraction techniques (e.g. association, generalisation, specialisation and hierarchisation) to define the syntax of a language. Ontological anchoring provides the semantic alignment and mapping to the domain.

The viewpoint of elevated semantics is followed within this thesis as ontologies are understood as a knowledge representation, as a model, that supports processing techniques due to its representation format.

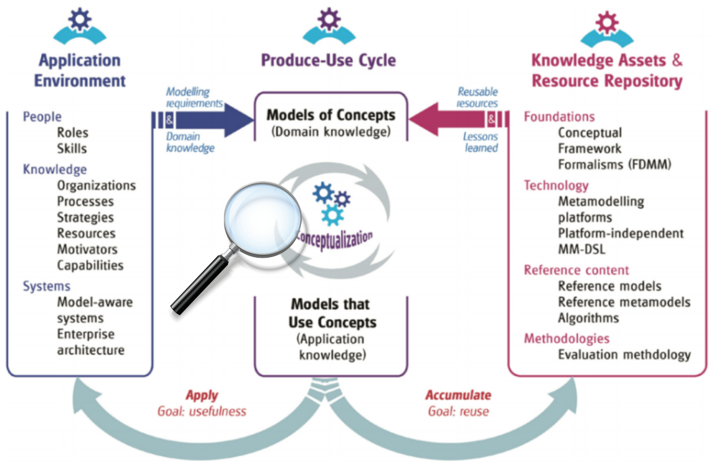
The above views on metamodels are the reduced set of definitions, applicable for the work performed and related to the research objective. Specifically, the related work presented targets adaptivity and semantics/intelligence in metamodels. The contribution in the field of software engineering and the applicability of metamodels in (Walter and Ebert, 2009; Ebert and Franzke, 1995; Ebert, Süttenbach et al., 1997) as well as the approach by (Jeusfeld et al., 2010) on a logic-based metamodel derived from data interoperability concerns is relevant.

Metamodelling Platforms: Karagiannis/Kühn introduce in (Karagiannis and Kühn, 2002) a generic framework for the structure of modelling methods and establish the definition of a metamodel on meta^2 level. Following the notion of language-based metamodels, the framework is composed of the model technique (consisting of the modelling procedure and modelling language) and processing functionalities as mechanisms and algorithms to be considered during the conceptualisation. The framework is considered as the guiding thinking model for the thesis as it provides a formal structure for the elements and their dependencies/associations to be considered.

As an aggregation of the language-based definition within the syntactical element, situational aspects in the form of modelling procedures, ontological foundation as the semantic mapping and domain and operationalisation as mechanisms and algorithms its relevance is given. Fig. 2.3 shows the elements that result in a modelling method graphically. The relevant part for the thesis, concerning structural aspects is the modelling language, highlighted in the figure.

Related Work. Related work considered for the theme of metamodels and metamodelling has been reviewed in the scope of multi-layer and -perspective modelling in combined environments. As the objective set forth is on an artefact that supports the interrelation of different layers of abstractions in information systems design, a review of literature establishes the foundation for the development of the concept of DeMoMa-(*).

The challenge of combining metamodels for different stakeholder needs has been discussed in literature extensively. This can be related on one hand to the issue of DSML, that aim to be close to the modeller and address the specific needs of a domain and increase productivity; on the other hand, effort and economics of scale considerations impact the design and development of such language artefacts, introduced in (Almeida et al., 2018, p. 24). The need for techniques to compose and integrate DSMLs appropriately is discussed from various viewpoints. (Kühn, 2004) has developed in his PhD a)



b)

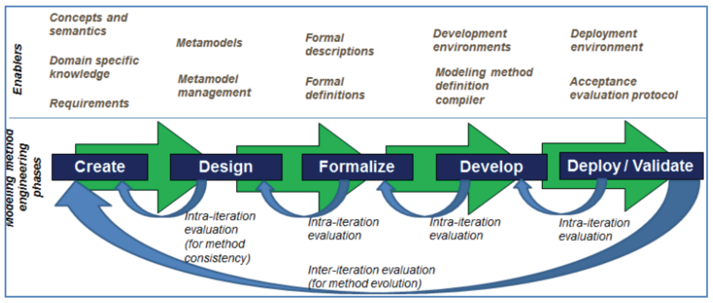

Enablers
Concepts and semantics
Domain specific knowledge
Requirements
Metamodels
Metamodel management
Formal descriptions
Formal definitions
Development environments
Modeling method definition compiler
Deployment environment
Acceptance evaluation protocol
Modelling method engineering phases
Create
Design
Formalize
Develop
Deploy / Validate
Intra-iteration evaluation (for method consistency)
Intra-iteration evaluation
Intra-iteration evaluation
Intra-iteration evaluation
Inter-iteration evaluation (for method evolution)

Figure 2.4: Agile Modelling Method Engineering Framework (a) and Conceptualisation Lifecycle (b) (Karagiannis, 2015)

thesis an integration framework for metamodels, and identified types of integration approaches from loose coupling to strong integration of metamodels and hybrid forms. The discussion is based on the modelling method framework presented in Fig. 2.3 and considers syntactical and semantical considerations. From the viewpoint of a distributed environment, such forms of integration are understood as a composition of multiple metamodels into a common structure. This guarantees operationalisation as interrelations are well-defined syntactically and semantically but the drawback is related to responsiveness to change and flexibility considerations. Assuming a new or change in requirements of stakeholders on a specific conceptual level, the overall structure is impacted and results in a re-engineering of the integrated metamodel.

Another example for the combination of multiple modelling levels into a common structure can be observed within the Multi-perspective Enterprise MOdeling (MEMO) approach discussed in (Frank, 2014; Bock and Frank, 2016). The approach builds on a common meta^2 level called "MEMO MML". All metamodels considered in MEMO are derived through instantiation and specialisation, covering the requirements on maintainability of the composed results and enable "self-referential systems which clearly contributes to more flexible systems and promote user empowerment" (Almeida et al., 2018, p. 24). Tooling of the metamodels in MEMO is an aspect that elevates efficiency and integrity.

From an industrial viewpoint, standardisation initiatives and the evolution of GPML have focused on the multi-level requirements. An example for this observation can be found in the work performed by committees such as Object Management Group (OMG) or the OpenGroup and their modelling approaches. Instead of focusing on a specific aspect or stakeholder, standards are defined and continuously extended to include any relevant aspect for a specific field. This is exemplified in the evolution of the BPMN notation in (Object Management Group (OMG), 2011a) from a modelling environment to support the "executable semantics of business process" towards an extended set of constructs for business analysts and implementation specialists in a common environment. A similar observation is applicable in the field of Enterprise Architecture Management (EAM). The standardisation done by the OpenGroup for ArchiMate (Open Group, 2019): ArchiMate spans from strategy to operational aspects of an organisation and aims to provide modelling constructs for any need of an involved stakeholder.

For the all the above cases on metamodel integration as *customisation* as a design concept is applied to adapt a metamodel element to changing requirements. These design patterns typically cover the abstraction of a specific domain into a conceptual representation. (Cho and J. Gray, 2011) discuss metamodel design patterns for the application in metamodel composition, where a new metamodel is created by re-using all constructs or parts of pre-existing metamodels, or metamodel inference, where metamodels are identified from existing model instances. The design patterns discussed consider base metamodel representation, typed relations and containment patterns.

Such patterns are relevant for the work performed, as the identification of commonalities is needed to provide alignment, matching and operations on an abstract level. In contrast to the integration and composition work discussed in literature, metamodels and their combination are understood as networked entities. This means that each metamodel required represents a node in the modelling environment for information systems design; these nodes are not strongly or loosely integrated with each other, but interrelation are defined as edges in a modelling ecosystem.

In (Selway et al., 2015) the authors discuss the implications of such considerations and propose a relationship framework for ecosystem modelling. The relationships between abstraction layers are extended by typing the instantiation and specialisation types towards extendibility, support for categories of external/secondary entities and subsetting specification links. This distinction is considered relevant for the identification of a common conceptual structures and interrelations between metamodels.

2.7 Knowledge Representation: Conceptual Structures

The foundations on knowledge representation, and specifically using conceptual structures, are relevant for the purpose of the artefact developed as a harmonisation layer for heterogenous modelling environments is required to enable global processing functionalities for analysis. In the following the theoretical foundation of knowledge representation is introduced and the understanding of conceptual graphs and its applicability in metamodel design and combination is discussed.

Foundation. Knowledge representation has been discussed thoroughly in (Davis et al., 1993) as the foundation for AI. It aims at representing information and knowledge about the SuS in a form, that machine-processing becomes feasible. The authors characterise the term according to its role as a) a "surrogate", that replaces or substitutes for reasoning and interference purposes, b) an "ontological commitment", even though abstraction is applied resulting in imperfection, it truncates a domain to purposeful elements, c) based on "fragmentation theory to support intelligent reasoning, supporting inference as a knowledge base, d) pragmatic computation support as efficiency are reflected in the representation and e) enable human expression and interpretation as a medium for communication.

Definition 2.11: Knowledge Representation
Knowledge representation is concerned with the design of a computer representation of a specific domain for the purpose of machine inference/human cognition in a problem space.

Conceptual structures, as a form of knowledge representation, have been prominently discussed in (Jackendoff, 1992) and (Sowa, 1984). The theoretical foundation is grounded as a logic-based technique. The term is derived from linguistics, where "a conceptual structure allows the representation of concepts in terms of a small number of conceptual primitives ('Conceptual Structure' 2019) that can be expressed mathematically as a conceptual graph."

Sowa defines in (Sowa, 2009) a conceptual graph as the "notation for representing the conceptual structures that relate language to perception and action", where the structure must exist and their properties are inferred. The notation has evolved over time, from semantic networks with the quantifiers of predicate calculus to the application of Peirce's Existential Graphs (EG). Conceptual graphs can be visually illustrated in the so-called "Conceptual Graph (CG) display form", the linear representation called Conceptual Graph Interchange Format (CGIF) has been standardised as one one of the three standard dialects for Common Logic (ISO/IEC 24707).

Definition 2.12: Conceptual Graph
A conceptual graph, as the formal representation of a conceptual structure, is a knowledge representation formalism that constructs the domain using conceptual primitives to support knowledge operations such as inference.

The applicability of the conceptual graphs for metamodelling and conceptual modelling in line with the motivational example presented in Fig. 1.4. It is assumed that the notion of metamodels as a conceptual graph (syntactical and semantical) could provide value during the design of a metamodel as capabilities of a specific metamodelling techniques are abstracted as a conceptual structure. Considering multi-level modelling environments, the representation of the contributing metamodels as conceptual graphs would support the definition of federated functionality spanning across the different layers identified.

Related Work. Metamodels as conceptual structures/graphs are introduced in (Gerbé, G. W. Mineau et al., 2001). The authors discuss a "CG based metamodeling framework for the modeling of information systems". The definition of CG structures to represent modelling constructs and its relational types have influences the development of the artefact. Using primitive relation types, a modelling construct is defined as a conceptual graph that can be mapped to first-order-logic.

- *csubt*: is used to represent sub-typing in the form or specialisation
- *sntx*: is applied to use a concept type on data level
- *def*: defining the concept, in relation to other constructs identified
- *rstrct*: two types of restriction graphs that impose constraints on the concept are distinguished: restrictions and rule graphs.

Fig. 2.5 shows the use of these relation types to construct an example of a modelling construct. The same approach is applicable for relation types. The mapping to first-order-logic results in knowledge/logic based operations that can be executed on these graphs. Basic operations on such graphs are introduced in (Chein and M.-L. Mugnier, 2008) for basic conceptual graphs and (Montes-Y-Gómez et al., 2001) for matching techniques. Both areas of knowledge operations are relevant to the work performed as they support the verification and validation of towards consistent modelling ecosystems and provide input for required matching approaches on metamodel level.

Related work on conceptual structures (in a rather abstract sense) for the design of information systems are discussed in (Wieringa, 2014, p. 73-78). The author provide the definition of a conceptual framework relevant for information systems and related decomposition techniques. An open question worked on, relates to the abstract conceptual structure required to represent metamodels - the proposal made by for business process on (Gerbé, Keller et al., 1998) and refined/abstracted to metamodels in (Gerbé, G. W. Mineau et al., 2001) needs to be validated for the purpose of this thesis.

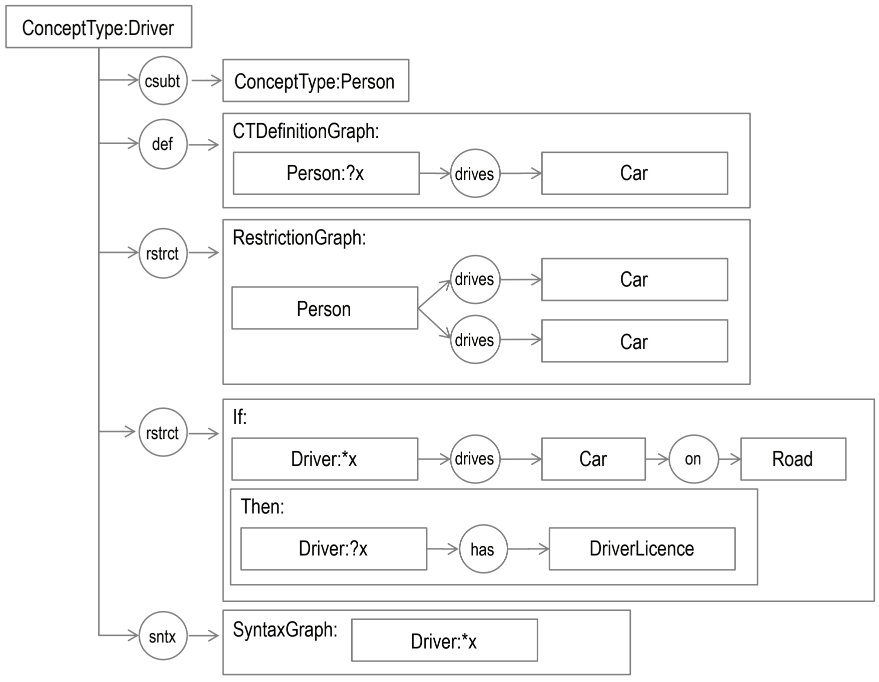


Figure 2.5: Example: Conceptual Graph (Conceptual Graph Display Format (CGDF)) of a Modelling Construct (Gerbé, G. W. Mineau et al., 2001)

2.8 Metamodels as Conceptual Graphs

The formal representation of metamodel has been analysed through an assessment of literature and existing metamodels within the OMiLAB context. As a result from this analysis, requirements have been established, initially published in (Utz, 2018b), further refined and concretised in (Karagiannis, Burzynski et al., 2019) and implemented as a graphical modelling notation using the Concept-Characteristic-Connector (CoChaCo) modelling environment as a foundation for the work on the federated metamodel. The following requirements have been considered:

– *Support Human Interpretation of Conceptual Structure* (derived from linguistics, from language to formal structure):
The conceptual structure needs to be readable and interpretable. This means that the co-design lifecycle is enabled in a way, that the metamodel engineer and the user are supported in their requirements elicit-

ation endeavour. From an external and knowledge sharing perspective, design artefacts are accessible for others to support re-use patterns.

- *Realise the Ability to transform/generate formal representation into arbitrary metamodel representations*:
  The conceptualisation results should be independent of existing formalisms but rather support the transformation/generation into a representation required for the operationalisation. This requirement also considers different levels of abstraction.

- Combination and Composition through *Formation Rules of Conceptual Structures*
  Metamodel operations are enabled by formation rules provided through conceptual structures and their formal representation as conceptual graphs.

- *Provide Similarity Matching based on Conceptual Structures*: The design concept enables detection of formal structures within artefacts and provide matching techniques to bind processing mechanisms and algorithms as needed. This requirement builds on the assumption, that a) the integration of processing mechanisms contributes to the model value consideration of the metamodel and b) implementation of mechanisms follows a binding approach. This means that existing algorithmic solutions can be discovered and dynamically attached to the metamodel.

- *Verification and Validation Approaches*: verification mechanisms are required to guarantee a smooth translation between design and operation phases. Validation mechanisms are established to evaluate the expressiveness and adequance of the result.

2.9 Federated Architecture in Information Systems Design

Federated architectures have been studies initially for database systems, with the requirement to establish an architecture where "components must maintain as much autonomy as possible, but the components must be able to achieve a reasonable degree of information sharing" (Heimbigner and McLeod, 1985). As an architectural pattern, interoperability and sharing of decentralised information, in the sense of distributed systems, is in scope. The term "federation" means minimised central authority, yet support for partial sharing and coordination among participating elements.

Considering a distributed modelling ecosystems, federated architectures are a candidate to overcome the limitations of integration and composition approaches introduced earlier. The participating modelling systems stay autonomous with their specific expressiveness but information sharing for the purpose of externalising integration links and holistic behaviour analysis can be realised.

Foundation. In information systems design, specifically enterprise architecture management, the concept of federated architectures has been established, as a principle to tackle interoperability and integration issues. (Goethals, 2011) distinguishes two types of frameworks: the classic enterprise architecture frameworks, that are established through centralisation, whereas federated frameworks link their building blocks by "a kind of umbrella". (Chen et al., 2008) have defined the baseline for this consideration by classifying three approaches to enable interoperability: a) integrated approach, where a common format for all models exist, b) unified approach, where a common format on meta-level exists and c) federated approach, where no common format is imposed by participating blocks on model, language or method level and interoperability aspects are considered "on-the-fly".

Definition 2.13: Federated Architecture
A federated architecture enforced autonomy of individual components of a system and, as a pattern, enables decentralised interoperability and information sharing. Individual capabilities of specific components are extended by global functionalities in a homogenous way, overcoming the limitations of distributed system design.

Related Work. Federation approaches have a long tradition in computer science in relation to information sharing and interoperability of application systems. In (Heimbigner and McLeod, 1985) the concept has been established for database system, defining the field of federated database systems. The architectural pattern has been extended and adapted to various application fields such as enterprise architecture management as discussed in the foundation part of this section. Related to this work, relevant literature with respect to concrete federation patterns is reviewed that provide instantiations and refinements for the concept. (Fernandez et al., 2003) discuss in their article three distinct design patterns for federated architecture that are considered for the research work presented:

1. *Federation* aims to have a "homomorphism between the organisation and system architecture"(Fernandez et al., 2003, p. 140), in broad terms, the information systems. Through homomorphism, the applications independence and data integrity is preserved, and access to each information of other systems happens in a controlled manner.

2. *Dependency separation* is concerned with the dependencies that exist between architectural components due to the way an organisation works. As such these dependencies are the result of the process architecture. A distinction is made between processing and informational dependencies whereas processing dependencies between different domain needs to be resolved so that inter-domain and inter-application dependencies are removed.

3. *Interface connection* establishes the domain-based communication mechanism. Communication channels can be resolved through direct interfaces within a domain or syndication, meaning via a coordinating node.

Complementing the view on federation, (Guychard et al., 2013) introduce observations that have motivated the development of "model-federation" for conceptual interoperability. The authors list four observations that a designer/analyst of any type of system faces: "a) multiplicity of views: any stakeholder of the system needs a set of custom views, expressing his viewpoint, to be sure that each of his concerns has been documented; b) multiplicity of concerns: the space of functional and non-functional concerns often does not fit with the organisation of views, as any view on the system might address several concerns; c) multiplicity of models: depending on the modelling intention, one has to choose the right formalisms and representations; and d) heterogeneity of modelling artefacts: those can come in the form of structured diagrams or text (requirements), drawings, spreadsheets, process-flow data, etc." (Guychard et al., 2013, p. 2). The authors introduce a separation of the modelling environment into three spaces: conceptual modelling, design and projection. A graphical representation of this separation is shown in Fig. 2.6.

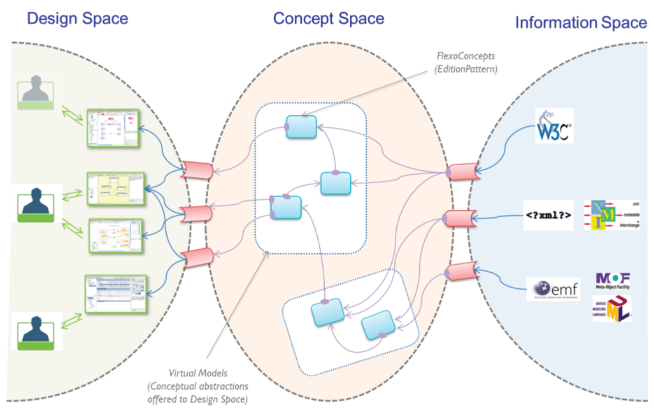


Figure 2.6: Federated Modelling Environment
(Guychard et al., 2013, p. 6)

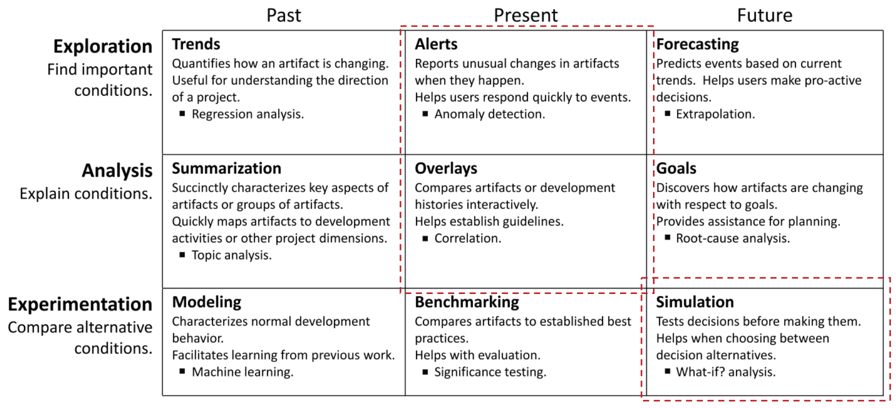
2.10 Software System Behaviour Analysis

Based on the assumption that understanding the behaviour of a modelling system contributes to the explorative needs in the design process, the terminology and foundation related to software system analysis is reviewed in the following, specifically focusing on behaviour and visualisation aspects. The precondition for this assessment is that the metamodel has been operationalised in a modelling tool.

Foundation. The analysis of software-based systems has been studied in literature from different viewpoints and perspectives. The editors of (Menzies and Zimmermann, 2013) provide an overview on the objectives of software analytics and position it as a field of high demand and impact from the perspective of knowledge management and the impact such analytical techniques have on an organisation. Their definition of the term "software analytics" is based on the work of (Buse and Zimmermann, 2012) and follows the objective to understand the question of information of "What happened", but also the question of insight "How did it happen and why" (Buse and Zimmermann, 2012, p. 987). The generalised definition that builds on quantitative and also qualitative views, discussed as "insights" derived from software analytics is presented below.

Definition 2.14: Software Analytics
Software analytics is analytics on software data for managers and software engineers with the aim of empowering software development individuals and teams to gain and share insight from their data to make better decisions. (Menzies and Zimmermann, 2013, p. 32)

The article of (Menzies and Zimmermann, 2013) presents the use cases for software analytics, ranging from performance, security and interaction capabilities of software artefacts to visualisation for comprehension. A possible categorisation of such scenarios is introduced in (Buse and Zimmermann, 2012) from a temporal perspective as a result of a quantitative and qualitative study performed with 110 experts from the field, graphically show in Fig. 2.7



	Past	Present	Future
Exploration Find important conditions.	**Trends** Quantifies how an artifact is changing. Useful for understanding the direction of a project. ▪ Regression analysis.	**Alerts** Reports unusual changes in artifacts when they happen. Helps users respond quickly to events. ▪ Anomaly detection.	**Forecasting** Predicts events based on current trends. Helps users make pro-active decisions. ▪ Extrapolation.
Analysis Explain conditions.	**Summarization** Succinctly characterizes key aspects of artifacts or groups of artifacts. Quickly maps artifacts to development activities or other project dimensions. ▪ Topic analysis.	**Overlays** Compares artifacts or development histories interactively. Helps establish guidelines. ▪ Correlation.	**Goals** Discovers how artifacts are changing with respect to goals. Provides assistance for planning. ▪ Root-cause analysis.
Experimentation Compare alternative conditions.	**Modeling** Characterizes normal development behavior. Facilitates learning from previous work. ▪ Machine learning.	**Benchmarking** Compares artifacts to established best practices. Helps with evaluation. ▪ Significance testing.	**Simulation** Tests decisions before making them. Helps when choosing between decision alternatives. ▪ What-if? analysis.

Figure 2.7: Software Analytics from a Temporal Perspective
(Buse and Zimmermann, 2012, p. 994)

For this thesis, the focus and scope is highlighted in Fig. 2.7. To understand a complex, distributed modelling ecosystem, the heterogenous software system and its behaviour needs to be acknowledged by involved stakeholders. Based on the assessment presented, analytical techniques for the behaviour of run-time components are investigated in, specifically focusing on the requirements towards visualisation.

This position for the thesis builds on the assumption, that externalising and visualising the interrelation between and within the participating components of the modelling system supports the exploration requirement.

Therefore a focus in the review of literature is put on behaviour analysis of software systems and its visualisation.

Definition 2.15: Software Visualisation
Software visualisation is defined as the visualisation of information related to software-based systems for analytical purposes. The term refers to the graphical display of any characteristic of a software system (Zhang et al., 1996, p. V) from design and architecture, programs, algorithms to run-time aspects such as behaviour and evolution following the broad definition of (Diehl, 2007, p. 3).

Related Work. Diehl's assessment of software visualisation categorisation is applicable for the visualisation requirements identified. In (Diehl, 2007, p. 3-4) an initial distinction is made between *Structure*, *Behaviour* and *Evolution*, whereas the structure category is concerned with static elements (e.g. source code and its architecture), behaviour consists of visualisation techniques for the behaviour and evolution considers the development/implementation process, applying a generalised visualisation method as a pipeline to all categories. The pipeline considers the phases of

1. *Collection*: concerned with the retrieval of data relevant for the visualisation, also termed "Data Acquisition",
2. *Analysis*: methods for data manipulation e.g. filtering, static program analysis, statistical methods,
3. *Visualisation*: transformation into an applicable graphical representation format.

The method introduced has been refined and applied to the development of model-based data services in (Utz and Woitsch, 2017), evaluated in the context of energy-efficient data streams and software systems.

For the behaviour analysis, a set of visualisation techniques are proposed in (Diehl, 2007, p. 79-125), coined Dynamic Program Visualisation, as a means to understand visually the behaviour of run-time components. Three types are distinguished: *accumulation*, considering the data produced/consumed by the running software over time, *spatial projection*, to plot the events and traces in a spatial representation and *animation* as a technique to replay/review the behaviour as it has occurred. For the visualisation, diagrammatic representation are deemed necessary, which are either derived/-generated or dynamically projected upon (e.g. dynamic software architecture visualisation). For the first case, the models are generated based on behaviour, for the second case static models of e.g. software architectures are augmented. A detailed analysis of the visualisation requirements for modelling ecosystems in presented in chapter 3, mapping the requirements of the artefact to existing visualisation approaches.

An application for the above concepts is discussed in (Rohr et al., 2008) for the visualisation of the behaviour of a multi-user, Java-based web application. The concept introduced, builds on the conceptual models of the software architecture and reconstructs the behaviour as a visualisation of operations in the system, as a composition of execution and message traces. Conceptual models used for the visualisation are UML sequence diagrams, Markov Chains, Component Dependency Diagrams, and Trace Timing Diagrams.

A conclusion that can be drawn from the discussion on software behaviour visualisation is that conceptual models (of software architecture and their interrelations) are currently applied, generated or augmented to support the visualisation of interaction behaviour.

Summary. Chapter 2 discussed the knowledge base and scientific rigor for the work performed in the research project.

Based on observations related to intelligence in information systems for industrial stakeholders, that argue for intelligent offerings and interactions, the background knowledge in the field and terminology is defined. Intelligence as a broad topic, especially in the era of continuous innovation, is regarded the key argument for the development of smart models. These smart models represent an information system on one hand, but are also applicable for digital innovation processes and the cognitive assessment of enterprise artefacts evolving or retiring within a service or products lifecycle. The motivation of this work relates to the need to have multiple, distributed modelling services that harmonise with each other, even though they are not directly related to each other.

Therefore the terms *intelligence* and *ecosystems* are considered as fundamental pillars towards the establishment of *smart models*, assuming that the intelligence functionalities can be related to semantic rich metamodels of each modelling service. Initially introduced in (Walch, 2019), the research objective is based in the observation that mechanisms are required to design and operationalise metamodels supporting smart, distributed ecosystems. Advances in the field of software behaviour analysis and federated system design are contributing to this argumentation, observable in the relevant literature on modelling, metamodelling, design and information systems management.

The observations have led to the formulation of hypothesis on adequate metamodels considering domain-specific aspects and the need for a representation formalism to capture this knowledge and make it processable, considering the distributed nature of current, and future, systems. The research question and objective set is structure around these hypothesis: which formalism can be developed to support intelligence and how can the design of adequate metamodels (including functionalities that span multiple nodes in a modelling ecosystem) be realised. The scope of this objectives relates to the analytical, intelligence features of the design, impacting use and assessment in an efficient manner.

The research questions and objectives are targeted as guiding elements within the development of the conceptual framework supporting the co-design of metamodels in a distributed manner.

Chapter 3

Design of Digital Intelligence Ecosystems

This chapter introduces the artefact developed as a method and functionality to design and specify metamodels of a modelling method to support digital intelligent ecosystems. The term "concept" in this chapter is understood as an architectural blueprint of the DeMoMa-(*) artefact, its relation and dependencies to the field of conceptual modelling/metamodelling and requirements identified for the intelligent behavioral analysis of information systems in digital transformation processes. In line with the observations and objective, comprehension is a relevant aspect for explorational and experimental interactions on any layer individually and holistically (system-wide). Therefore the scope of the artefact developed is articulated as comprehension of digital ecosystems as an analysis and design space.

This chapter is structured according to the research methodology, iteratively refining the artefact, considering relevance and rigor in its development: in an initial section, the problem space as introduced in Fig. 1.1 is refined and concretised. The OMiLAB's layered architecture is introduced structuring the work presented. Specific instantiations of metamodels for each layer and their interrelations guide the requirements elicitation process, with respect to the relevance of the work presented and an its evaluation of the artefact.

The DeMoMa-(*) concept considers the following building blocks, specified in detail in the sections below:

- Design a *Metamodel for Digital Intelligence Ecosystems*, using the underlying metamodel of each element and derived patterns for the capabilities. This means that a metamodel fragment's library is proposed for visualisation on type/pattern level, rather than on concrete instances. Exploration techniques are proposed following the federation approach within the ecosystem.

– Enable *Federation functionality* for metamodels on different levels of abstraction. This technique is proposed on syntactical/structural and semantic level, needed for the exploration of behaviour using conceptual graphs as a common knowledge representation formalism to describe metamodels and their instances.

– Realise knowledge operations for *Similarity Matching* between the metamodels for information systems design within a heterogenous environment and visualisation metamodels, considering the syntax of the metamodel and semantics encapsulated.

In the next section the problem space is introduced as a refinement of the problem statement introduced in section 1.3.

3.1 Problem Space: Digital Intelligence Environment

Considering the abstract representation of the problem statement graphically shown in Fig. 1.1, an instantiation of it done using results from the OMiLAB digital innovation laboratory to concretise challenges in focus.

The information system in the laboratory is designed for digital innovation and is structured in three layers, from a physical point of view (infrastructure/layout) as well as from a virtual, modelling software ecosystems perspective. The structure of the ecosystem is introduced in (Bork, Buchmann et al., 2019, p. 32) as a framework for "model-value co-creation", manifesting itself as the conceptualisation space for modelling methods on one hand and the experimentation/exploration environment as the digital product space on the other hand. Two assumptions have contributed to this development:

– Metamodel Availability: the conceptualisation space produces continuously metamodels, embedded within modelling methods following the Generic Modelling Method Framework defined in Fig. 2.3. These modelling methods are co-created by the community and establish metamodels on different levels of granularity, abstraction and expressiveness. As such they refine concepts from the application domain and (ideally) expose model-value functionality to the modeller.

– Digital Product Capabilities: the capabilities usable for the development of digital products and services can be abstracted and generalised. These capabilities are increasingly changing the model-value co-creation as the model processing becomes runtime-aware via Internet of Things (IOT) adaptors and the semantically rich representation in conceptual models. This observation has been studied in detail in (Walch, 2019), who developed a framework for knowledge externalisation to interact with physical devices.

The intelligent environment envisioned for the development of a digital product follows the three layer approach "Design-Model-Make (DeMoMa)", graphically shown in Fig. 3.1. Each of these layers is defined using a model-based approaches to support design and analysis. Interrelations between them realise the concept refinement of the domain as a decomposition, or representation of functional capabilities as an abstraction, even though these relations are not made explicit yet. Metamodels on each layer define the expressiveness of the modelling approach.

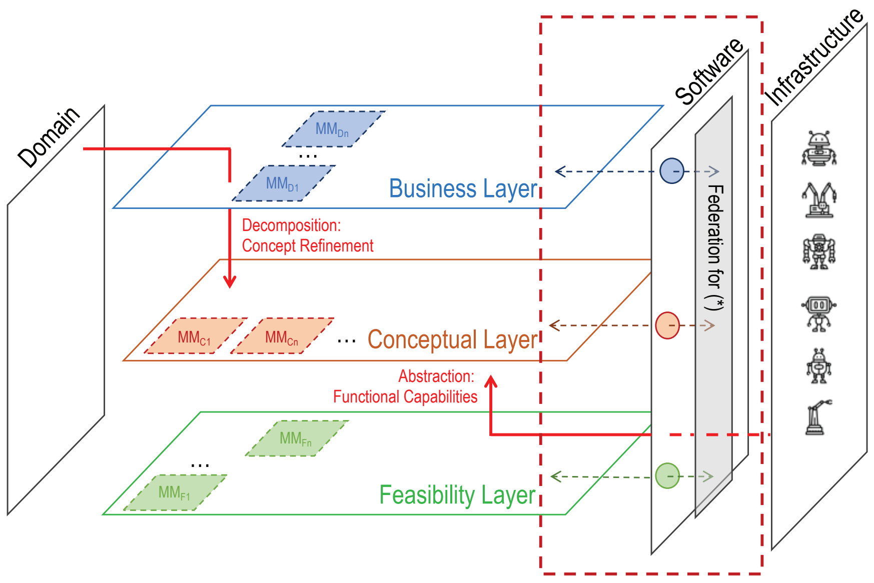


Figure 3.1: Digital Product Design Framework (adapted from (Bork, Buchmann et al., 2019, p. 44)

The layers for DeMoMa are distinguished by their purpose during the digital product development process:

– **De** *Design Layer* of Business Models: used for the definition of novel business model, user stories, value propositions and chains of application cases using design-thinking techniques. The layer is characterised by a high-level of abstraction to involve stakeholders, identify roles and responsibilities and their contribution to the value co-creation. Examples for techniques on this layer are the SAP Scenes approach introduced in (SAP User Experience Design Services, 2019), or the Business Model Canvas by (Osterwalder and Pigneur, 2010),

- **Mo** *Conceptual Model Layer*: as a refinement of high-level concepts from the design layer, the concepts identified are decomposed into conceptual models following its traditional understanding. Examples of modelling approaches on this layer are process models, organisational charts, business rules, data structures. The layer is concerned with a decomposition on one hand, but also acts as a mitigation layer towards the feasibility layer (matching with available functional capabilities),
- **Ma** *Feasibility Layer*: Metamodels on this layer are concerned with representing/abstracting the functional capabilities required for the design and validation/verification of a digital product. The feasibility layer is the experimentation and exploration space for the design and conceptual artefacts. The CPS becomes accessible for this purpose.

Each of the layer is well-understood in literature. Modelling techniques are available, researched and applied. In addition, integrated techniques are available, commonly described as multi-level modelling techniques. Assuming that time-consuming integration is not feasible or proves to be inefficient due to frequent changes and flexibility, openness and agility requirements, harmonisation is needed.

Federation for * is defined following the understanding of federated systems as granting each contributing component a maximum on autonomy and realising decentralised interoperability. For the thesis a harmonisation approach is introduced that builds on patterns identified within providing metamodels or fragments defined for a specific purpose and matching techniques as a harmonisation instrument. The * is defined as the purpose of digital intelligence within the ecosystem.

The objective of the thesis is to instantiate the "*" with comprehension as software analysis and visualisation. Any interaction of the operational, software-based metamodels in the digital product design space are subject to visualisation to enhance comprehension of design/analysis outputs. Nevertheless, the concept developed is generalisable, as the purpose can be re-defined, resulting in fragments required by the purpose that are re-matched.

For the motivational cases, that are used as high-level case description, a common framework of modelling approaches is assumed for simplification. Fig. 3.2 shows the selection graphically and provides an outlook on high-level requirement towards the federated architecture and its needs. It is assumed that the selection of modelling approaches for the different layers depend on the industrial and application domain. A single approach per layer, as shown in the figure is not realistic and multiplicity needs to be considered in the solution design.

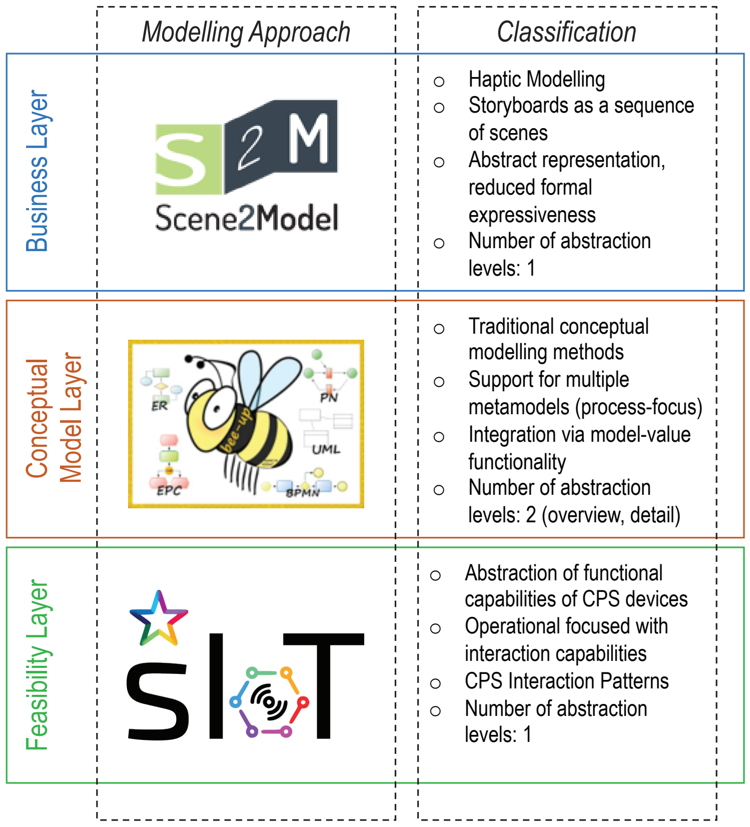

Modelling Approach
Classification
Business Layer
Scene2Model
Haptic Modelling
Storyboards as a sequence of scenes
Abstract representation, reduced formal expressiveness
Number of abstraction levels: 1
Conceptual Model Layer
ER
PN
UML
EPC
BPMN
Traditional conceptual modelling methods
Support for multiple metamodels (process-focus)
Integration via model-value functionality
Number of abstraction levels: 2 (overview, detail)
Feasibility Layer
SIoT
Abstraction of functional capabilities of CPS devices
Operational focused with interaction capabilities
CPS Interaction Patterns
Number of abstraction levels: 1

Figure 3.2: Motivational Cases: Modelling Approaches (derived from (Miron et al., 2018), (Karagiannis, Buchmann et al., 2016; Muck et al., 2018) and (Walch, 2019))

3.1.1 Motivational Case: Smart On-Demand Mobility (Approach: Top-Down, Decomposition)

The motivational example for the development of the concept instantiates the layers and is described using the OMiLAB setup. On the design layer, the haptic, modelling approach Scene2Model is used, the conceptual modelling layer consists of traditional modelling techniques, amalgamated in the Bee-Up metamodels and on the feasibility layer, a modelling technique as a subset of s*IoT modelling method (Walch, 2019) is applied.

As a scenario, demonstrating the purpose and relevance of the artefact, a case has been selected in the field of smart on-demand mobility. The scenario is within the field of a novel product/service solution of car-sharing provider, transforming the business model of the ecosystem by digitising establishing a smart offering.

Case Description. Currently, car sharing providers operate their fleet on a global scale and provide a mobility solution that replaces ownership with utility. This is based on the observation that it is not important anymore to own a car but use mobility according to personal preferences, availability and location-specific setting. The current offering puts the user's knowledge in the foreground: depending on the current location, available cars are offered and the user makes the choice to select the car based on the destination, the required size and space (in case of transportation), environmental conditions and other preferences. To provide this offering, the physical object (the car) has been digitised e.g. the lock/unlock mechanism is available online, payment processes are automised and dynamic pricing models are implemented. Nevertheless, the knowledge of the user is essential before and during usage.

In a future business model for smart on-demand mobility, the information system supports the potential driver in this decision process. Cars (of different providers) compete for customers in an intelligent manner. This requires a fundamental change in the way the intelligent offering is defined: the potential user signals demands to the system using a mobile, location-aware device, providing the destination and transportation relevant parameters. Based on the past service usage, the user profile and preferences are known and provided to the "cloud" of cars. Each car then individually assesses the conditions applicable to provide the ride (from an economical, environmental, location point of view). Alternatives are offered and ranked and the user can select the ride most appropriate. Considering advances in autonomous driving, one can imagine that there is no need in the future to find the car, open it and perform the ride, but the best-rated (and selected) car arranges for a rendezvous, picks up the client and performs the mobility service. Automated processes in the back-office run after-sales activities such as payment, satisfaction analysis and preference profile updates.

Considering this innovative service idea, the question remains how such an idea can be designed on a business model level, constructed from an organisational and processes perspective and how feasibility be evaluated. As the new business model drives the change, a top-down approach for the realisation of feasibility is suggested, described in the following. The steps and results along a heterogenous method chain are visually shown in Fig. 3.3, including the iterations cycles in design, concept and feasibility assessment.

For the feasibility assessment of the innovative idea for on-demand mobility, a top-down approach in the sense of decomposition of concepts is applied. Design decision on business model level impact the realisation and required capabilities on conceptual and feasibility level. This means that the decisions taken strategically, are transformed into operational aspects (e.g. processes, organisational structure) and further transformed to derive required functionalities on physical level.

1. Design of the business model (*Business Layer*): The innovative idea is transformed into a business model building on the expertise and creativity of various stakeholders within the team. Design thinking methods, as introduced, are applied as they support "communication and interaction between stakeholders" (Elmansy, 2018), which is deemed key to achieve meaningful results. Facilitation in the form of moderation and tooling is essential. For the creation of the business model in the case described above the SAP Scenes approach (SAP User Experience Design Services, 2019) is selected. Using haptic elements, the mobility solution is created in a workshop situation that focuses on the actors involved (driver, provider, car) and their relations among each other. The individual scenes are composed into a storyboard and represent the way the new solution is intended to work, from triggering the need of mobility, the selection and proposal process as a knowledge-intense task performed by the car, to the operation and after-services processes. The questions addressed in such a setting are how an intelligent offering can be realised, which actors are involved and how value

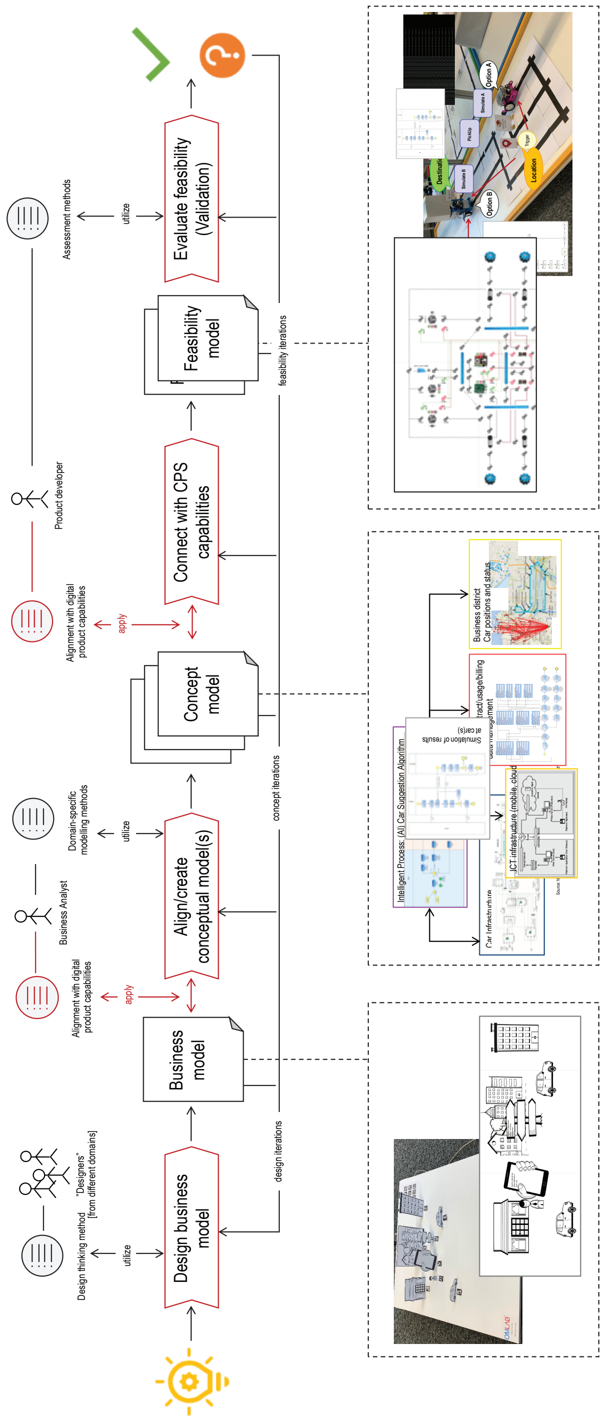

Design business model
Business model
Align/create conceptual model(s)
Concept model
Connect with CPS capabilities
Feasibility model
Evaluate feasibility (Validation)
utilize
apply
utilize
apply
utilize
Design thinking method
"Designers" [from different domains]
Alignment with digital product capabilities
Business Analyst
Domain-specific modelling methods
Alignment with digital product capabilities
Product developer
Assessment methods
design iterations
concept iterations
feasibility iterations
Intelligent Process, (AI) Car Suggestion Algorithm
Simulation of results at car(s)
ICT infrastructure (mobile, cloud)
Car infrastructure
Business district Car positions and status
Option A
Option B
Location
Destination

Figure 3.3: From Business Model to Feasibility Study (Top-Down): Smart On-Demand Mobility

is created for the end-user as well as the providing enterprise.

The requirement in the design process to share and include the expertise of experts not physically present is covered through tool support; the Scene2Model implementation as introduced in (Miron et al., 2018; Muck et al., 2018) is applied. The tool enables the transformation of the haptic design results into diagrammatic models using QR codes and a camera setup for recognition. Design iterations result in am abstract business model definition that is agreed upon all participants. The business model is considered a representation of actors, their interrelation to establish a value-proposition, the pricing/cost structure and channels for distribution.

2. Alignment with/creation of conceptual models (*Conceptual Layer*): In a next step, the business model is aligned with conceptual models available. These models are either already available and operational within the enterprise and its ecosystem, retrieved and specialised from reference models in the industry or created from scratch by business analysts. Adequate domain-specific modelling methods are applied for this purpose, decomposing the aspects of the business model. A challenge in this phase is is the alignment between the design results and conceptual models as the expressiveness on both levels differs due abstraction differences. Therefore, the design artefact needs to be semantically lifted to become more expressive, similarly done for the conceptual model repositories to allow for matching and alignment techniques.

   For the case presented, the conceptual level is established using traditional conceptual modelling approaches for processes, data and organisation networks. The business analyst matches the business model with available resources in a semi-automatic way. Intelligent mechanisms assess the semantic distance, create semantic enrichments based on the model content and underlying metamodel and propose interrelations. Design effectiveness of the solution can be assessed early on in the process through model-value functionalities, e.g. simulation of alternative solutions for the end-to-end rental process using quantitative facets of the conceptual models. These functionalities are provided for the case using the Bee-Up implementation: the matching mechanisms are fed with the RDF representation of the models created, considering the underlying metamodel as the grammar, model-value functionalities are available for different process-based metamodels.

3. Connect with CPS functionalities (*Feasibility Layer*): Early feasibility assessment of the innovative idea is essential for refinement iterations of the design and conceptual results. The conceptual models are extended in this phase with operation calls to the CPS environment. Within the laboratory setting, the feasibility space provides the required devices to test the idea. These devices are understood as the prototype setting for an explorative evaluation, answering questions such as which capabilities need to be available for the system to run or what communication and processing protocols are adequate. The exploration does not aim to verify a real-life instance but is defined with a laboratory, prototyping environment. For the case, the capabilities needed on the physical device are tested, e.g. context-awareness of the car, processing capabilities needed to simulate pricing and decision making aspects, or rule-based preference assessments.

   An input for the phase are abstract representation of the different capabilities that can be embedded in the conceptual models and triggered. The CPS devices therefore become aware of the conceptual definition. Similar as for the transition between business and concept level, the gap between available capabilities and provided interfaces result in requirements from the conceptual layer that need to be bridged. Missing elements are made available by adapting or customising the implementation on the device, exposing new functional capabilities as interfaces and providing them in an abstract manner to the conceptual layer. An important consideration is that the models on this layer are not aware of the actual device (type, technology). This means that the abstract view of a smart car is a composition result of its capabilities but can be exchanged dynamically (as long as the abstract capabilities can be mapped).

4. Explore viability on *Feasibility Layer*: Evaluation of the feasibility is needed as an immediate feedback to the design and conceptual layers. Assessment methods provide results back and trigger new iterations. The objective is to have the possibility to observe how design decision on the above layers change the operational level.

In the case of smart on-demand mobility, the business model design drives the evolution of the conceptual layer and realisation of digital capabilities. Intelligence in the translation and transformation between the layers is required and needs to be become transparent to provide an explanation of complex system dependencies and their impact resulting from high-level design decisions.

As a second case, the application of the inverse methodological approach can also be considered. This means that the digitalisation of a physical device triggers and drives the process.

3.1.2 Motivational Case: Smart Battery Management (Approach: Bottom-Up, Abstraction)

Case Description. In this motivational case the innovation is triggered on physical object level. Traditionally, batteries power many IOT devices and in case of lower voltage, need to be replaced periodically to make sure the device is operational and can be used. An example for this observation relates to micro-processing devices such as the Raspberry Pi that needs a stable voltage stream in order to operate properly. When connected to a power source, power outages are prevented but limit the location independent use of a device. Battery management is therefore essential to overcome the limitation of a static location context. This results in the need to add a functional capability to the battery to monitor the its levels and expose them, as input to react in a timely manner and reduce downtimes. Alternative solutions on conceptual level result in differing business models that are assessed and studied for viability.

For the feasibility assessment of the product innovation, a bottom-up approach in the sense of abstraction of physical capabilities and composition of business model scenarios is applied. Design decision on product engineering level and their abstraction impact the realisation of the concept and business layer models.

1. Realise a product prototype on the *Feasibility Layer*: product engineers develop a prototype that demonstrates smart capabilities of a battery design. They consider sensing and connectivity issues in their engineering task as a foundation for smart service offerings. The prototype is designed and realised in laboratory conditions to study the technical requirements to be covered: data streams, interfaces protocols and technologies to be applied.
2. Use the functional capabilities on *Conceptual Layer* to define oper-

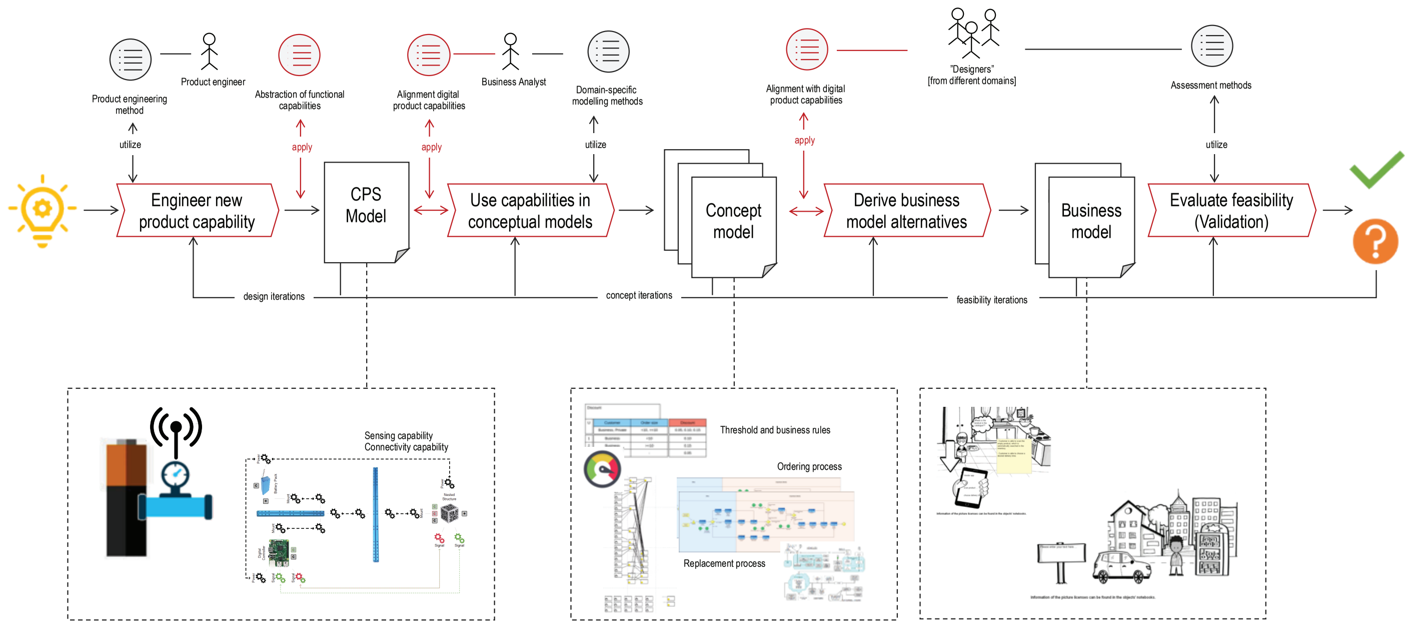

Product engineering method
Product engineer
Abstraction of functional capabilities
Alignment digital product capabilities
Business Analyst
Domain-specific modelling methods
Alignment with digital product capabilities
"Designers" [from different domains]
Assessment methods
utilize
apply
apply
utilize
apply
utilize
Engineer new product capability
CPS Model
Use capabilities in conceptual models
Concept model
Derive business model alternatives
Business model
Evaluate feasibility (Validation)
design iterations
concept iterations
feasibility iterations
Sensing capability
Connectivity capability
Threshold and business rules
Ordering process
Replacement process

Figure 3.4: From Product Innovation to Business Model Feasibility (Bottom-Up): Smart Battery Management

ational aspects: the analytical work is performed on the *conceptual layer*. The processes to retrieve data, handle exceptions and integrate operational processes are covered. Thresholds and triggering mechanisms are evaluated in relation to the CPS models established. Data services are hierarchically connected to the sensor interfaces and retrieve the current battery status e.g. as voltage levels to determine the health status. Processes describe the operational aspects and business rules are used to define thresholds. The product feasibility is validated using static and dynamic analysis techniques for the monitoring, escalation and ordering processes.

The abstraction approach applied on the lower level is extended and formally aligned with available or new models of the enterprise. This targets systems, data and behavioural aspects.

3. Assess business model scenarios on the *Business Layer*: the stakeholders from product engineering, operations and strategy jointly assess different options to establish a novel business model based on the new product functionality. Service offerings and value propositions can be compared and analysed, e.g. replacement and ordering process triggers based on user preference; just-in-time options for the replacement and risk assessments.

   In this case, the conceptual level limits (or filters) the expressiveness of the design thinking approach: only aspects considered on conceptual level can be designed and assessed. Identifying a new aspect not considered results in an update on conceptual level.

3.1.3 Requirements for Digital Intelligence Ecosystems

From the case and the description of the design, model and make steps above, we can derive requirement towards the harmonisation of such a system following federation concepts as introduced earlier. The literature review in chapter 2 established the context to related work and background knowledge.

Model Alignment: Independent whether a bottom-up or top-down approach is followed, transitions between layers, but also within, differ from case-to-case. Support is required to align the layers with each other (in case pre-existing artefacts exist) or hand-over of the context of one layer to the other. As the modelling approaches are independent of each other, the alignment is considered a transformation on one hand, respecting the syntax and semantic of source and target environment, on the other hand, an intelligent mechanism that retrieves information, prepares proposals for the modeller and defines the interrelations on a federated level. The metamodels are projected upon the federated level to achieve this requirement.

Traceability: Traceability of the design and analysis process is essential. As a process, the artefacts evolve in a co-creation manner; traceability links between valid/approved versions support the understanding of the information system on a holistic level. Traceability links are established on horizontal level, in between models of the same abstraction and granularity and vertically, in between layers. They can be understood as cause-and-effect links between model artefacts. Changes in the models are reflected in the metamodel as a response to changing environment and context settings.

Visualisation of Interactions: Visualisation is considered an outcome of the above requirement, that aims at making the operational system transparent to the user. Mechanisms for retrieval and matching, system and device interface calls in a complex experimental setup should become visible and understandable. The requirement is concerned with collecting all types of software-based interactions, visualising their impact on the underlying technology. Interactions are broadly understood as any type of human-to-machine or machine-to-machine operations (in various types from synchronous, asynchronous, void calls to push and pull interfaces).

Information Integration: is concerned with the distribution of information (as a model artefact or the result of a model processing step) within the ecosystem. The integration builds on traceability relations and supports the interchange of information achieved. An example for a this requirement is related to simulation outcomes, as a result of design effectiveness consideration on the model layer, that are distributed to the business level to provide a quantitative assessment of design alternatives.

3.2 Metamodel Federation for Digital Intelligence Ecosystems

The above requirements result in the development of a concept applicable to the abstract definition of intelligent ecosystems. The principle of federated architectures has been applied in the development that aims to create an artefact that is minimal intrusive, elevates individual functionalities of participants with operations on a federal level. This approach is considered as harmonisation, as underlying artefacts become interoperable, but stay at the same time autonomous. The two building blocks identified are a common metamodel that is defined as a projection of the individual metamodels, and intelligent operations that can be realised on this metamodel that provide the functionalities required. Fig. 5.7 shows the two elements in a graphical way.

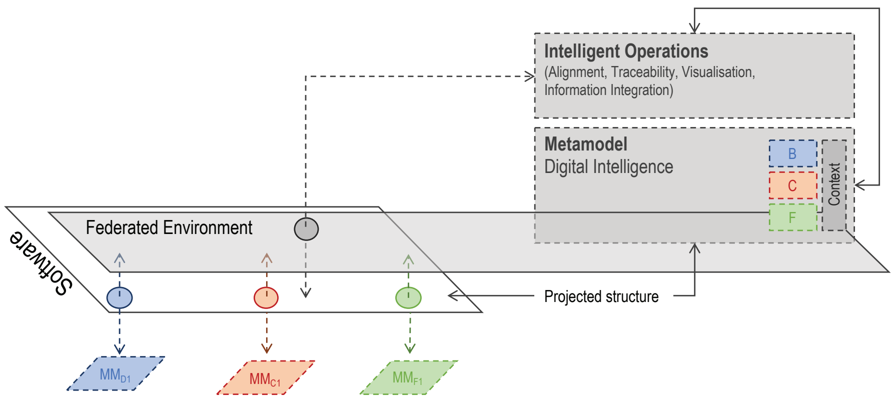


Figure 3.5: Intelligent Operations in a Federated Environment

The federated metamodel builds on the three distinct layers introduced earlier, and a common element for all layers covering the context of the ecosystem. It aims to provide abstract constructs that are used as a projection of an arbitrary metamodel (and consequently its model artefacts) of each layer. Contextual elements define the semantic system boundaries that is applicable for all layers. In order to describe the functionalities of each layer element, the CoChaCo design approach is applied as introduced in (Karagiannis, Burzynski et al., 2019) that supports the requirements engineering of metamodels.

In the following the CoChaCo Approach is introduced, including the formal extensions required to represent metamodels as conceptual graphs. This extension is applicable on on hand to have a common description format for any metamodel, and provide mechanisms for alignment of individual artefacts towards to common structure.

3.3 CoChaCo Metamodels as Conceptual Structures

Considering the related work in the area of metamodel design on one hand and conceptual structures on the other, the CoChaCo approach has been realised and extended to enable processing functionality based on a formal representation that results in generalisable functionality of similarity matching as a projection. Conceptual graphs have been identified in the related work section as the theoretical foundation for the formulation, building on the results of (Gerbé, G. W. Mineau et al., 2001). The proposal in (Gerbé, Keller et al., 1998) guides the specification of the common grammar elements.

The functional capabilities of conceptual graphs, as introduced in (Chein and M.-L. Mugnier, 2008) and defined as formation rules, contribute to the identification of graph-based patterns, their composition, translation between representation and verification rules for model processing functionalities. In a first step, the formal representation in CoChaCo is discussed as the structural element, functionalities on the knowledge representation are introduced, operating upon this structure.

3.3.1 CoChaCo Metamodel: Concept-Characteristics-Connector

The CoChaCo approach based on the requirements above is introduced in this section. The specification of CoChaCo builds on the assumption that metamodels can be designed and represented using conceptual modelling itself, leveraging on the intuitive interaction capabilities and adaptable views for the engineers and users. An early version of the approach has been discussed and evaluated in (Utz, 2019a), initially defined in (Karagiannis, Burzynski et al., 2019). The representation of a metamodel as a conceptual graph builds upon the notational aspects of CoChaCo, extended by the conceptual graph specification introduced in (Gerbé, G. W. Mineau et al., 2001). ConceptTypes and RelationTypes are precisely defined using nested conceptual graphs for the specification of its logical semantic.

The base version of CoChaCo aims to provide a design space for metamodelling. This design aspect can be observed by the weak definitions of concepts and relations proposed, especially within the definition of the semantic of relation types: any design can be created and instances defined can be connected with each other freely. This semantic weakness aims to overcome limitations during the design process in the spirit of creativity - anything can be designed without restrictions from an operational perspective. A drawback of this extended creativity is that the machine interpretation of the models is only partially supported and requires semantic alignment/lifting during processing. Mechanisms and algorithms operating on the structure reflect the interpretation required.

As an overview, the design capabilities of CoChaCo are presented in Fig. 5.7.

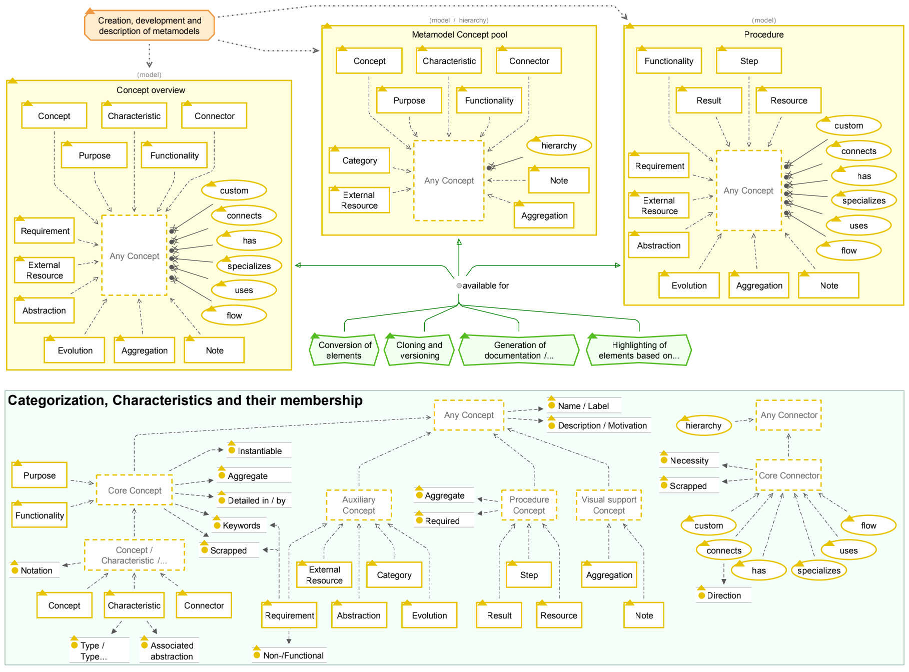


Figure 3.6: CoChaCo Metamodel
(Karagiannis, Burzynski et al., 2019, p. 205)

Using CoChaCo, a metamodel is defined as a combination of the three elements: concepts, defined as modelling types, characteristics as properties of these concepts, and connector, defining possible relationships between concepts. The elements are related with each other using relation types (connects, has, specialises, uses, flow). Support elements for modelling in CoChaCo are disregarded in the analysis of the approach (e.g. note and aggregation as visual aids, evolution and abstraction elements to define temporal dependencies between specification outcomes).

3.3.2 Metamodelling using CoChaCo: Grammar

In the following the grammar of the CoChaCo metamodel is described, following the MM-DSL logic introduced in (Visic, H. G. Fill et al., 2015; Višić, 2016), refined and adapted for the purpose of representing arbitrary metamodels in a common format, independent of technological platforms. From a syntactical point of view, Extended Backus–Naur form (EBNF) form has been selected, visually represented as railroad diagrams in appendix A for convenient readability. The meaning of each construct is verbally described, and the notation is introduced for each language element in table format. The constructs listed and graphically presented in Table 3.2 are required to design a metamodel, extended by behavioural elements required to add processing functionality to a metamodel, and relations/association between structural and behaviour elements defined in Table 3.3.

$$\begin{aligned} \langle\text{metamodel}\rangle \quad ::= \quad & \langle\text{construct}\rangle \texttt{ +} \\ & \langle\text{relation}\rangle \texttt{ +} \end{aligned} \tag{3.1}$$

$$\begin{aligned} \langle\text{construct}\rangle \quad ::= \quad & \langle\text{structure}\rangle \ | \\ & \langle\text{behaviour}\rangle \end{aligned} \tag{3.2}$$

$$\begin{aligned} \langle\text{structure}\rangle \quad ::= \quad & \langle\text{concept}\rangle \ | \\ & \langle\text{characteristic}\rangle \ | \\ & \langle\text{connector}\rangle \end{aligned} \tag{3.3}$$

$$\begin{aligned} \langle\text{behaviour-construct}\rangle \quad ::= \quad & \langle\text{purpose}\rangle \ | \\ & \langle\text{functionality}\rangle \ | \\ & \langle\text{stakeholder}\rangle \end{aligned}$$

$$\begin{aligned} \langle\text{relation}\rangle \quad ::= \quad & \langle\text{connects}\rangle \ \mid \\ & \langle\text{has}\rangle \ \mid \\ & \langle\text{specialises}\rangle \ \mid \\ & \langle\text{uses}\rangle \ \mid \\ & \langle\text{custom}\rangle \ \mid \\ & \langle\text{flows}\rangle \end{aligned} \tag{3.4}$$

Table 3.1: CoChaCo Grammar: Metamodel

Following the grammar definition of above, a metamodel is defined as a set of at least one constructs and one relations (3.1), whereas a construct is considered to be either structural or behavioural (3.2). Structural constructs relate to the metamodel elements and are typed as concepts, characteristics or connectors, behavioural constructs define the processing logic as purpose, functionality and stakeholders.

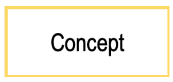
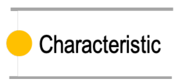
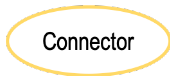
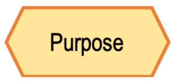
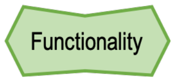
Construct	**Description**
Structural Constructs	
	Depicts modelling construct on an arbitrary level of abstraction. This could be used to represent a graphical symbol of the language or an abstract element for functionality and model processing.
	Any type of characteristic of a concept. Characteristics can be on syntax level, defining attributes and properties, behaviour on the dynamic characteristics or notational on the graphical aspects of a concept.
	Describes a relationship in the metamodel between concepts as a graphical connector, a hyperlink or containment and composition dependencies.
Behavioural Constructs	
	Identifies the purpose of a specific mechanism or algorithm in relation with the modelling procedure and results to be perceived. The purpose includes considerations on steps and resources required for a specific processing approach.
	Relates concepts and characteristics with processing functionalities. This construct is used to identify functionalities in relation to abstract and concrete metamodel structures.

Table 3.2: CoChaCo: Notation of Structural and Behavioural Constructs

Generic relations (3.4) establish typed associations between the constructs of any flavour and have exactly two endpoints (start and end for directed relations) assigned. In the following, in the decomposition of the grammar, we focus on the structural elements (3.3) as they represent the metamodel definition. The abstract *structural-construct* is refined in definition 3.5.

$$
\begin{aligned}
\langle\text{structure}\rangle \quad ::= \quad & (\langle\text{concept}\rangle \ | \\
& \langle\text{characteristic}\rangle \ | \\
& \langle\text{connector}\rangle) \\
& \langle\text{isInstantiable}\rangle \\
& \texttt{has}\langle\text{name}\rangle \\
& \texttt{has}\langle\text{description}\rangle \\
& (\texttt{aggregates}\langle\text{structure}\rangle)\ * \\
& (\texttt{has}\langle\text{characteristic}\rangle)\ *
\end{aligned}
\tag{3.5}
$$

Structural constructs define the language elements and their characteristics. The defining properties are their name and description. Aggregation allow a composition of structural elements. Abstract structural elements are applicable for for any type and specify constructs that are not directly instantiable, but embed semantics on higher levels of abstraction.

A structural construct is defined as being either a concept, characteristic or connector. Common properties are a descriptive name, further detailed using a free-text description (both name and description could be considered as instantiated characteristics, but are made explicit as a result of the specification characteristic of CoChaCo), whether the construct can be instantiated (resulting in a concrete modelling class in the metamodel, or alternatively an abstract class for further specialisation), whether it is composed and aggregated of other constructs, and the domain-specific characteristics that refine and specify each construct. Each of the sub-types can be used for further specialisation in a hierarchical manner (see *specialises* relation for each type). This means that constructs can be structured hierarchically, whereas hierarchy is not limited to a strict tree structure.

$$
\begin{aligned}
\langle\text{concept}\rangle \quad ::= \quad & (\texttt{specialises}\langle\text{concept}\rangle)? \\
& (\texttt{isEndpoint}\langle\text{connector}\rangle)\ * \\
& \texttt{has}\langle\text{notation}\rangle
\end{aligned}
\tag{3.6}
$$

The concept is the core element of any language. A concept of a modelling language is considered within the environment it exists in (defined by its connectors and where it acts as an endpoint). Graphical elements define a notation as the concrete syntax.

A concept is defined by its connector relations (in 3.6) defined as endpoints, which are derived from the *(*hasFromEndpoint) and *(*hasToEndpoint) in 3.7. A similar technique is applicable for characteristics that refine the structure, notation and behaviour of a construct. The *belongsTo* relation is derived from the assignment of characteristics to structural-constructs as an inverse view on the *has* relation defined for any structural construct.

$$\begin{aligned} \langle\text{connector}\rangle \quad ::= \quad & (\texttt{specialises}\langle\text{connector}\rangle)\ \texttt{?} \\ & (\texttt{hasFromEndpoint}\langle\text{concept}\rangle)\ \texttt{+} \\ & (\texttt{hasToEndpoint}\langle\text{concept}\rangle)\ \texttt{+} \\ & \texttt{has}\langle\text{notation}\rangle \end{aligned} \tag{3.7}$$

A connector defines the relational dependencies of concepts. The connectors are typed and establish the relational semantics for each concept within its context.

$$\begin{aligned} \langle\text{characteristic}\rangle \quad ::= \quad & (\texttt{specialises}\langle\text{characteristic}\rangle)\ \texttt{?} \\ & (\texttt{belongsTo}\langle\text{concept}\rangle \mid \langle\text{characteristic}\rangle \\ & \langle\text{connector}\rangle))\ \texttt{+} \end{aligned} \tag{3.8}$$

A characteristic defines the syntax and semantic of a concept or connector. In the narrow sense characteristics establish the properties (from a structural perspective) and behavioural aspects of the object.

$$\begin{aligned} \langle\text{isInstantiable}\rangle \quad &::= \quad \texttt{true} \mid \texttt{false} \\ \langle\text{name}\rangle \quad &::= \quad \texttt{STRING} \\ \langle\text{description}\rangle \quad &::= \quad \texttt{LONGSTRING} \\ \langle\text{notation}\rangle \quad &::= \quad \texttt{REPRESENTATION} \end{aligned}$$

Types are established as general-purpose characteristics of any concept, connector or characteristic.

Relations in CoChaCo represent association between the constructs and establish the abstract syntax of the metamodel designed. These relations are defined in Table 3.3.2. Six different types of relations are supported by the approach, defined on grammar level below. The relation types are generic in a sense that their use is proposed, but not restricted. A metamodel designer can interpret and apply them flexibly. This freedom of interpretation can be observed in the grammar definition (to and from endpoints are defined on construct level) as well as in the definition of each type.

$$\begin{aligned}\langle\text{connects}\rangle \quad ::= \quad & \texttt{from}\langle\text{construct}\rangle \\ & \texttt{to}\langle\text{construct}\rangle \\ & \texttt{direction(none} \mid \texttt{to} \mid \texttt{from} \mid \texttt{both)} \\ & \texttt{necessity(none} \mid \texttt{unspecified} \mid \texttt{optional} \mid \\ & \texttt{mandatory} \mid \texttt{mandatory in certain cases)}\end{aligned} \tag{3.9}$$

For the "connects" relation, the source is a typically of type Connector and the target a Concept or Connector, even though it is not restricted. It specifies that the source connects something specific (the target) to it, itself or anything else (all the connects relations of the source have to be considered).

$$\begin{aligned}\langle\text{custom}\rangle \quad ::= \quad & \texttt{from}\langle\text{construct}\rangle \\ & \texttt{to}\langle\text{construct}\rangle \\ & \texttt{direction(none} \mid \texttt{to} \mid \texttt{from} \mid \texttt{both)} \\ & \texttt{necessity(none} \mid \texttt{unspecified} \mid \texttt{optional} \mid \\ & \texttt{mandatory} \mid \texttt{mandatory in certain cases)}\end{aligned} \tag{3.10}$$

The "custom" relation has no restriction and meaning is established individually by the modeller.

$$
\begin{aligned}
\langle\text{specialises}\rangle \quad ::= \quad & \texttt{from}\langle\text{construct}\rangle \\
& \texttt{to}\langle\text{construct}\rangle \\
& \texttt{necessity(none | unspecified | optional |} \\
& \texttt{mandatory | mandatory in certain cases)}
\end{aligned} \tag{3.11}
$$

The "specialises" relation defines a relation between two elements of the same type, but also from a characteristic to a Concept or Connector; it specifies that something (the source) is the specialisation of something else (the target) [think sub-type / inheritance] or that it specialises something else (e.g. using a characteristic to handle sub-types). The inverse relation "hierarchy" structures the concepts/characteristics/connectors on a type level.

$$
\begin{aligned}
\langle\text{flows}\rangle \quad ::= \quad & \texttt{from}\langle\text{construct}\rangle \\
& \texttt{to}\langle\text{construct}\rangle
\end{aligned} \tag{3.12}
$$

The "flows" relation can be defined between step, resource and/or result; it specifies that something flows from one (the source) to another (the target), but what exactly depends on the context (e.g. when the source is a resource and the target a step then the Resource most likely flows into the step as input).

$$
\begin{aligned}
\langle\text{has}\rangle \quad ::= \quad & \texttt{from}\langle\text{construct}\rangle \\
& \texttt{to}\langle\text{construct}\rangle \\
& \texttt{necessity(none | unspecified | optional |} \\
& \texttt{mandatory | mandatory in certain cases)}
\end{aligned} \tag{3.13}
$$

The "has" relation" is typically defined between two structure constructs, two purposes, two functionalities or from any other to a requirement; It specifies that something (the source) has or owns something else (the target).

$$\begin{aligned}\langle\text{uses}\rangle \quad ::= \quad & \texttt{from}\langle\text{construct}\rangle \\ & \texttt{to}\langle\text{construct}\rangle \\ & \texttt{frequency(always | often | sometimes |} \\ & \texttt{a little | unspecified)} \\ & \texttt{when}\langle\text{description}\rangle\end{aligned} \tag{3.14}$$

The "uses" relation is defined as a relation that a construct uses another one, without a specific semantic meaning attached/predefined.

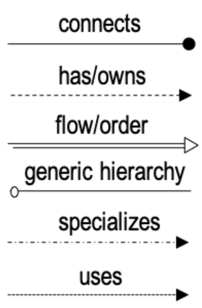
Relations	
 connects has/owns flow/order generic hierarchy specializes uses	Visual connector provide the design means to relate constructs above with each other. *Connects* is used to assign characteristics to concepts, whereas generic characteristics can be assigned to multiple concepts. The *has/owns* relation is intended to be used to represent partioning and composition of concepts. *Flows* defines sequences and control logic of functionalities. The *hierarchy* and *specializes* relation are used to structure concepts and identify conceptual hierarchies. *Uses* is applied to represent the dependency of specific concept and their characteristics with functionalities.

Table 3.3: CoChaCo Notation: Relations

Summary. The conceptual modelling (or design) of metamodels uses three main constructs provided by the CoChaCo metamodel: Concepts, Characteristics and Connectors. Relation types allow to connect the elements. An advantage of the approach relates to the visual interaction capabilities and creativity support. A designer has the opportunity to a) articulate and document design efforts of a metamodel, b) derive and verify artefacts formally for technical platform support, using custom extensions and transformations and c) provide machine processable input knowledge operations.

The syntactical view in CoChaCo using the grammar specification above supports knowledge operations with respect to structural aspects. The metamodel design can be assessed using static analysis mechanisms (e.g. verification for correctness using syntactical rules derived for specific runtime platforms, and code generation as input for implementation efforts, analysis functionalities as queries upon the design). As knowledge-based operations based on the semantics of the metamodel, an extended notion is required. This extension is presented in the following section introducing the notion of conceptual graphs, as the formal representation of a conceptual structure, on metamodels.

3.4 Foundation: Conceptual Graphs

In this section, the understanding of conceptual graphs for metamodels is introduced. In a first step, the foundational definition derived from literature for the concept developed is introduced. A conceptual graph, in its base definition, is a logical formalism that includes concepts, relations, individuals and quantifiers. More precisely, a conceptual graph is a finite, connected, bipartite graph with nodes of the first kind called concepts and nodes of the second kind called relations, where each relation has one or more arcs, each of which must be attached to a concept. If a relation has n arcs, it is n-adic with labels from 1...n (Sowa, 1979, p. 42). They are a graph-based knowledge representation of conceptual structures and reasoning model, that can be derived from their representation and maps to first-order logic/predicate logic to enable reasoning through inference. The translation mechanism, called "mapping" in (Wermelinger, 1995), is a re-defined translation algorithm/function between a graph-based representation and a logic-based view.

3.4.1 Structure of Conceptual Graph

The definition of a conceptual graph follows the formulation of the fundamental notions presented in (Nguyen and Corbett, 2006), repeated and summarised below as input for the mapping towards the ChoCaCo language. The mathematisation has been derived from Sowa's original idea discussed in (Sowa, 1984). The definition respects the "support" structure required for conceptual graphs, as "a conceptual graph has no meaning in isolation. Only through the semantic network are its concepts and relations linked to context, language, emotion, and perception" (Chein and M. L. Mugnier, 1992, p. 366).

Conceptual Graph. A conceptual graph G that respects the support structure coined "canon" as a quintuple.

$$\mathbf{G} = (\mathbf{C}, \mathbf{R}, \mathit{type}, referent, arg) \tag{3.15}$$

where the following definitions are applicable, referencing the support structure as required:

1. **G** is a bipartite (two disjoint sets of **C** and **R**, without adjacent of two vertices within the same set), connected (any vertice can be reached from any other vertice), finite (a defined set of vertices) graph.
2. **C** is the set of concept nodes. $C = \{c - vertices\}$ defines the concept set, where each c-vertices is derived from a concept type. All concept types in C are denoted as T_{CG}. T_{CG} is a subset of T_C.
3. **R** the set of relation nodes. $R = \{r - vertices\}$ defines the relation set, where each r-vertices is derived from a relation type. All relation types in C are denoted as T_{RG}. T_{RG} is a subset of T_R.

   An important distinction between C and T_{CG}, R and T_{RG} respectively is that for the C and R may contain duplicate entries, whereas the type sets are disjunct, as a consequence from T_C and T_R being disjunct. Details on types (concept, relation, individual are available in 3.21 as used in the *type* function below.
4. *type* is the surjective function that describes this association between a concept/relation and its type. The function is expressed as

   $$\mathit{type} : \mathbf{C} \cup \mathbf{R} \rightarrow T_{CG} \cup T_{RG} \tag{3.16}$$

   with

   $$\begin{aligned} &\forall \mathbf{C} \in \mathbf{C} \quad \mathit{type}(C) \in T_{CG} \quad \mathit{and}\, T_{CG} = \mathit{type}(\mathbf{C}) \\ &\forall \mathbf{R} \in \mathbf{R} \quad \mathit{type}(R) \in T_{RG} \quad \mathit{and}\, T_{RG} = \mathit{type}(\mathbf{R}) \end{aligned}$$
5. *referent* is the surjective function that associates the concepts with individual markers as:

   $$\mathit{referent} : \mathbf{C} \rightarrow I_G \quad \mathit{and}\, I_G \subseteq I. \tag{3.17}$$

A blank referent is denoted as the generic marker * . The definition of I as the set of individuals of the support structure is defined in 3.21.

6. *arg* is a partial function that associates a relation node with ith concepts, respecting the signature of R.

$$arg\,:\,\mathbb{N}\times\mathbf{R}\rightarrow\mathbf{C} \tag{3.18}$$

where $\mathbb{N}$ are the strictly positive natural numbers and the ith argument as a concept vertice, is

$$\forall i\in\mathbb{N}\,\forall R\in R\quad arg(i,R)$$

if it exists in the relations signature.

For the above definition of G, the following conditions are applicable in the context of the canon K.

– *referent* and *arg* must satisfy the conformance relation *conf* of K. This has the implication that individual markers can be used in different concept nodes, but need to be compatible in a sense of subsumption, defined in the *conf* function. This can be expressed as

$$\forall\mathbf{C}\in\mathbf{C}\,conf(referent(C))=i\leq type(C) \tag{3.19}$$

– *arg* and *type* must satisfy the canonicial basis function B of K.

$$\forall i\in\mathbb{N}^{+}\forall R\in\mathbf{R}\forall C\in\mathbf{C}arg(i,R)=C<=>type(C) \tag{3.20}$$

Canon. The conceptual graph G builds on the canon K defined as

$$\mathbf{K}=(\mathbf{T},\mathbf{I},\leq,conf,B) \tag{3.21}$$

where the following definitions apply:

1. **T** is a set of concept T_C and relation types T_R, defined as $T = T_C \cup T_R$. The two sets are disjunctive and finite. It is assumed that each set has a lattice structure, established through a subsumption or subtype relation $\leq$. Both sets are distinct; no duplicates exist.

2. **I** as the set of individuals or markers. An element of I is the instantiation or realisation of a concept type T_C. **I** also includes the generic marker/individual *****. **I** does not contain any duplicates either.

3. $\leq$ is the subsumption relation/function of **T** and is defined as

$$\leq: \; (T_C \times T_C) \cup (T_C \times T_C) \rightarrow \{true, false\} \tag{3.22}$$

4. *conf* is the function that relate each individual marker, with the exception of the generic one ***** to a concept type. The restriction proposed in (Nguyen and Corbett, 2006, p. 263) that each individual marker can only relate to one concept type is followed, considering the subsumption hierarchy of concepts derived from concept types.

$$\mathit{conf} : \mathbf{I} \setminus \{*\} \rightarrow T_C \tag{3.23}$$

5. B is the canonical basis function responsible to associate each relation type with the concept types that are allowed in the relation. The set B(r) is an ordered set and may contain duplicates, n denotes the arity or valence of the relation type, as a result from each c_n defined as the argument. The function is defined as

$$B\, : T_R \rightarrow \varphi(T_C) \tag{3.24}$$

   where

$$\forall r \in T_R \quad B(r) = \{c_1, ..., c_n\}.$$

For the above definition of K, the relations/functions $\leq$ and B are linked. This means that the canonical basis function of any relation type needs to satisfy the subsumption defined for the supertype.

The ontology definition in (Nguyen and Corbett, 2006, p. 264) is respected for the alignment to ChoCaCo but not explicitly described in the section as it is defined as a subset of the sets defined for K, within the context of the ontology O for a domain M. The relations/functions are carried forward from K, applied on the subsets defined for the ontology/domain.

3.4.2 Basic Operations on Conceptual Graph

The above definitions represent the support structure, usually denoted vocabulary of the conceptual graph formalism. The core feature of conceptual graphs is the transformation logic to first-order-logic, defined as Φ, laid out in (Sowa, 1984) in detail, and continuously refined and corrected (e.g. (Wermelinger, 1995)). The understanding of metamodels as conceptual graphs results in the implication that logic based operations can be applied upon the structure that support knowledge operations. In (Nguyen and Corbett, 2006, p. 268-270), these operations are classified as projections. Such projections are defined to enable comparison and canonical operations, supporting inference and discovery of knowledge. Consequently, the graph-based representation enables a deductive discovery of hidden concepts and relations within metamodel fragments as input for the dynamic composition operations.

(Chein, M. L. Mugnier and Croitoru, 2013) consider these projections and defines them as "Core operations for Conceptual Graphs". These operations are introduced below as the foundation for metamodel operations proposed, supporting the federated design of metamodels. Operations identified by the authors build upon syntactic operation on knowledge graphs, which are called graph homomorphism. The authors summarise in a first step mathematically how graphs can be used as a representation format for reasoning. Such operations build on the definition of a graph for knowledge representation consisting of

– the *Conceptual Vocabulary* as two sets of node types and relation types,

– *Basic Graph (BG)*, that build upon the vocabulary and represent the knowledge of the underlying domain, and

– extensions towards *Complex Graphs* that consider nesting and rules as hypothesis - conclusion relations within the graphs.

At its core, the subsumption relation introduced above in the formal definition 3.21 defines the knowledge operations of conceptual graphs. Reasoning (defined as a deductive approach) "can be defined either by a sequence of elementary operations or by the classical homomorphism notion" (Chein, M. L. Mugnier and Croitoru, 2013, p. 255). The elementary operations are classified as generalisation and (inverse) specialisation operations upon BG, whereas the intuitive definition of the authors in (Chein, M. L. Mugnier and Croitoru, 2013, p. 255). *Specialisation* is the category of knowledge operation that summarises any extension of knowledge to an existing graph. The resulting graph, after the operation, represents more knowledge of the initial one. *Generalisation*, as the inverse operations, constructs a graph that contains less knowledge, a more general picture of the system under study is the result.

Generalisation Operations. Generalisation operations enable the transformation of one conceptual graph to another that is less or equal expressive than the previous one. As such, this operation is characterised as a unary transformation of the graph; different types of these operations can be distinguished. The assumption is that the underlying vocabulary stays stable.

– *Subtract*: the subtract operation deletes one or more components of the initial graph. As such the resulting graph is more general than the previous one, as information is reduced.

– *Detach*: detaching means splitting a concept node into two concept nodes, the initial edges related to the node are shared between the two new nodes. The detached BGs are more general, as the relational nodes are not linked to a single instance of the node, but distributed to the detached ones.

– *Increase*: increasing means to elevate the label of a concept or relation, either on type or marker level. For the type level increase, the vocabulary hierarchy needs to be respected, e.g. elevating/increasing it to a high level node/relation type within the same tree branch. This means that the increase operation on node and relation type is limited to the super-types defined in the vocabulary. For a marker increase, the same logic applies wheres the increasing to the generic marker (as a it is consider greater than any individual marker) is always possible. This implies that a concrete marker can also be substituted by the generic "any".

- *Relation duplication*: this operation duplicates a relation node, resulting in a "twin" relation. Both nodes (original and duplicated) have the same signature and type. As no new information is in the resulting BG, the operation is considered as a generalisation in the broad sense (the result it the semantically equivalent).
- *Copy*: copying a BG results in an "isomorphic and disjoint copy of it"

Based on the definition, a conceptual graph G is a generalisation of H if a sequence of elementary generalisation operations can be identified that leads from H to G. An approach that targets similarity matching based on generalisation operations is introduced in in (Huibers et al., 1996). The authors propose a framework to determine whether a graph is "about" another one, investigating on the generalisation operations that have led to the resulting graph. Elementary generalisation steps can be identified and an assessment whether a specific graph is about another one can be derived. This technique is relevant for the federation concept proposed in this research work, as the aboutness, i.e. the degree of similarity for abstract metamodel fragments in specified ones can be determined.

Specialisation Operations. Specialisation operations target any transformation of a conceptual graph that elevates the knowledge representation by it. Specialisation operations are defined as inverse to generalisation operations. The *Copy* operation of above can also be considered as a specialisation, as the resulting graph is an isomorphic copy and is skipped in the listing below.

- *Disjoint sum*: as the inverse operation to *Subtract*, the operation sums two disjoint conceptual graphs into a a common one
- *Join*: the inverse operation to *Detach*, two concept nodes with the same label are combined, resulting in a join of the conceptual graphs.
- *Restrict*: the label of a node or relation are specialised. In contrary to the *Increase* operation, the type or marker are made more specific during the operation.
- *Relation simplify*: considers the removal of twin operations with a conceptual graph.

Specialisation operations can be summarised as any transformation of a conceptual graph that results in a more concrete and specific knowledge representation, therefore providing a higher degree of expressiveness.

Homomorphism. Homomorphism can be assessed through the elementary specialisation operations. It determines whether two conceptual graphs are mapped on each other as a result of a transformation. Homomorphism between two conceptual graphs G to H exists in case a) the concept node set and relation node set of G can be mapped to H, b) edges are preserved and c) concept/relation labels may be decreased. Homomorphism implies that G is a generalisation of H, and H is a specialisation of G. Elementary generalisation and specialisation can be identified from G to H and vice versa.

Logical Correctness and Reasoning. Logic equivalent and the mapping from conceptual graphs have been introduced by Sowa in (Sowa, 1984). Reasoning operations on conceptual graphs are enabled by the semantics defined in the mapping towards first-order-logic of the graph. This has the implication, that one one hand graph-based operations are applicable using the graph syntax, and subsumption is enabled as a deduction within the logic-based representation of the conceptual graph.

Summary. Chapter 3 translates the abstract observations, hypothesis and needs articulated in chapter 1 and 2 to a concrete level: design requirements for digital intelligence ecosystems. The concretisation is based on motivational cases derived from the OMiLAB environment in order to provide application cases that articulate the need observed. These cases are instantiations of intelligent offerings and showcase how cognitive functionality is established within a distributed space of modelling services. Harmonisation and federation concepts are defined as bridging elements of conceptual structures. In contrast to monitoring and data-driven approaches, the contribution established for digital intelligence is anchored on type level i.e. the metamodels defining the dependencies, links and relationships within the ecosystem.

The foundation to enable such functionality is identified: a graphical approach to design metamodels is proposed building upon the Concept-Characteristic-Connector (CoChaCo) space and its formalisation as conceptual structures. CoChaCo is positioned as a language to describe a) requirements of modelling services on metamodel level with a high degree of freedom and flexibility. As the constraints imposed in the language are on a minimal level, design decisions can be visualised and depicted in an agile manner. In parallel, a formalism to represent the knowledge within the design is proposed. Sowa's conceptual structures are applied to translate the design language into formal concepts that support machine processable semantics.

The formalism, as initially specified by Sowa and refined by various research groups is introduced and its key characteristics for knowledge representation are discussed: on one hand the structural aspect are defined, utilising the mathematical foundation as a baseline. This formalism enables capabilities on processing that are assumed to be applicable for the design process of metamodels.

Both elements, CoChaCo as a design language and knowledge representation formalism of conceptual structures are combined in the following chapter 4, elevating the semantic expressiveness of metamodels, its dependencies and relations to other domains and requirements. This technique is coined "Harmonisation", as the contributing/participating metamodel in a modelling ecosystem are a) exposed formally and b) enable intelligence functionality. Harmonisation is defined as a coupling of concepts (virtual or through similarity/equivalence relations that encompass functionality on each relation identified. Intelligence functionality is defined on a federated level. This means that the functionality spans potentially multiple metamodels and is independent from a concrete implementation case.

Chapter 4

Metamodels as Conceptual Structures

For the specification of the technique presented, metamodels are understood as a knowledge representation of a specific domain (considering the distinction between application domain and industrial domain) that support the conceptual design of the underlying system, applying a vocabulary that is close to the experts perception. Related work in this field has shown that this closeness (in contrast to general purpose approaches) elevates the efficiency and quality of the created model.

In order to support querying and deductive reasoning on metamodel artefacts, conceptual graphs have been selected as a knowledge representation format as the embedded logical foundation enables knowledge operations on design level and provides the foundation for the intended federation concept. Federation is understood as the capability to identify similarities between available fragments. Similarities are on one hand structural, in case the metamodel matches against is domain agnosticn one, on the other hand semantical in case domain-specifics are relevant for the combination and harmonisation.

In this section, an appropriate mapping between the metamodel design language CoChaCo and the theory of conceptual graph is introduced. This implies that the syntactical specification of CoChaCo acts as the common denominator for the conceptual graph structure and on the other hand specialises the generic conceptual graph/logical representation for metamodels. The definition of this mapping is based on the following observation:

– *Common Canon/Conceptual Vocabulary:* the structure of CoChaCo as shown in 3.6 is defined as the invariant for the definition of the "canon" of the conceptual graph. This requirements is based on the assumption, that any metamodel can be represented in CoChaCo, due to its generic and abstract nature. For the mapping towards conceptual graphs, the term Metamodel Foundation Graph (MFG) is introduced as an invariant on abstract vocabulary level. The rationale behind this assumption is to establish a harmonised data structure for metamodels, that is applicable for advanced design operations. The EBNF syntax of CoChaCo is used as an input for the common and abstract vocabulary specification. Following this approach, the challenge of isolated graphs as discussed in (Cyre, 1997) is prevented. A set of isolated graphs, as shown in the motivational examples in Fig. 3.3 and 3.4 are harmonised via the abstraction towards a common vocabulary. The use of EBNF follows the observation in (Xia and Glinz, 2003, p. 187), that graphical modelling can be defined in an efficient and re-useable manner, using "the elegance and simplicity of EBNF".

Even though it is assumed that any metamodel can be mapped to CoChaCo, evaluation is required building on the results of the OMiLAB as input, presented and discussed in section 7.1.

- *Alignment between Metamodel Design Process and Conceptual Graph Representation:* Metamodels for a specific domain are designed using the CoChaCo language. As the common canon is established, these design results instantiate the concepts, characteristics and connectors and provide additional meaning. The canon is therefore specialised on type level initially and semantically enriched through the specification. The graph-based representation should stay transparent for the metamodel engineer, the interaction is either graphical or programatically via CoChaCo.

- *Relational Semantics:* Relations between the concepts are considered on two level, representing the semantics of the metamodel as a conceptual graph: a type-level specialisation/generalisation to identify hierarchies within the concepts, and associations between concepts as relation types. The meaning of a concept is therefore a combination of these relational aspects and its attribute syntax. Related work by (Kocura, 2000) is considered for this requirement.

- *Similarity Matching:* Instance-based matching is considered in the design. This requirement stems from the observation, where the metamodel fragment is semantically weak and the instance information constitute the meaning. Following the approach presented, instances can be represented and queried similarly as concept elements as they are understood as labelled instances of concept and relation types.

The observations of above result in the mapping of the language-based understanding of a metamodel towards a knowledge representation as conceptual graphs. The mapping is graphically shown in Fig. 4.1 and formally described below applying the syntax introduced in section 3.4.

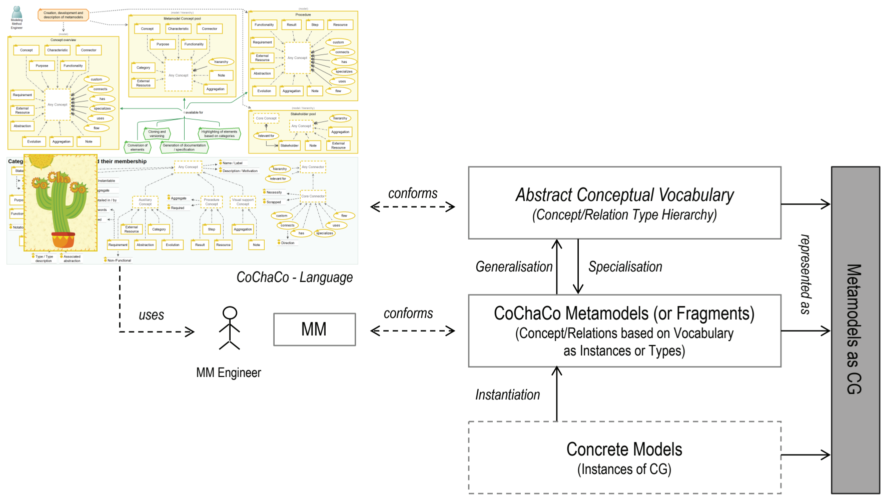

CoChaCo - Language
conforms
uses
MM Engineer
MM
conforms
Abstract Conceptual Vocabulary
(Concept/Relation Type Hierarchy)
Generalisation
Specialisation
CoChaCo Metamodels (or Fragments)
(Concept/Relations based on Vocabulary as Instances or Types)
Instantiation
Concrete Models
(Instances of CG)
represented as
Metamodels as CG

Figure 4.1: Mapping CoChaCo to Conceptual Graphs

For the research objectives of this thesis, the abstract conceptual vocabulary and CoChaCo Metamodels are considered and conceptually introduced in the following. The instantiation level is shown in Fig. 4.1 for completeness reasons as the actual use of metamodels and models derived from them implicates harmonisation aspects, but is not in scope of the research performed.

4.1 Abstract Conceptual Vocabulary for Metamodels

The abstract conceptual vocabulary for metamodels (or "Canon" as defined in 3.21) consist of the language elements of CoChaCo on syntax level and hierarchy, detailed below for the concept types, relation types and their subsumption.

$$\mathbf{K}_{CoChaCo} = (\mathbf{T}_{CoChaCo}, \mathbf{I}_{CoChaCo}, \leq, conf, B) \tag{4.1}$$

where the following definition are applicable:

$$T_{CoChaCo} = T_{C_{CoChaCo}} \cup T_{R_{CoChaCo}}$$

defining the collection of all concept and relation types, detailed below

$$\begin{aligned} T_{C_{CoChaCo}} = \{ &construct, structure, behaviour, \\ &concept, characteristic, connector, \\ &purpose, functionality, stakeholder, \\ &name, notation, isInstantiable, description, \\ &bidirectional\_connector, unidirectional\_connector\} \end{aligned} \tag{4.2}$$

as the collection of concept types exclusively available within the abstract vocabulary,

$$\begin{aligned} T_{R_{CoChaCo}} = \{ &relation, has, custom, connects, uses, specialises, \\ &flows, compose, aggregates, \\ &derived_relations, belongs_to\} \end{aligned} \tag{4.3}$$

as the collection of relation types exclusively available within the abstract vocabulary,

$$I = \{*\} \tag{4.4}$$

as the set of allowed instances. Only the generic marker defined as the "fictitious instance that can be associated with any marker" (Nguyen and Corbett, 2006, p. 262) is allowed on this level.

The hierarchical structure is established using the subsumption relation in the canon, detailed mathematically and graphically below. The subsumption relation for concept types is specified using the generic "construct" concept, that is used as a root element without any specific meaning for concept types.

$$\begin{aligned}
&structure \leq construct, \qquad (4.5)\\
&behaviour \leq construct,\\
&concept \leq structure,\\
&characteristic \leq structure,\\
&connector \leq structure,\\
&purpose \leq behaviour,\\
&functionality \leq behaviour,\\
&stakeholder \leq behaviour,\\
&name \leq characteristic,\\
&description \leq characteristic,\\
&notation \leq characteristic,\\
&isInstantiable \leq characteristic,\\
&bidirectional\_connector \leq connector,\\
&unidirectional\_connector \ \leq connector
\end{aligned}$$

The graphically representation is shown in Fig. 4.2. For the purpose of graphical modelling, specific types for characteristics (name, notation, description, capability to be instantiated) and connector specification (bidirectional and unidirectional) are provided explicitly. An important consideration for the understanding of metamodels as conceptual graphs relates to the specific semantics of connectors. Connectors are defined as a concept type, rather than a relation type to enable querying, matching and reasoning upon the concept.

The hierarchical structure is established using the subsumption relation in the canon, detailed mathematically and graphically below. The subsumption relation for relation types is specified using the generic "relation" node, that is used as a root element without any specific meaning for relation types.

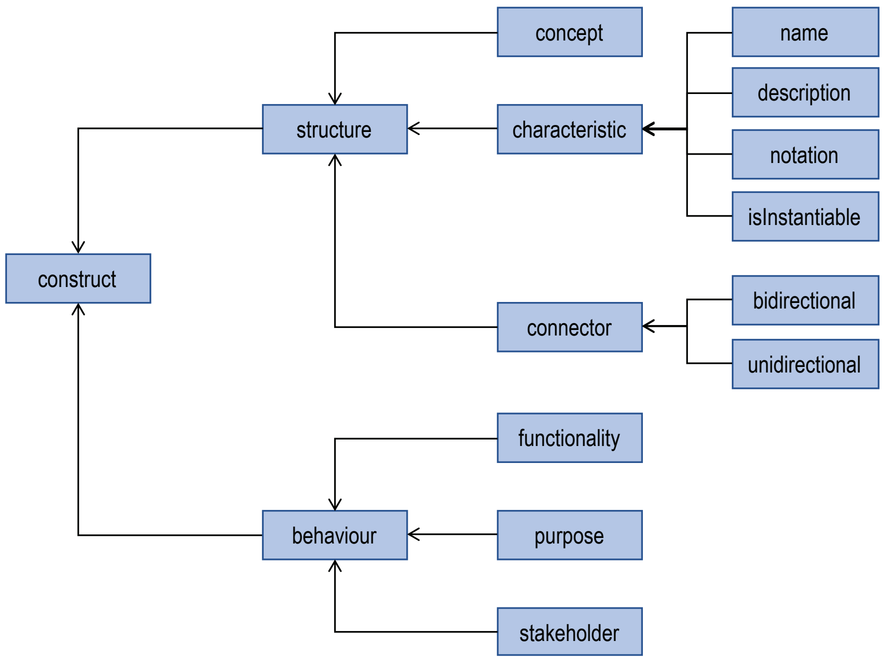


Figure 4.2: Concept Types: Abstract Metamodel CG Vocabulary

$$
\begin{aligned}
&has \leq relation, \\
&custom \leq relation, \\
&connects \leq relation, \\
&uses \leq relation, \\
&specialises \leq relation, \\
&flows \leq relation, \\
&compose \leq relation, \\
&derived\_relation \leq relation, \\
&aggregates \leq composition, \\
&belongs\_to \leq derived\_relation, \\
&generalises \leq derived\_relation
\end{aligned} \tag{4.6}
$$

The graphically representation is shown in Fig. 4.3. As a notation, the CGDF as proposed by (Sowa, 2009) is used. The relation types are applicable for the design of metamodels/metamodel fragments. Inverse relations are provided in the vocabulary to deduce relations and derive additional meaning semantic from the specification. An aspect considered relates specifically to the "specialise" (and its inverse opponent "generalise"), as it is explicitly available in the canon and not only via the subsumption logic of the canon. This design decision has been taken to allow for a general applicable approach to define "is a" or subtype specification during the design, without the need to adapt the canon for domain-specific considerations.

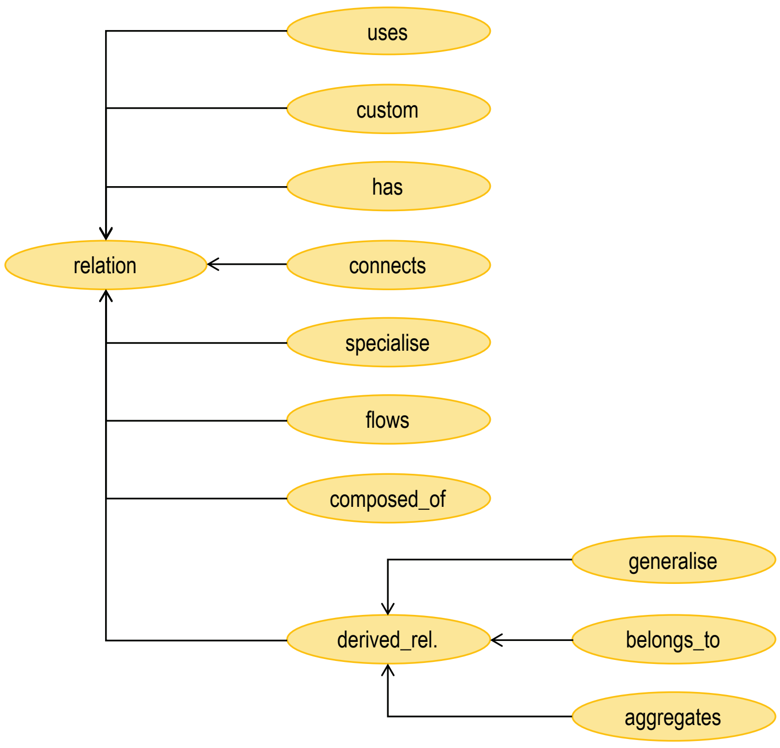


Figure 4.3: Relation Types: Abstract Metamodel CG Vocabulary

The conformance specification of instances is not applicable on vocabulary level, as only the generic marker is provided on this level. Applying the canon during the design will a) establish instances of concepts and b) allow for a conformance evaluation through the type function.

$$conf = \{\} \tag{4.7}$$

The canonical basis function B is defined for the relation types in a trivial manner. This design is applicable as it provides a flexible design space for metamodel, in contrast to a restrictive approach. All relation types are defined with an arity of 2, subsumption conformance is verified for this definition.

$$\begin{aligned}
&B_{relation} = \{construct, construct\} \\
&B_{has} = \{construct, characteristic\} \\
&B_{custom} = \{construct, construct\} \\
&B_{connects} = \{construct, construct\} \\
&B_{compose} = \{construct, construct\} \\
&B_{aggregates} = \{construct, construct\} \\
&B_{uses} = \{functionality, construct\} \\
&B_{specialises} = \{construct, construct\} \\
&B_{flows} = \{construct, construct\} \\
&B_{derived\_relation} = \{construct, construct\} \\
&B_{belongs\_to} = \{characteristic, construct\} \\
&B_{generalises} = \{construct, construct\}
\end{aligned} \tag{4.8}$$

The abstract vocabulary of CGs for metamodel design as introduced above defined the design space and elements to support syntactic and semantic knowledge operations beyond the basic conceptual graph operations introduced above. The advance on this level relates to how basic conceptual graph operations can be defined through chaining and composition to provide value for the design approach of federated metamodels.

The notion of elementary operations, homomorphism and logic correctness/reasoning introduced above are discussed in detail and exemplified in (Chein, M. L. Mugnier and Croitoru, 2013, p. 257). Applying any of these operations can be understood as a projection of the original graph. This projection is used to find out whether "two conceptual graphs are semantically compatible or similar" (Nguyen and Corbett, 2006, p. 266). Different types of projections can be distinguished (Concept, Relation, Conceptual Graph and Ontology Projection) based on the scope the target for the similarity identification. Such projections are a necessary means for the concept of federated metamodels: based on the assumption that a specific functionality is expressed in its required metamodel, and the processing mechanisms and algorithms this structure enables, it is required to match the required metamodel during the design/extension activities of a new one within the digital intelligence environment. The degree of similarity (structural, syntactical, semantical) determines whether a specific algorithm can be aligned with the design (top-down). This implies that the metamodel designer can assess and apply these building blocks, embed them using intelligent matching algorithms and adapt the design towards a fitting abstraction.

In the following section the representation of an example metamodel using the vocabulary of above is introduced. Querying and reasoning mechanisms are defined as input for matching techniques required in the design of digital intelligence ecosystems, operating on the canon and ~CG representation of the metamodels.

4.2 CoChaCo Metamodels as CGs

Two alternative approaches to define a metamodel that conforms to the CoChaCo language as conceptual graphs are discussed, both operating on the same abstract vocabulary introduced above. The concept developed in this section contributes to the challenge of aligning the metamodel design phases and processes to support knowledge operations based on CGs.

Language-based Approach: Metamodel Concepts as CG Instances. Following this approach, the concepts of a metamodel are defined as instances of the abstract metamodel. This implies that the set of instances of the vocabulary becomes concretely specified as instantiations of the types. The consequence is that a further instantiation is not possible on conceptual graph level, but needs to be mapped via the specific relation types "specialise". In CGs instances of instances are not considered in the core definition and would violate the conformance relation defined.

The abstract vocabulary, defined by CoChaCo, is not extended and stays invariant. Instances of the abstract vocabulary represent the metamodel and describe their relation semantically. This restriction applies for concept and relation types. To derive the conceptual graph, the transformation from CoChaCo is performed through mapping the concepts as instances of the vocabulary, relations are directly mappable. Fig. 4.4 shows an neutral example metamodel in CGDF. The concepts represented as a conceptual graph are introduced in plain text and defined in CGIF.

The following definition hold for the example metamodel:

– modeltype X is composed of concept A, B and C,

– modeltype Y is composed of concept A and B,

– modeltype X has a characteristic N,

– concept B specialises concept C,

– connector A2B connects concept A with B,

– concept C has a characteristic N.

The CGIF representation for the example metamodel is defined as

$$\begin{aligned}
[metamodel\_abstract] \quad & concept(< A >), \\
& concept(modeltypeX), \\
& concept(modeltypeY), \\
& connector(< A2B >), \\
& concept(< C >), \\
& concept(< B >), \\
& characteristic(< N >) \\
& composed\_of(modeltypeX, < A >), \\
& composed\_of(modeltypeX, < B >), \\
& composed\_of(modeltypeX, < C >), \\
& composed\_of(modeltypeY, < A >), \\
& composed\_of(modeltypeY, < B >), \\
& specialises(< B >, < C >), \\
& has(< C >, < N >), \\
& connects(< A >, < A2B >), \\
& connects(< A2B >, < B >), \\
& has(modeltypeX, < N >).
\end{aligned} \tag{4.9}$$

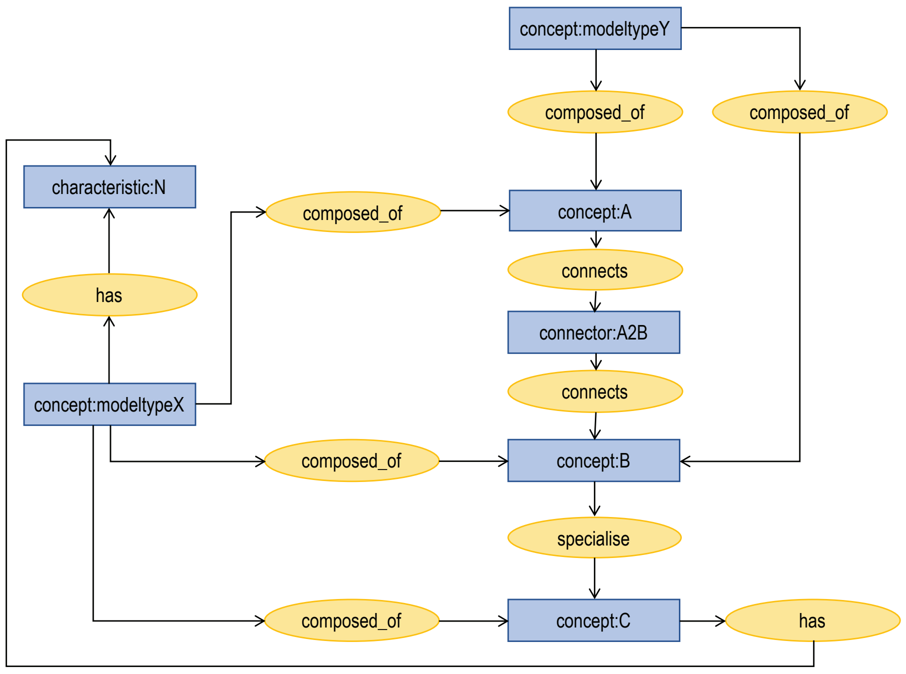


Figure 4.4: Example Metamodel in CGDF: Language-based Approach

The approach presented above shows that the metamodel and its concepts can represented as a conceptual graph applying the language mapping on instance level. The advantage of the instance based approach relates to the design of the graph, as no type related decisions need to be taken. The direct mapping via the language mapping to CoChaCo supports can be achieved as a direct translation between the different knowledge representations. Deficiencies can be observed with respect to the meaning of the instantiations. The semantics of a specific instance is related to the instance label and its relations (e.g. the meaning of a model type as a view is implicit - the "compose of" relation in addition to the instance label provide only an indication) and can not directly be used for reasoning and querying the metamodel, as all elements in the graph are either concepts, connectors or characteristics. A possibility to overcome this limitation on the semantic expressiveness is to define rules to deduce the knowledge on specific types and queries as input for further alignment and harmonisation phases. A rule to detect whether a concept is a model-type container/view or a modelling construct, applicable within the instance-based approach is formulated in the following. Co-reference links establish the fact-conclusion relation.

$$[derived\_types] \tag{4.10}$$
$$\begin{aligned}
&concept(modeltype),\\
&concept(modelling_concept)\\
&specialises(X1, modeltype),\\
&specialises(X2, modelling_concept),\\
&:-\\
&composed\_of(X1, X2),\\
&concept(X2),\\
&concept(X1).
\end{aligned}$$

This rule can be translated to the following statement: "For any concept X1 that is related to another concept X2 with type "composed of", X1 is a specialisation of the new concept modeltype, X2 a specialisation of the new concept modelling concept, the outcome when applied on the example metamodel is shown in 4.5, where concepts and relation in red are deduced from the original graph and can be further queried based on the types identified.

Similar rules are possible on the generic structure, nevertheless only syntactical results can be achieved, as the vocabulary is not aware of the domain, specific types required by the domain and only implicitly providing the means to externalise its meaning. Considering the harmonisation challenge introduced, this semantic expressiveness is limited to the language constructs provided by CoChaCo. As a result of this observations, an alternative approach is proposed that build upon the generic vocabulary, but extends on type level. The assumption is that based on elevated semantics within the vocabulary, knowledge operations for harmonisation and alignment can be supported in a richer way and assessing distributed metamodels becomes feasible.

Semantic-based Approach: Metamodel Concepts as CG Types. The semantic-based approach identified extends the language-based approach with additional semantics within the vocabulary. This implies that the ab-

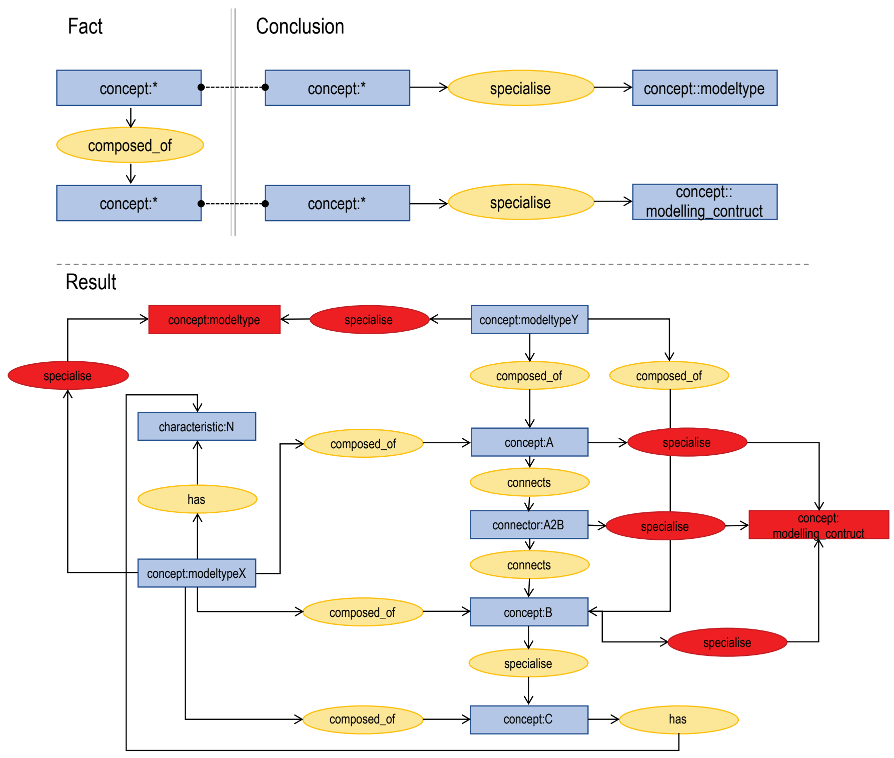


Figure 4.5: Type Identification Rule

stract vocabulary is used as the foundation, relevant concept and relation types are added to the hierarchy during the design. This elevation with domain-specific results in a higher expressiveness of the metamodel without loosing the capabilities of the language-based technique. The extension of the vocabulary and consequently of the graph representing the metamodel is considered a specialisation activity.

The assumed advantage of this approach in contrast to a pure language oriented technique relates to the expressiveness: a) the metamodel design can build on more meaningful concepts, including domain-aspects, b) technological aspect of e.g. metamodelling platforms can be introduced in the vocabulary, and c) instances level information can be represented, if required, as the concepts of a metamodel are defined on type level and instantiation is applicable.

The type-based understanding of a concept has implications on the mapping from CoChaCo as the redundancy in the specialisation/generalisation becomes obvious. Concept types are represented through their subsumption logic (see Fig. 4.2 and 4.3). Subsumption is considered as a "is a" relation, that can be understood as a specialisation/generalisation relation, applicable for both concepts and relations in a conceptual graph.

Applying this approach on the example used in the previous section, the following representation applies.

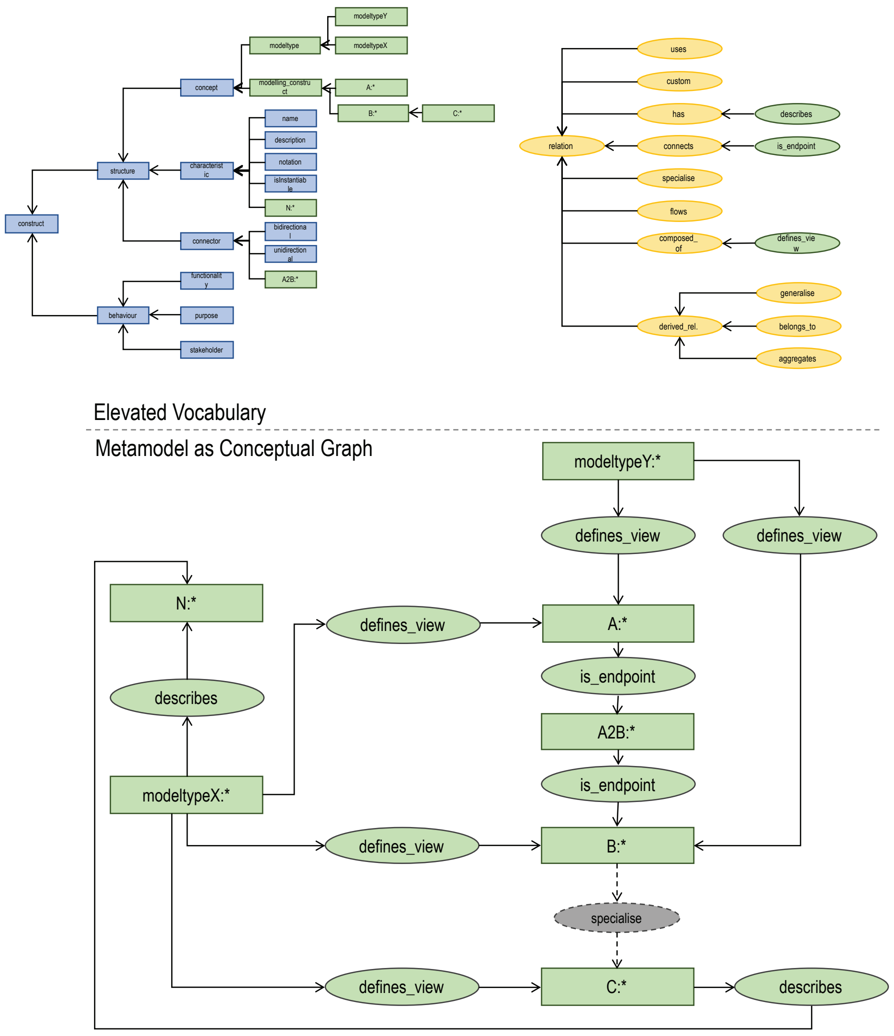


Figure 4.6: Example Metamodel in CGDF: Semantic-based Approach

From the example we can recognise, that the concept and relation types have been specialised by the specifics of the metamodel. Concrete types are now available for the definition of the metamodel that is represented using the generic marker for instantiation. The meaning of this representation can be summarised by the following statements:

- EVERY modeltype X defines as a view from using ANY modelling construct A, B and C instances,
- EVERY modeltype Y is defines as a view using ANY modelling construct A and B instances,
- EVERY modeltype X is described through a characteristic N,
- concepts A, B and C are modelling constructs,
- concept X and Y are modeltypes,
- A2B is a connector type,
- A is an endpoint to connector A2B,
- B is an endpoint to connector A2B,
- connector A2B connects modelling construct via endpoints of A with B instances,
- EVERY modelling construct C is described through a characteristic N.

Semantically this statements seem more adequate than the above, as the applicability as the type level formulation allows for an exemplification on instance level. In addition, a reduction of the metamodel design scope can also be recognised, as specialisation and generalisation are performed on type level, therefore do not need to be considered within the specification. In case required, the view on specialisation can be reconstructed through trivial reasoning that considers the type hierarchy and subsumption logic and re-introduces (virtually) the specialisation relation.

An additional aspect to be considered relates to the typing possibilities of concepts: using only the language-constructs, the query all defined modeltypes that have a specific characteristic N (e.g. as input for alignment and harmonisation), additional knowledge on the specifics and meaning of the term "modeltype" need to be introduced via reasoning. In the case presented, a modeltype is defined as a construct that has at least one "compose of" relation to another concept. Based on this rule, the type is derived. The rule has the implication, that modeltypes can be composed of other modeltypes and are an aggregation. In a next step we can perform the actual query as a projection, since the type is now known. The semantic-based approach allows to introduce domain-specific types directly in the hierarchy; concepts derived from these types are considered specifically and meaning can be attached to those concepts.

As the two approaches are building upon of each other, the semantic technique to extend the type hierarchy encapsulates and inherits the capabilities of the language-oriented viewpoint. This means that eh higher expressiveness also encompasses querying, projection and reasoning capabilities. Generic queries as discussed for the type recognition in Fig. 4.5 are still applicable, but can be refined and adapted to domain-specific considerations.

The CGIF representation for the example metamodel using the semantic approach is defined as

$$
\begin{aligned}
&[metamodel\_abstract\_semantic] \\
&\quad modeltypeY(X2), \\
&\quad < B > (X5), \\
&\quad < N > (X3), \\
&\quad modeltypeX(X1), \\
&\quad < A2B > (X7), \\
&\quad < C > (X4), \\
&\quad < A > (X6), \\
&\quad describes(X1, X3), \\
&\quad defines\_view(X1, X6), \\
&\quad defines\_view(X1, X5), \\
&\quad defines\_view(X1, X4), \\
&\quad defines\_view(X2, X6), \\
&\quad defines\_view(X2, X5), \\
&\quad describes(X4, X3), \\
&\quad is\_endpoint(X6, X7), \\
&\quad is\_endpoint(X7, X5).
\end{aligned}
\tag{4.11}
$$

In light of the harmonisation challenge within this thesis the semantic approach is applied. This has the advantage of elevated expressiveness and creates an extended vocabulary of the domains considered.

Summary. Chapter 4 is consider the foundational specification of the concept developed as the language based approach in CoChaCo is combined with the knowledge representation formalism in conceptual structures. The mapping between language and knowledge representation combines the advantages of both worlds: flexibility in the application (as the constraint in the language are designer-friendly) and query/reasoning capabilities of a fact-based format. It could be demonstrated that conceptual structures and their mathematical representation as conceptual graphs are applicable for a mapping as they are semantically close (in contrast to other knowledge representation formalism) to the understanding of a metamodel defined in (Karagiannis and Kühn, 2002; Kühn, 2004). Even though an overhead in construction can be observed (abstract vocabulary, canon), the semantic expressiveness is an adequate for the definition and transformation, especially when considering functionality mappings towards the formalism. This observation has been evaluated through alternative conceptual mappings and assessing their expressiveness in first-order-logic/predicate logic. The extension capabilities of the selected approach (hierarchical vocabulary and instantiation in the fact base in contrast to individual-oriented specification) showcase that it is potentially possible to also handle model instance with the same formal mapping.

As an outcome and take-away result from chapter 4, the mapping function and technique to represent CoChaCo is established. This result is input to the design techniques (how are metamodels in an ecosystem design and operationalised, how are operational metamodels applicable for the technique) presented in the following chapter, including harmonisation (similarity, virtual relation concept) and federation approaches (capabilities mapping). An evaluation performed in this respect relates to operations that support similarity identification for the purpose of closeness calculation of metamodel fragments and identification of functionalities during the design.

Chapter 5

Modelling Ecosystems: Harmonisation and Federation

Modelling ecosystems build on an arbitrary set of modelling methods, and their corresponding metamodels. Based on the observation, that a general-purpose metamodel can not cover any aspect of such an ecosystem, harmonisation concepts are required to provide guidance to the knowledge engineer how the glue between the constituting metamodel elements can be established. These concepts are introduced and discussed in this chapter.

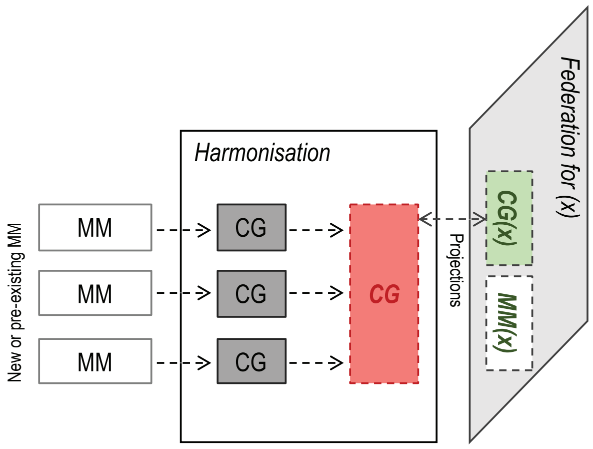


Figure 5.1: Harmonisation towards Modelling Ecosystems

In addition to harmonisation, intelligence mechanism are needed that build on the harmonised ecosystem as a foundation and provide means to assess, evaluate, visualise and consequently comprehend the complex relation between the model artefacts established. For this purpose, federation concepts are proposed and introduced.

The common layer and foundation for harmonisation and federation is the notion of metamodels as conceptual graphs introduced in Chapter 4. For harmonisation, the applicability of CGs during design and operation are discussed, specifically focusing on the "glueing" aspects between metamodels on similar or different abstraction levels. Similarity matching is considered a possibility to identify and assess equivalence between concepts and relation building on the same vocabulary, nevertheless "virtual" semantic relations between concepts are enabled through conceptual relation, that do not explicitly influence the metamodels but establish the ecosystem characteristics as an extension to the base definition in CGs.

5.1 Harmonisation Concepts for Metamodels

Harmonisation of metamodels aims at combining and glueing together metamodels of different abstraction and purpose. The harmonisation is considered to be performed in a manner that the autonomy of each metamodel is not effected, but a common layer of interaction is established that can be used for cross-layer functionalities.

5.1.1 Harmonisation of Metamodels during Design

Harmonisation can already happen during the design of a metamodel. By re-using abstract patterns of metamodels, the metamodel engineer construct the metamodel and implicitly aligns the results achieved on one hand with the abstract vocabulary, on the other hand with patterns defined for a specific purpose.

The following design patterns are based on observations of modelling method realisation and implementation projects within the OMiLAB at the University of Vienna. The practical experiences of realising modelling methods within the lab have been used to classify typical design approaches, introduced initially in (Utz, 2018b). Design techniques recognised are derived from specific development projects. Additional design patterns might exist, as it can also be seen from the analysis of resulting metamodel design formats discussed in (Bork, Karagiannis et al., 2018). The following list presents an indication on the need to explicate such processes.

- **Design from Scratch.** A new metamodel is designed from scratch based on the requirements of the domain. Mechanisms to support creativity without enforcing formal rules on the design artefacts are used

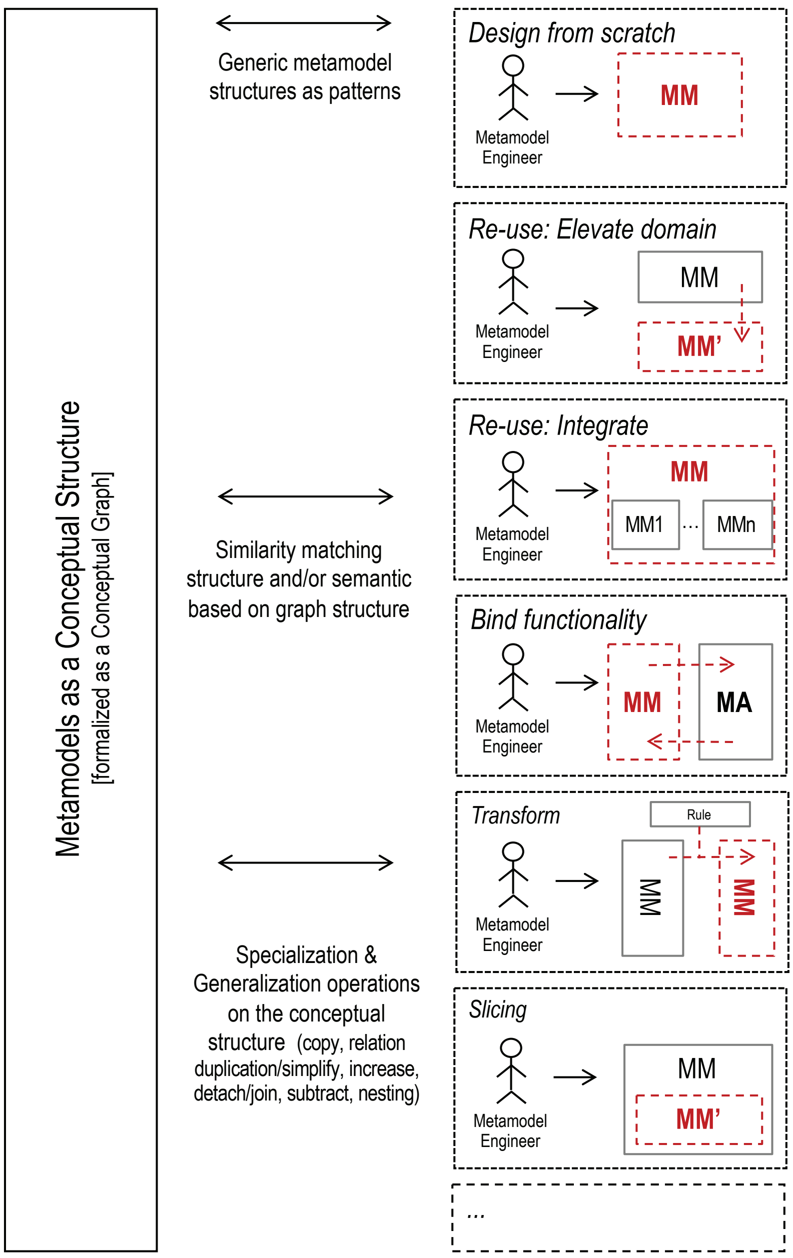

Metamodels as a Conceptual Structure
[formalized as a Conceptual Graph]
Generic metamodel structures as patterns
Similarity matching structure and/or semantic based on graph structure
Specialization & Generalization operations on the conceptual structure (copy, relation duplication/simplify, increase, detach/join, subtract, nesting)
Design from scratch
Metamodel Engineer
MM
Re-use: Elevate domain
MM
MM'
Re-use: Integrate
MM
MM1
MMn
Bind functionality
MM
MA
Transform
Rule
MM
MM
Slicing
MM
MM'
...

Figure 5.2: Metamodel Design Techniques (Utz, 2019a)

during this phase. A possible approach to identify and externalise the domain can be found in design thinking approaches as discussed in (Miron et al., 2018). Exemplification and persona-based techniques that put the modeller(s) and their requirements as scenarios in fo-

cus are also applied. Typically, domain expert describe the modelling requirements as stories and scenarios that are then abstracted and aligned with processing functionalities required.
Patterns on abstract level support the design of the new metamodel, as syntactical elements and semantical construct to implement common structure based on the same approach. An example for such a pattern has been introduced in the motivational chapter of this thesis for the definition of a process-oriented metamodel. The definition of such patterns is based on the results achieved as a reflective process, using conceptual graphs of metamodels for the syntax definition and annotations to search and retrieve appropriate patterns.
The theoretical foundation to define abstract patterns are discussed in (Zwarts and Verkuyl, 1994) for conceptual semantics and (G. Mineau et al., 1999) to establish methodological support through formal concept analysis methods.

Conceptual Graph Operations. As an operation, generic patterns are the foundation for this design technique. The metamodel engineer selects a pattern, similar as a programmer selects a data structure for the implementation of an algorithms. The basic set of these patterns is generic and independent of domain aspects; elevated patterns could be understood as contributions to the repository of patterns on a higher expressiveness. CG operations required are related to specialisation of a selected generic pattern.

– **Re-use: Elevate Domain/Integrate.** Re-use constitutes an often applied design technique for the construction of domain-specific metamodels. Two different types of this technique can be observed in the practice of metamodel design: a) domain elevation, where an existing metamodel is extended and specialised for a domain or application need, and b) integration where fragments or sub-sets of pre-existing metamodels are re-purposed and combined in an integrated manner. For the elevation type, multiple examples can be found in literature where general purpose modelling languages are elevated to cover additional domain aspects (e.g. in (Zor et al., 2011) that introduces an extension of BPMN for the manufacturing domain or (Utz and Lee, 2017) that propose an elevation of a domain-specific metamodel for industrial production processes towards analytical simulation features).

Conceptual Graph Operations. Related work with respect to formation rules in conceptual graphs as discussed in (Chein and M. L. Mugnier, 1992; Chein and M.-L. Mugnier, 2008) provide the foundation as formation rules for this design technique.
Re-use as domain elevation is supported as a specialisation operation of abstract vocabulary, including pattern projections via queries as discussed in the "Design from Scatch" technique.

- **Bind functionality.** Another design technique recognised from the implementation projects in OMiLAB are related to extending a metamodel with processing capabilities. This implies that either the metamodel already provides the necessary syntactical or semantical requirements of the processing algorithm and mechanisms or needs to be extended with these aspects. Graph-based matching approaches as discussed in (Gallagher, 2006) are considered relevant, since the metamodel is presented as a graph structure and therefore allows for such matching algorithms.

  Conceptual Graph Operations. Matching as a form of discovery of structure and semantic is supported in CGs through query mechanisms. A required metamodel fragment, concrete or abstract, for a specific processing algorithm/mechanism is provided as the query towards the domain-specific representation. The result of the matching returns as projections (if existing) which means that the algorithms can be aligned to each projection or further refinement requirements.

- **Transformation.** The transformation of metamodels as conceptual graphs is defined as a graph rewriting operation that translates a metamodel in a specific format to another, maintaining the semantical notion from the source within the target format as a homomorphism from one to the other. Transformations are required to align and harmonise varying input formats for to trigger the design process, provide the structure required for alignment to a common canon or more generically, to have an aligned starting point for further design tasks. In case the transformation is semantically identical, meaning that no semantic loss can be observed, the source conceptual graph and target conceptual graph are isomorph. Transformation also considers interchangeability and interoperability of metamodels as conceptual graphs.

Historically, standardisation efforts for metamodels have been driven by the software engineering community. The evolution of XMI to interchange metamodels in a common format based on MOF and UML is applicable for a common language family, for the purpose of domain-specific specification/design, this is inappropriate, as the common denominator is not existing. Work performed in (Gutierrez et al., 2009; Baget et al., 2010) are considered with respect to model interchange as a transformation of conceptual graphs to ontological representations.

Conceptual Graph Operations. Transformation is supported through base operations on conceptual graphs as a combination of querying and extraction. It is specifically relevant for merging two metamodels of different nature into a common representation. As an initial step the alignment is supposed to happen on abstract vocabulary. Adaptors are proposed that perform the translation of an arbitrary format (e.g. standard schema representations, ontological representation) to the language-based vocabulary established by CoChaCo.

- **Slicing.** Slicing has been introduces as a design concept in (Bork, Karagiannis et al., 2018). Within the publication, the technique is used to identify reoccurring patterns of design during the practical work work with metamodels. As a design practice, slicing is understood within this research project with a wider scope, including semantic aspects of a metamodel and their purpose. Observing the work in academia of well-known metamodels, one can recognise that the inclusion of a specific domain aspect or purpose results in an extension of concepts and their scope. Prominent examples are the MEMO approach by discussed in (Frank, 2014), that encompasses a extensive set of diagrams/views to establish a multi-perspective environment; similar in Bee-Up, introduced in (Karagiannis, Buchmann et al., 2016; *Bee-Up* 2019), that applies a hybrid approach to combine metamodels of specific nature and background into a common environment. Re-using certain aspects of these metamodel sets is a challenge, as interdependencies need to be resolved and mitigated during the design. This is especially relevant, as the aspect of "purpose" requires a specific implementation rather than a metamodel that potentially covers anything.

  Conceptual Graph Operations. Slicing is supported by CG operations as an orchestration of query and extraction operations. The query formulated retrieves the aspects of the metamodel required for the specific purpose, extraction is applied retrieve the query results as projections into the derived conceptual graph.

The list of design operation above is considered as an indicative result based on observations from the practical metamodel design work performed within different OMiLAB projects and does not aim to be complete. The design techniques defined show that each operation is established as an orchestration of basic operations on conceptual graphs that translate the source to the target graph as a homomorphic representation. The design operations provide therefore the ability to transform/generate the formal knowledge representation of a metamodel considering the aspect of design trajectories (human interpretation of the conceptual structure and impact of a specific technique on the design) as a combination and composition of formation rules, orchestrated by the metamodel engineer. This is supported by intelligent mechanisms to infer/deduct additional knowledge through rules and verify the outcomes continuously.

5.1.2 Harmonisation Concept: Metamodel Alignment

The design techniques above result in fragments that are applicable for a specific domain and support the engineering of these artefacts. Building upon these outcomes, alignment in heterogenous modelling environment considers the aspects to combine the results in a meaningful manner. The need for this alignment has been introduced in the motivation cases in section 3.1.1 and 3.1.2. Two approaches have been identified as relevant for the alignment:

– *Similarity Matching.* aims at identifying overlapping elements/structure within two metamodels. These overlaps represent a possible semantic match that can be used to translate from one abstraction layer to the other (considering vertical integration aspects as introduced in (Kühn, 2004)) or horizontal similarity. A simplified graphical representation is provided in 5.3.

– *Semantic Alignment.* introduces the concept of "virtual relations" between concepts in different metamodels. Trajectories between concepts of different metamodels are established by the metamodel engineer as semantically rich chains/links. The concept builds on the

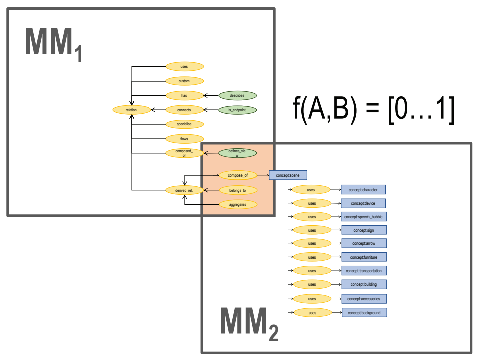


Figure 5.3: Similarity Matching of Metamodels

Semantic Lifting approach introduced in (Woitsch, 2013) and generalises it towards transitive links between concept and relation types. A simplified graphical representation is provided in 5.4.

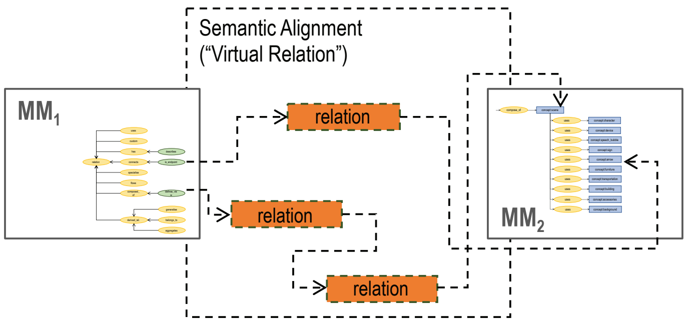


Figure 5.4: Semantic Alignment of Metamodels

Both techniques are discussed using relevant literature below, and the applicability for metamodel design and alignment is assessed.

5.1.2.1 Similarity Matching of Metamodels

Similarity matching builds on the assumption that a semantic distance or equivalence between two concepts/conceptsets can be determined and used as an input for alignment of two varying knowledge representations. The relation derived through similarity matching, states the degree or closeness of the compares concepts with each other.

(Poole and Campbell, 1995) introduce a metric to determine the similarity of conceptual graphs by applying an interest function based on *surface*, *structural* and *thematic* similarity. The approach the authors proposed uses the formula in 5.1 (Poole and Campbell, 1995, p. 296).

$$similarity[A, B] = \frac{interest[G]}{max(interest[A], interest[B])} \tag{5.1}$$

The degree of similarity is based on the ratio derived through the interest function applied on G, as the largest set of common information, divided by the maximum of the interest function applied on the information found in the graphs to compare A and B. The type of similarity to be assessed is defined within the interest function. The ratio as a result from the formula is 1 for an identical knowledge representation and 0 for no overlap/similarity identified. The interest function considers the type/purpose of the similarity matching. For *surface similarity*, the function considers the objects and attributes and ignores the relational environment, *structural similarity* would assign more weight on general connectivity (e.g. number of arcs, path length), whereas *thematic similarity* considers specific patterns in the graph for comparison. The authors in (Poole and Campbell, 1995) suggest a hybrid combination of the above functions as a domain specific implementation for similarity matching.

Corpus-based Metrics. In (Zhu and Iglesias, 2017), the authors focus on semantic similarity in knowledge graphs. They distinguish between a corpus-based approach and a knowledge-based approach to detect such similarities to categorise the "relatively large number of semantic similarity metrics" (Zhu and Iglesias, 2017, p. 74). The corpus-based approach uses distributed semantic similarity as derived through a large number of text content "learned" by a system as text corpora. Examples for this techniques are implementations by Wikipedia, that uses the occurrence of a term in their articles or the Google Distance Matrix, that builds on the search results of specific terms (Karve et al., 2019). This technique builds on the assumption that that the meaning of similarity of a concept is derived through a quantifier derived from a large set of data, can be represented in e.g. a multidimensional vector and compared. A relevant approach, as an elevation of basic corpus-based approaches has been discussed in (Patwardhan et al., 2003). The authors use WordNet (https://wordnet.princeton.edu/, (Miller, 1995; Fellbaum, 1998)) and derive similarity measures for terms as a result of the knowledge representation defined in the lexical database. The knowledge represented in WordNet, especially the depth in the hierarchy is considered and various algorithms have been proposed upon this structure to identify a similarity measure. An example of such a similarity can be found below, identifying the closeness between the term "person" and "actor", as potential concept types within two metamodels, the example uses the WUP algorithm (WuPalmer Similarity), calculating the "relatedness by considering the depths of the two synsets in the WordNet taxonomies, along with the depth of the LCS (Least Common Subsumer)" (Pedersen et al., 2004; Wu and Palmer, 1994).

$$Wu\_Palmer_{similarity} = 2 * \frac{depth(lcs(c1, c2))}{(depth(c1) + depth(c2))} \tag{5.2}$$

where
c1 ... label of concept type 1
c2 ... label of concept type 2

Instantiating c1 with "person" and c2 with "actor", results in a similarity score of 0.94 (0.00 means no relatedness, 1.00 identical), an indication that both terms are close to each other. Nevertheless when reviewing the lexical definitions of both terms, the limitation of such a context-free evaluation becomes obvious: the meaning, defined for the terms in the ontological knowledge representation has to be accepted and does not consider the context of the metamodel directly. This could potentially be done considered within the algorithmic comparison (similar as above, as an interest function) that covers the domain, but would impact the design process.

For the purpose of identifying similarities, the approach is considered as a support function for the metamodel engineer to identify potential overlaps and then act accordingly. A grid layout of two metamodels to be aligned is proposed that provides the metrics for each concept and relation type defined as a specialisation of the abstract vocabulary, ordered by the subsumption hierarchy defined.

Knowledge-based Metrics. In addition to the corpus-based approach, (Zhu and Iglesias, 2017) propose a knowledge-based approach that considers the context of the specific domain. The proposal is based on the semantic distance between concepts in the knowledge graph, the algorithmic solutions are comparable to those of the corpus-based approach with the difference that the context-specific knowledge graph is considered for the calculation rather than the derived from external sources. The metrics is calculated based on the subsumption hierarchy, either on concept/relation type level (hierarch depth, nesting), the path between the concepts (shortest path metric) as a distance measure or relational semantics (substituting the understanding that the meaning of a concept is derived from its hierarchical level, nesting and attributes defining it, with the meaning of the relation the concept owns/provides).

A relevant approach for conceptual graphs and its similarities has been developed in (Croitoru et al., 2007) as an ontology measure. The idea of this approach is to use queries in conceptual graphs and its resulting projections as a measure for similarity. The assumption is that a similarity is the result of a syntactical projection of one graph on the other, also argued within (Wang and Liu, 2008), proposing a measure for similarity based on objects and attributes.

As an input for the metamodel alignment technique proposed, the results in (Montes-Y-Gómez et al., 2001) are considered relevant from the perspective of the steps that are required to identify commonalities. The authors suggest the following method (Montes-Y-Gómez et al., 2001, p. 106):

1. Define the overlap between two conceptual graphs: in a preparation step, the two metamodels as conceptual graphs are merged as an overlap graph. MM_C represents the overlap graph that contains all concept and relation types of MM_1 and MM_2 respectively.

$$MM_C = MM_1 \cap GMM_2 \tag{5.3}$$

2. Measure the overlap of one graph towards the overlap graph. The measure is an indication based on the overlap established beforehand. Two measures are proposed by the authors, as a result of the bipartite nature of a conceptual graph: concept similarity and relational similarity both as relative scores between the individual and overlap graph.
3. Perform/Derive alignment modifications on a MM_1 or MM_2 and iteratively evaluate the measure.

A limitation of the approach for knowledge-based metrics relates to isolation within the metamodels under investigation. In case case the metamodels are not relate to each other and not provide an overlap at all, commonly refered to as "isolated graphs", the similarity measurements are not feasible. The abstract vocabulary defined in section 4.1 is considered a mitigation for this limitation. Assuming that any metamodel can be mapped to the abstract conceptual vocabulary, a common denominator is available for similarity measures, nevertheless alignment techniques are required that allow the metamodel engineer to define relations that cannot be derived based on existing input conceptual structures.

5.1.2.2 Semantic Alignment of Metamodels

Similarity matching focuses on equivalence; overlaps within the two constructs to be compares are quantified. This information can be used by the metamodel engineer to refine the design and perform alignment operations. In case of an existing similarity, the subsumption logic of one metamodel (consuming element) is extended with the counterpart. In case this similarity is not existing or only weak, an additional technique is required that elevates similarity to semantic rich alignment relations. These alignment relations are considered independent and do not change either part of the metamodel. Alignment relations are defined as "virtual". They exist as relations with a distinct meaning for the alignment to be performed as a means to reach harmonisation and elevate capabilities of the modelling ecosystem for digital intelligence. The following definition and characteristics apply for virtual relationships:

– Knowledge-Intense: are defined with a specific meaning assigned by the metamodel engineer explicitly,

– Combination: can be applied to align concept and relation types,

– Transitivity: are transitive and can therefore be derived also from other virtual relations that have been defined beforehand,

– Meaning: the semantics of such a relations is assigned, similar as for the conceptual graph itself, through a label, subsumption and conformance relations,

– Independent: no influence and impact is exposed on any metamodel this relation combines and harmonises,

– Useful/Applicable for Federation Concept: the relation is available for advanced federation concepts, similar as any other relation type/instance within the conceptual graph.

Figure 5.4 already introduced on an abstract level the concept, a indicative example is introduced below in 5.5 to showcase the purpose of virtual relations.

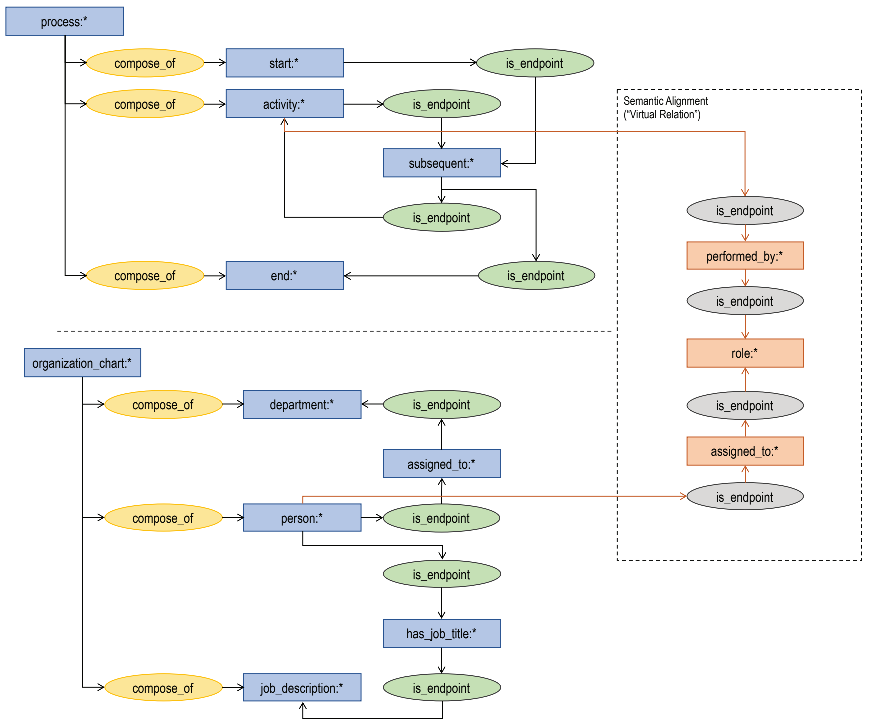


Figure 5.5: Harmonisation Example: Process and Organisation Metamodels

The example introduces two distinct metamodels: a process metamodel that considers the concepts of start, activity and end, logical connected using the relation type "subsequent". On the other hand, a metamodel that is capable to represent the organisational structure of an organisation is introduce, considering organisational units, persons and job titles as constructs.

Harmonising these metamodels following the similarity matching approach is not possible - as neither the corpus based approach nor the knowledge based approach yield any meaningful results. Based on the example above, the results exemplify this statement. For the calculation, the seman tic relatedness from (Wu and Palmer, 1994) as a reference points. The terms used for matching are used in a first step without any further contextualisation.

	process	start	end	activity
organization_chart	0.47	0.62	0.73	0.46
department	0.53	0.38	0.7	0.43
person	0.5	0.43	0.76	0.46
job_description	0.44	0.57	0.67	0.43

Table 5.1: Similarity scores based on (Wu and Palmer, 1994)

On a first glance, an indication that a semantic similarity is recognised, specifically between the "end" concept and "person", "department" and "organisation chart". Investigating the result of 0.76 for the relation between the concept "person" and "end", the context free nature of this assessment becomes obvious:

– "Person": for the term, the first found definition is used - the "a human being",

– "End": as all possible definition are allowed, without and contextualisation, "end" is defined as "the person who plays at one end of the line of scrimmage", a definition of a position in American football.

As a result, the term "person" and "end", match in the corpus-based approach, but are semantically distant. This observation can be mitigated by specialising the terminology used, e.g. "end" as "the final stage or concluding parts of an event or occurrence". The outcome of the similarity matching is then reverted to the following:

	$process_{\#1}$	$start_{\#5}$	$end_{\#3}$	$task_{\#1}$
$organization\_chart_{\#1}$	0.4	0.35	0.4	0.38
$department_{\#1}$	0.33	0.3	0.33	0.32
$person_{\#1}$	0.25	0.22	0.25	0.24
$job\_description_{\#1}$	0.38	0.33	0.38	0.35

Table 5.2: Similarity scores based on (Wu and Palmer, 1994) (refined definitions)

This refinement, as a result of subsetting the definition space to be closer to the context/domain is shown in Table 5.2. The numbering next to the concept name indicate the definition selected or adapted (activity has been replaced with task, as the term "activity" in WordNet does not specify the semantics intended. Significant overlaps on a semantic level can not be recognised. Consequently, the need to establish relations between concurrent metamodels has been developed.

Virtual relations are introduced to overcome such non-significant similarity based on any context-free assessment. The concept of "Role" is introduced virtually that covers the aspect of organisational assignment ("Every person has an assigned role") and process-oriented view ("Every activity is executed by a role"). This extension to the metamodels is not assigned to a specific one, but enabled as a virtual bridging element. The virtual elements (concept and relation types are specifically marked as such and used for federated model processing functionalities. These relations are understood as an elevation of the semantic lifting approach. Semantic lifting as introduced by (Woitsch, 2013) elevates both metamodels on a level that similarity can be defined and an integration is possible. Virtual relations cover these aspects naturally (lifting of domain concepts and integration as linking) with the additional consideration, that these elevations and relations are established on a virtual level. The alignment is therefore not subject to extension and modification of individual metamodels and provides the required means to harmonise also *operational metamodels* (defined as metamodels that are already implemented and deployed).

5.1.3 Harmonisation Concept: Operational Metamodels

Harmonisation of operational metamodels, defined as implemented and deployed metamodels already in use to create model artefacts, pose a specific challenge for the above concepts: as the metamodel is in operation, requirements for harmonisation can not be reflected in the design, but need to be realised as an alignment extension that is dynamically embedded. The operationalisation limits therefore the capabilities a knowledge engineer can apply to perform an alignment: models might already exist, that implement model fitting and interpretation of the user, the implementation approach and technologies chosen cannot be adapted or changed. Considering these different realisation platforms, a) adaptors are required to retrieve the running metamodel as a design artefact, b) translate it to a common representation format, c) define the alignment relations following the "virtual relation" concept presented in the previous section and d) define the impact of model artefacts during the harmonisation.

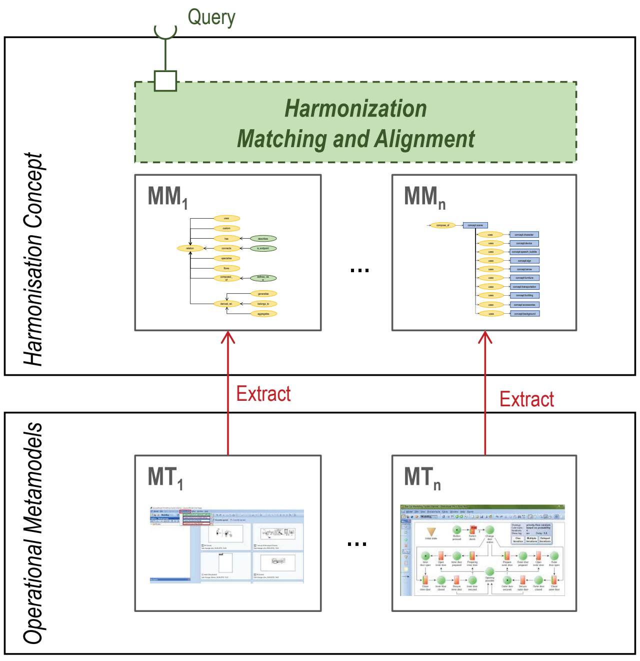


Figure 5.6: Harmonisation of Operational Metamodels

In such environments, virtual relations established act as the semantic "glue" between the metamodels; in addition, in the case of operational metamodels, also are applied to define queries and translation logic on instance level. Consequently, the conceptual graphs are not only a design artefact, but define the translation and query interface to assess a modelling ecosystem and provide input for comprehension as model based intelligence. Potential application cases for the above are:

– *Query related information*: virtual relations are used as a specification to define complex, cross-spanning querying mechanisms. The queries established through linking are not focused on a specific metamodel (and the modelling tool implementing it) but span across the linked implementations,

– *Information integration*: semantic relations allow to integrate information from one environment to the other as a decomposition (in case of top-down) modelling or abstraction (in case of a bottom-up approach). Knowledge represented in one environment is queried, extracted and translated to the related domain align the relations,

– *Dependency analysis*: visualisation of dependencies is supported on the harmonised level. Concepts used in one operational metamodel become useful in another domain or abstraction level, including provenance of knowledge.

For the realisation of the harmonisation concept, the implication is that the virtual linking environment is considered as an additional operational environment that operates independently, in exchange with the metamodel considered. These application cases add the requirement that the harmonisation concept is also exposed as functionality to the underlying metamodels and their users respectively. Depending on the realising technologies different manifestations can be noted: aspect-oriented injection techniques are applicable to impose the functionality on each individual operational metamodel and extend the individual functionality, service-oriented techniques can be used to add application/domain-specific service calls to the implementation, where harmonisation is considered as-a-service. The underlying technique applied can be summarised as "extract-combine-query" as the semantics of each involved metamodel needs to become accessible, defined as a conceptual graph and individual query techniques allow for an assessment of the combination.

5.2 Federation Concepts in Heterogenous Modelling Ecosystems

The harmonisation concept supports as an alignment approach the assessment of heterogenous modelling environments. Input are novel or pre-existing metamodels on different abstraction layers that are semantically aligned through virtual relations. The resulting conceptual graph represents the modelling ecosystem as a common, harmonised layer, useful for *intelligence operations* beyond model processing functionalities that are embedded in the individual metamodels.

The federation approach introduced in this section focus on these intelligence operation and how they can be realised for distributed modelling systems. As input for the development of the federation concept, *intelligence operations* are defined, based on the definition of "Digital Intelligence" introduced in section 2.1 and the motivational cases in section 3.1.1 and 3.1.2. In the context of heterogenous modelling environments, *intelligence operations* are defined as "any operation that provides insights in the structure, design, and operation of a distributed information system". This implies that stakeholders are collaboratively working on an intelligent offering, and need to respect/understand the impact of design decisions taken within their own scope and on other abstraction layers. Analytical functionality is therefore required to assess, evaluate and visualise the behaviour within the system. In an attempt to structure such operations, the following categories have been identified. based on the results in the field of software system analysis. Based on the related work on software visualisation in section 2.10 and specifically the results discussed in (Diehl, 2007), five categories supporting intelligence operations have been identified, ranging from basic logging to complex interactive visualisation and focus on the behaviour perceived within the environment.

1. *Logging* as a base/foundational functionality, interactions of any actor (human or system) become explicit. Logging is applicable to understand the behaviour of a system over time, assess system states and explicate provenance of changes within the environment. For a modelling system *logging* is relevant to depict the evolution of specific model artefact and track the model evolution process,

2. *Tracing* builds upon the logging of events in a specific distributed envir-

onment and elevates it towards a logic sequence of interactions within a time-span. Tracing provides the means to capture a sequence of event within a specific context and correlate logged events with each other. In software engineering, a trace is typically used to a) limit and filter the log entries, and b) track the dependencies between system components,

3. *Impact Assessment* elevates traces received towards an adequate assessment on a semantic level. This operation is performs the combination of logs and/or traces with the knowledge represented within the metamodel. A trace (or log), as a time-stamped series of event information, is correlated with the metamodels defining the stream of information, as a tree or network structure,
4. *Visualisation* is responsible to provide meaningful, graphical representation of the intelligence data. Visualisation is coupled with the structure and semantics of the contributing metamodels,
5. *Interaction* covers aspects related to user or machine interaction on the intelligence information. Interaction ranges from basic querying of information (e.g. log reviews, model information assessment) to complex interaction patterns for drill-downs and model harmonisation.

The categories introduced above have motivated the development of a federation concept for heterogenous modelling environments, graphically introduced in Fig. 5.7. The anatomy builds on the related work in the field discussed in section 2.9 and 2.10.

5.2.1 Digital Intelligence Building Blocks: Anatomy

Digital intelligence building blocks are introduced as the instantiation of the federation concept. As building blocks of the environment they are defined as re-useable elements that encapsulate intelligence operation that operates upon the common conceptual structure as a result of the harmonisation. They are defined on metamodel level and become useful within the operation of the heterogenous modelling ecosystem. Fig. 5.7 shows graphically the structure of such building blocks and its relation to the harmonised metamodel environment using conceptual graphs.

The objective of federation is to provide insight on the invisible behaviour within a modelling ecosystem that describes the information system in the current state or future evolution. Traces and dependencies between actors (human domain experts, functionalities as services) become accessible.

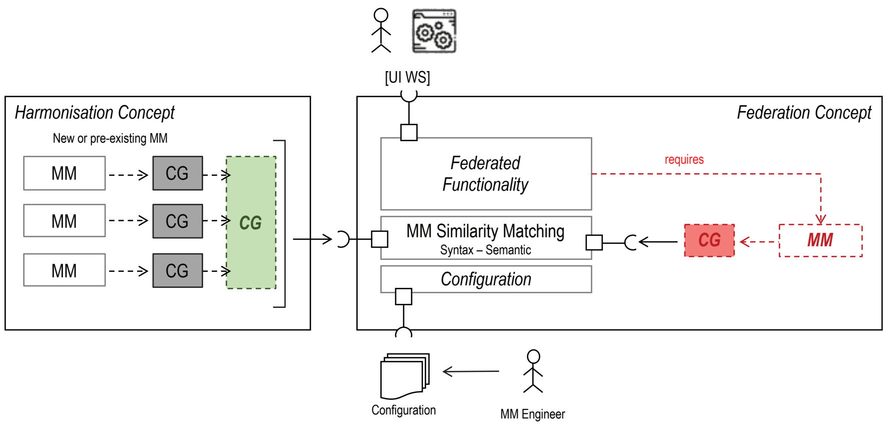


Figure 5.7: Anatomy of a Digital Intelligence Building Block

As a building block, the common structure has been identified based on the representation of metamodels as conceptual graphs. Building on the assumption that federated functionalities operates on a required structure, defined as a metamodel and formalised as a conceptual graph, the alignment of functionality to available metamodels is performed as a similarity matching function. Each building block consumes the harmonised conceptual graph during design time, matching is performed in line with the configuration defined for the building block. As an outcome the federated functionality is proposed and based on the intervention of the metamodel engineer, enacted within the ecosystem.

Federation techniques are enabled as domain specific building blocks upon the harmonised and aligned metamodel. They are required to be non-intrusive: the functionality defined in the federation building block is added on ecosystem level, rather than the individual metamodel. This is accomplished through listening to specific events in the ecosystem and deriving actions on federated level. The configuration of this listeners is the outcome of the design-time matching in the federated building block.

5.2.2 Operation of Digital Intelligence Building Blocks

The architecture in Fig. 5.7 is on type level, abstract elements, marked in italic, are instantiated for each specific federation approach realised. Each instantiation results in a micro-service that provides the interfaces (User Interface (UI) and web-services) specified and provides the functionality globally to the user. The instantiation architecture for the federation framework is discussed as a technological environment to embed specific type of intelligence services on a per case basis, support re-use and adaptation using suggestion mechanisms based on the design results of a metamodel.

The operation of these building blocks considers two phases: design as matching required structure and semantic and execution as listeners and federated functionality. The design phase identifies the adequate elements within the harmonised conceptual graph, including virtual relations and allows the metamodel engineer to map/configure the federated functionality upon this structure. The execution phase is responsible to instantiate listeners in operation and perform run the functionality accordingly. As a service-oriented paradigm is applied, functionality is independent of the underlying, operational metamodels and therefore applicable for more than one instantiation.

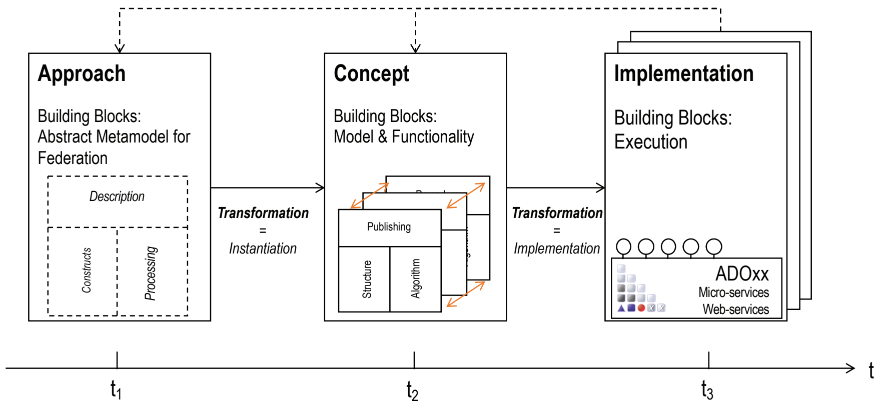


Figure 5.8: Federated Functionality as Metamodel Building Blocks based on (Utz, 2018b)

Fig. 5.8 describes the methodology to realise federated functionality using metamodel building blocks. The concept of such building blocks has been initially introduced in (Utz, Woitsch, Falcioni et al., 2018; Utz, 2018a), further refined and applied on design challenges in (Utz, 2018b) and used for the realisation of federated functionality in (Bork, H.-G. Fill et al., 2018). As a methodology, metamodelling using building blocks considers three phases, that are iteratively repeated and refined, as discussed in (Karagiannis, 2015) for AMME:

1. *Approach*: during this phase, the abstract metamodel for the federated functionality is defined. This metamodel is considered as the required elements and structure to be imposed upon the harmonised modelling ecosystem. The metamodel engineer defines the functionality descriptively, identifies the structure as constructs needed by the model processing/digital intelligence functionality of model processing,
2. *Concept*: transforming the approach into one or more concept blocks is considered an instantiation of the approach. Multiple outcomes are possible based on the environment, domain-specific characteristics and technology platform envisioned,
3. *Implementation*: transforms the concept building blocks into operational/executable building blocks. These blocks are defined as services that expose their functionality to the end-user via standard interfaces (web-services, micro-services) and accompanying user interfaces/interactions. As an indication, the ADOxx platform (ADOxx.org, 2020) is mentioned in Fig. 5.8 as an operational platform to execute these services.

Summary. The knowledge operation for the design of novel metamodel, harmonisation of existing ones in the context of a modelling ecosystem and intelligence operation using federation techniques are introduced in this chapter. An important aspect for building blocks realised using this approach is *reusability* and *configuration*: building blocks in any of the phases (Approach, Concept, Implementation) are generic in a sense that they can be further (re-)designed, conceptualised and implemented as additional instantiations of the original version. Re-use is support via alignment (required metamodel, projections, and binding) and configuration abilities as defined in the anatomy of digital intelligence building blocks.

This characteristics are reflected by the anatomy of metamodelling building blocks: the internal logic is exposed building upon the conceptual structure - for the required as well as provided elements. Matching is relevant on a structural/syntactical level (matching the abstract representation) but also on semantic level (matching the meaning of elements). The second part is regarded a challenge as similarity on semantic level is defined either following a close-world assumption (only the concept types available defined the system for match) or open-world techniques (matching linguistically the actual meaning). For open-world, canon-based techniques, the language-oriented viewpoint needs to consider a broad spectrum of definitions and lexical information, resulting in an extensive design decision space for the metamodel engineer.

The classification of functionalities has been performed as input for the types of virtual relations required within a modelling ecosystem. Three types have been identified from literature and application cases: a) querying information from various connected systems, b) information integration as power-relation between concepts (provider, supplier, value-interchange) and as a sub-category of querying, c) dependency analysis on a conceptual level.

The chapter concludes the specification of the harmonisation and federation concept for distributed metamodels and is applied within the technical realisation concept to define an architecture of building blocks that support the design process. These building blocks are defined as blue-prints in chapter 6 as input for a prototypical evaluation in chapter 7.

Chapter 6

Technical Realisation Concept

As a proof-of-concept for the above conceptual design and specification, a technological concept for realisation is discussed to evaluate the feasibility of the solution for the design of harmonised modelling ecosystem and the applicability of federated functionality for digital intelligence. The technological concept realised is coined "DeMoMa" as an abbreviation for "Design-Model-Make" methodology applied in the context of "smart models". The approach has been introduced by OMILAB (discussed in (Götzinger et al., 2016; Bork, Buchmann et al., 2019)) for the development of "intelligence offerings" defined in (Petry, 2019).

DeMoMa is defined as a metamodel design and implementation environment, whereas a core aspect of the prototype is on integration of support capabilities as an Integrated Development Environment (IDE). The foundation for the development environment, from a structural point of view, is on the conceptual graph implementing the CoChaCo grammar. From a functional point of view, extendable design support services are coupled by the IDE and become useful for the metamodel designer/engineer. DeMoMa consists (on a high abstraction level) of the two building blocks, introduced in detail in the following subsections:

- *DeMoMa::Harmony*: as a design component for harmonisation of metamodels. The component implements the concepts for metamodel design using conceptual graphs and integrates operations on the graph. This results in the capabilities to apply the design techniques defined in section 5.1.1 proposing a DSL implementation that enables the specification of metamodels or fragments following the grammar of CoChaCo, similarity matching functionality using a corpus- and knowledge-based approach, enable virtual relation definitions in a programmatic and graphical way. The component provides an interfaces to conceptual graph visualisations, verification/validation, query and deduction mechanisms, specifically adapted to the domain of metamodel design,

– *DeMoMa::Intel*: as a design and operationalisation component that enables the definition of digital intelligence building blocks (using the design capabilities of DeMoMa::Harmony), matching techniques and instantiation/deployment pipelines. As such, federated functionality can be realised and applied on the harmonised modelling ecosystem.

The realisation concept is input for the evaluation of the concepts on harmonisation and digital intelligence, introduced in section 7.1.

6.1 Architecture of DeMoMa

DeMaMo is characterised as a language-oriented approach to design metamodels, their harmonisation and align intelligence operations using a federated approach. As such, the architecture builds on the notion of a programmatic interaction of the designer with the tool. The metamodel designer applies a domain-specific language (DSL) for the design task and is supported through a graphical representation of the implementation results, visualising the underlying conceptual graph.

The two building blocks of the IDE are introduced as *DeMoMa::Harmony*, responsible to support design and harmonisation activities, and *DeMoMa-::Intel*, providing the capabilities to create, align and deploy intelligence functionalities. The building blocks are composed of modular submodules, that support specific challenges. Well-defined interfaces of these submodules allow for a dynamic extension, adaptation or replacement of functionality based on granular/domain-specific requirements of the engineer. General characteristics have been established in advance, to structure the implementation efforts for the prototype:

– *Standard technologies*: the prototype builds on standard technologies available in the field to allow for community uptake and modification. This is specifically relevant for interfaces within the building blocks. As a common practice, data is interchanged in Representational State Transfer (ReST) format, using JavaScript Object Notation (JSON) representations as a serialisation format,

– *Interfaces*: each module of the prototype defines input required and output produced as well-defined interfaces. The input and output

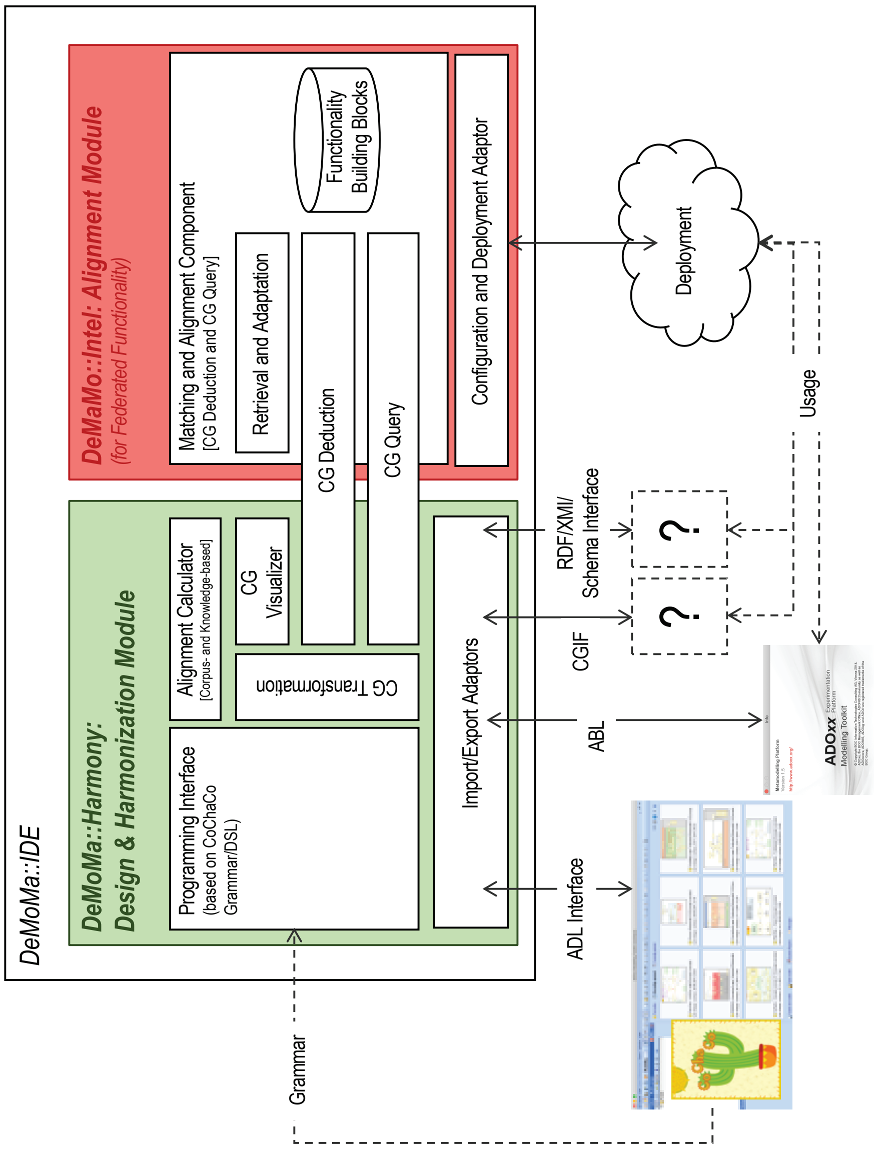

DeMoMa::IDE
DeMaMo::Harmony:
Design & Harmonization Module
Programming Interface
(based on CoChaCo Grammar/DSL)
Alignment Calculator
[Corpus- and Knowledge-based]
CG Visualizer
CG Transformation
Import/Export Adaptors
DeMaMo::Intel: Alignment Module
(for Federated Functionality)
Matching and Alignment Component
[CG Deduction and CG Query]
Retrieval and Adaptation
CG Deduction
CG Query
Functionality Building Blocks
Configuration and Deployment Adaptor
Deployment
Usage
RDF/XMI/
Schema Interface
CGIF
ABL
ADL Interface
Grammar
?
?
ADOxx
Modelling Toolkit

Figure 6.1: DeMoMa Logical Architecture (only external communication interfaces visualised)

definition is on format and syntax level, whereas emphasis is put on the use of standard formats, to overcome limitations of proprietary data formats,

– *Interaction as Orchestration*: user/developer interactions are flexible and understood as the result of a user-specific orchestration of module functionality. This characteristic targets the observation that the use of the prototype is not prescribed, but is adaptive to domain-specific needs.

The specification of each sub-module can be found in the following subsection. A common description format is used for each module. The template is proposed as an adapted version of IEEE 29148 standard (ISO et al., 2011) and Kruchten's Architectural Blueprints (4+1 model) in (Kruchten, 1995).

A. Purpose/Rationale	Defines the purpose and rational of the building block in relation with the overall architecture, describe architecture blueprints (e.g. interfaces).
B. Required Interfaces	Interfaces that the building block requires (internal or external).
C. Provided Interfaces	Interfaces that the building block provides to others (internal or external).
D. Functionality	Functionality and typical/foreseen usage pattern of the building block.
E. Submodules	List and references to submodules included, in case applicable.
F. Relevant Technologies	Technology assessment for the realisation of the component, defined and evaluated for main modules only.
G. Evaluation Criteria	Assessment and evaluation criteria based on (Prat et al., 2014) for the component (as input for the evolution of the prototype), defined and evaluated for main modules only.

Table 6.1: Technology Concept Modules: Specification Template

6.2 DeMoMa::IDE (Integrated Development Environment)

This module is defined as the overarching container of functionality realised for the prototype. It provides the user interface and binds the services/submodules specified.

A. Purpose/Rationale. As an integrated development environment, the metamodel design, harmonisation and extension (through intelligence operations) is supported. The integration aspect provides the engineer a single interaction point to run and trigger the functionalities needed for the design.

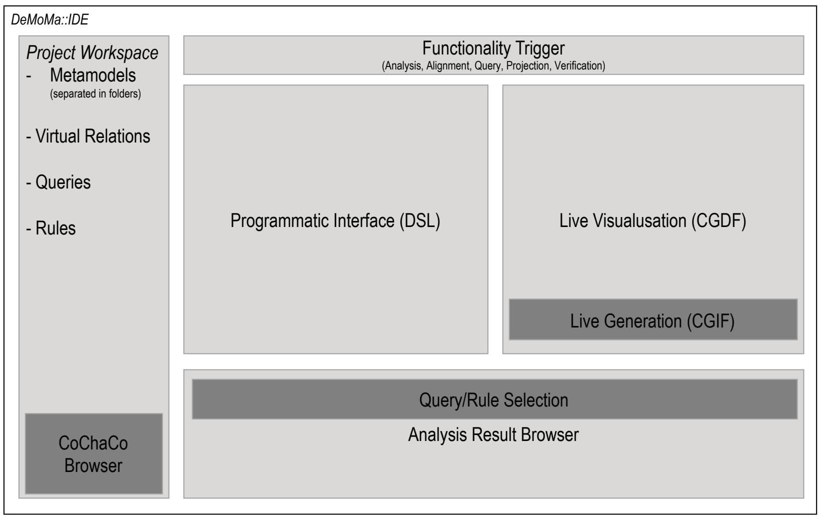


Figure 6.2: DeMoMa User Interface (Mockup)

Fig. 6.2 shows the mockup for the integrated development environment as a design study on user interaction. Each of the modules is dynamically added, the UI elements represent placeholders for the specific functionality to be embedded and offered through the UI. The conceptual architecture is shown graphically in Fig. 6.1.

B. Required Interfaces. The IDE consumes pre-existing metamodels through data adaptors (extract, load, transform), using standard representation for XML Metadata Interchange (XMI) and Resource Description Framework (RDF). It considers the CoChaCo environment as a common datastore and (implicitly) builds on the grammar defined based on CoChaCo in section 3.3.2.

C. Provided Interfaces. The environment provides interfaces to extract design artefacts, naturally in ADOxx Definition Language (ADL) format to exchange (load and store) metamodels with the CoChaCo environment. In addition, ADOxx Library Language (binary format) (ABL) format is produced directly as implementation skeletons for the ADOxx metamodelling platform and generic interfaces in XMI and RDF format.

D. Functionality. The metamodel engineer uses the IDE to apply the design techniques as defined in section 5.1.1. The functionality support the design process from early specification to extension and combination/alignment phases.

E. Submodules. The DeMoMa::IDE consists of logical modules:

– DeMoMa::Harmony (see section 6.3), and

– DeMoMa::Intel (see section 6.4).

F. Relevant Technologies. Technologies to realise IDEs are strongly related to the language support they offer. Typically an IDE consists of a code editor, a compiler and interpreter and build/deployment capabilities. In the field of DSL development, candidate technologies/platforms are ADOxx as a metamodelling platform, Eclipse (specifically the DSL workbench/EMF), MPS (Domain-Specific Language Creator by Jetbrains), Microsoft Visual Studio Code as an extendible, lightweight environment for programmers. (Visic and Karagiannis, 2014) provides an overview of technologies applicable for the development of a DSL.

G. Evaluation Criteria. The evaluation of the prototype, on a general level focuses on the criteria of *activity*, more specifically on the aspects of "consistency" (whether the prototype supports the consistent definition of harmonised metamodels for modelling ecosystems and "accuracy", interpreted as the support/utility function the implementation provides to the metamodel engineer with as adequate support in the design and development process.

6.3 DeMoMa::Harmony

DeMoMa::Harmony is responsible for the design and harmonisation of metamodels. The functionality offered in the submodule is related to the extraction (via interfaces), transformation and load of arbitrary metamodels into the system. Analytical support is provided to detect similarities between metamodels, propose virtual links and perform the alignment.

A. Purpose/Rationale. The module is concerned with the design of new, and alignment of existing metamodels. As such it covers the functionality needed to establish the notion of conceptual graphs for metamodels and provides the interaction access to verification, querying and deduction services. As a common foundation, the CoChaCo syntax is implemented and a programmatic interface with live visualisation (as serialisation into CGIF and CGDF) are provided.

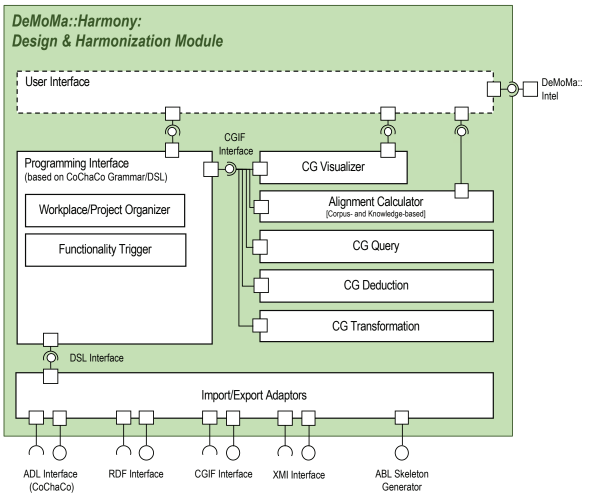


Figure 6.3: DeMoMa::Harmony Architecture

B. Required Interfaces. The module builds on the language interpretation of CoChaCo and its technical implementation as a DSL. Runtime interfaces are required to load metamodels and establish the structure within a project workspace. DeMoMa::Harmony is considered the "design environment" for metamodels that enables harmonisation.

C. Provided Interfaces. Interfaces are provided via adaptors and translators of metamodels in different formats, ADL-based repository persistence and for the federated functionality alignment module DeMoMa::Intel.

D. Functionality. The usage pattern depends on the availability of metamodels for harmonisation. The typical flow (in case of pre-existing metamodels) follows the interaction logic of load, extract, transform to setup the workplace, alignment and verification during the design phase.

1. Load and Extract: the metamodel is loaded through the user interface using a supported adaptor for extraction,
2. Transformation: The metamodel is made available in CoChaCo syntax, as a transformation following its extraction. The mapping to CoChaCo grammar is implemented in the specific adaptor. The transformation triggers the CoChaCo interface to persist it in the connected tool instance,
3. Workspace Organisation: step 1 and 2 are repeated for every metamodel considered in the ecosystem. The workspace is organised in folders (one folder per metamodel) and harmonisation elements (rules, queries, virtual relations),
4. Alignment: The metamodel engineer triggers the functionality as required. Typically a similarity matching is performed initially to get an impression of the relatedness of concept and relation types, followed by alignment steps derived from the similarities identified using pre-existing relation types (e.g. "is-a") or through virtual relation defined iteratively,
5. Verification: is performed continuously and is supported by a graphical view on the resulting conceptual graph and its validity,
6. Export and Compilation: the functionality triggers the compilation of the specific alignment service.

E. Submodules. DeMoMa::Harmony includes the following submodules:

- DSL Programming Interface (see 6.3.1)
- Alignment Calculator (see 6.3.2)
- Conceptual Graph Transformer (see 6.3.4)
- Conceptual Graph Visualizer (see 6.3.5)
- Import/Export Adaptors (see 6.3.3)

DeMoMa::Harmony shares the following submodules with DeMoMa::Intel:

- Conceptual Graph Query Interface (see 6.5.1)

– Conceptual Graph Reasoning Interface (6.5.2)

F. Relevant Technologies. For the specification of the module, technologies to establish a language-based DSL are considered relevant. This includes implementation the implementation of language workbenches introduced previously. Additionally, technologies for conceptual graphs contribute to the development.

G. Evaluation Criteria. Evaluation criteria selected for DeMoMa::Harmony are selected based on the purpose of the module. Targeting harmonisation, the criteria of *completeness* targets the evaluation of the functional coverage (can arbitrary metamodels be designed, operational metamodels be extracted and loaded), in line with the criteria of *usability* as a means to assess whether the results achieved are useful and comprehendible by metamodel engineers.

The direct submodules are introduced below, following the same template as for the main modules. Shared modules are discussed thereafter.

6.3.1 DSL Programming Interface (CoChaCo)

This module is responsible to provide the programmatic interface of the environment as the core interaction environment with the metamodel engineer. The DSL, defined by its grammar in section 3.3.2 and detailed in the annex A, is implemented. The module is responsible to organise the workspace for harmonisation, each metamodel is represented in distinct folder, created manually or established through an import via the CoChaCo modelling tool or other adaptors. Virtual relations, queries and rules are define following the same syntax and grammar. The internal representation of the DSL is based on the CGIF for interchange with functional components.

A. Purpose/Rationale. The module is responsible to load, edit and manipulate metamodels using the CoChaCo grammar and syntax defined conceptually in a programmatic manner. The interaction of the metamodel engineer is supported through syntax highlighting and auto- completion, aligned on one hand with the grammar and on the other the abstract vocabulary, continuously evolving during the programming task.

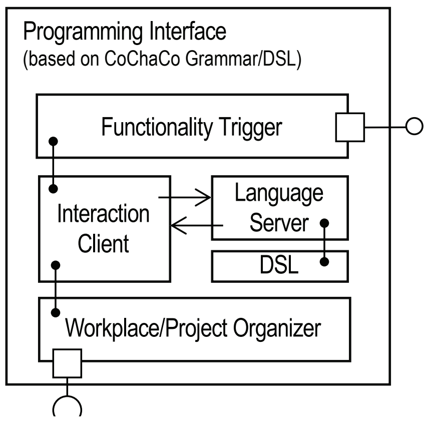


Figure 6.4: DSL Programming Interface (CoChaCo) Architecture

The purpose of the module relates to the conceptual definition and harmonisation as a baseline for federation concepts. The actual implementation is supported through the generation of platform specific skeletons.

B. Required Interfaces. Internally, the module builds on the language implementation and support mechanisms to support the programatic interaction, and organising the workspace of the metamodelling projects into building blocks that are coupled via the abstract vocabulary (template as a starting point for each project, including the CoChaCo concept/relation hierarchy and types of virtual relations, specialised during the development process).

A relevant interface required relates to import and export mechanisms, provided the Import/Export Adaptors module, detailed in section 6.3.3.

C. Provided Interfaces. The module provides interfaces to retrieve implementation results in CGIF format and DSL representation for a) functionality triggers and b) skeleton generation.

D. Functionality. The usage pattern for this module is to create a new workspace, and create new or import existing metamodels. This initialisation step is supported by the workspace organiser, responsible to harmonise the vocabulary based on the imports. During the implementation process the results are verified continuously syntactically and support functionality enables the validation of results before deployment and skeleton generation.

6.3.2 Alignment Calculator

The Alignment calculator is a support module to identify semantic relatedness between concepts of different metamodels (organised within the workspace) applying the two alternative approaches introduced earlier:

- Canon-based approach: connecting to open access services to calculate a similarity score between concept and relation types, or
- Knowledge-based approach: enabling custom interest functions to evaluate the closeness with respect to the vocabulary, relations and properties identified.

Both approaches result in a common visualisation as a matrix with rated similarity scores as a proposal for further action and definition.

A. Purpose/Rationale. The module is responsible to propose commonality and similarities between metamodels in focus. As such, it does not derive the matching automatically, but enables the metamodel engineer to select from various matching algorithms, and understand quickly a possible relatedness. As the matching is domain-specific, a flexible architecture is proposed for this module.

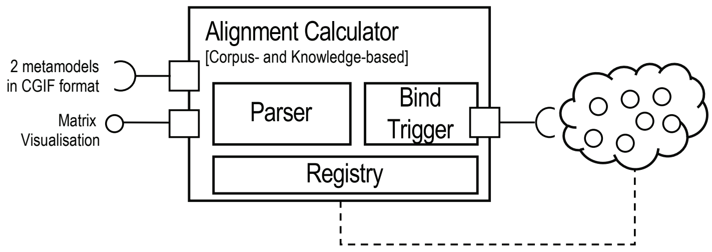


Figure 6.5: Alignment Calculator Architecture

The matching functionality is provided externally to the IDE as services that consume two metamodels in CGIF format and the corresponding vocabulary. The common service is to parse the structure as a two-dimensional matrix, and calculate/score the similarity. The calculation logic is embedded in the specific service that is registered in the IDE.

B. Required Interfaces. The module requires input from a content perspective, retrieving the metamodels in CGIF format, and the availability of matching services within the registry. The registry is defined as a store of all configured interfaces that can be triggered (endpoint, characteristics).

C. Provided Interfaces. The module provides interfaces to the IDE as the visualisation of calculation results in the form of a rated matrix (comparing concepts types or relation types of different metamodels).

D. Functionality. During the design of a the ecosystem consisting of multiple metamodels, the engineer triggers the functionality to identify potential similarity to be considered in the specific design or as virtual relations. The functionality offered by the module provides and indication of potential matching strategies to be applied.

6.3.3 Import/Export Adaptors

The Import/Export Adapters are concerned with the interaction between pre-existing results or fragments. Interfaces are foreseen on a generic level (to include results achieved using the CoChaCo modelling toolkit, directly mapping to the grammar and syntax) and standards level (capturing metamodels in standardised serialisation (e.g. as conceptual graphs, as schemas in RDF or XMI format. Additional adaptors can be dynamically added as graph-rewriting model transformations.

A. Purpose/Rationale. The module summarises and aggregates all adaptors relevant for the prototypical implementation. These adaptors transform input formats to the DSL representation for inclusion and provide skeleton exports to metamodelling platforms.

Transformation is performed through mapping rules defined for each source representation. Transformation rules are considered as model-to-model transformations, that follow a graph-based re-writing approach in a bi-directional manner.

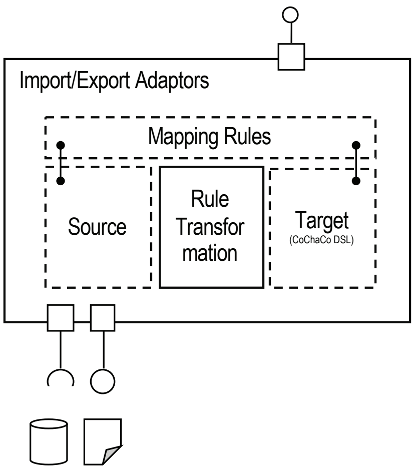


Figure 6.6: Import/Export Adaptors Architecture

B. Required Interfaces. The rule engine requires the source model as an input and has direct access to the target model as the CoChaCo grammar.

C. Provided Interfaces. The module provides its output (in CoChaCo representation) internally to the DSL Programming Interface and externally as output formats that can be generated to specific target formats. The generation of skeletons for implementations on platforms follows the same logical architecture.

D. Functionality. The module is used during the initialisation of the workspace to retrieve and import fragments via a database interface to the CoChaCo modelling tool or file-based for standard formats of metamodels.

6.3.4 Conceptual Graph Transformer

The Conceptual Graph Transformer as an internal component is responsible to translate the metamodel into CGIF format for further use within the modules. As a continuous background process, the DSL representation is translated into CGIF on any change that occurs in the definition.

A. Purpose/Rationale. The module acts in the background as an observer and generator of CGIF, based on the metamodel definition. Every artefact defined is translated in the CGIF logic. The approach follows the typical approach to semantically, syntactically sequence the input, format it and write it to the output format. The module also adapts the vocabulary as a common denominator of all design efforts on a common knowledge-base.

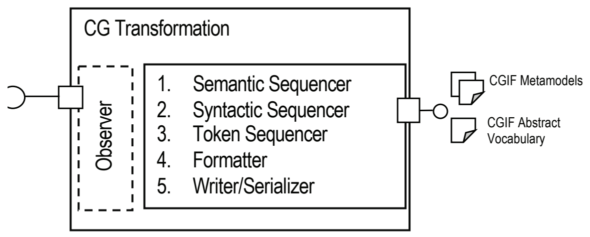


Figure 6.7: Conceptual Graph Transformer

The sequencing logic implemented as an observer pattern performs the generation efforts based on changes recognised. The generator works in two ways: a) create the output format in CGIF representation for each individual metamodel, and b) generate the abstract vocabulary as a single outcome of all metamodels within a project space.

B. Required Interfaces. The modules requires input from the programming interface and builds upon the available language server to identify changes and trigger the transformation following an observation pattern.

C. Provided Interfaces. The output is directly available in the workspace as CGIF files, that can be inspected by the metamodel engineer. These outputs are used for any design/elevation functionality for harmonisation and federated capability alignment.

D. Functionality. The use of the module is implicit and does not require any interaction by the user. This is achieved through the observation logic.

6.3.5 Conceptual Graph Visualizer

This module is responsible to provide a textual and graphical visualisations of the conceptual graph as an instrument to analyse, review the development process. Results from the analytical functionalities (e.g. query, deduction) are not only derived formally, but also visually, graphically shown via this module.

A. Purpose/Rationale. The module consumes the CGIF representation of the metamodels and generates a SVG visualisation from it, whereas the default notational aspects are considered (concepts as rectangles, relations as ellipses). The graphical view can be used for interaction (highlighting and synchronisation of the programming interface with the visualisation) and analysis (plotting result projections directly on the canvas. The visualisation builds on available 2D visualisation libraries.

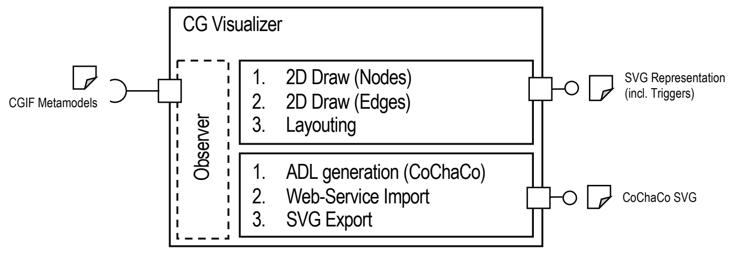


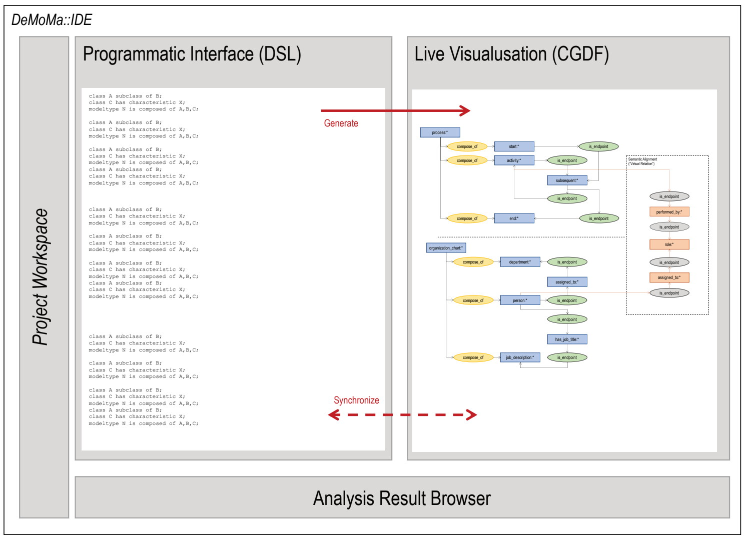
Figure 6.8: Conceptual Graph Visualizer

In addition, the graphical notation of CoChaCo is supported via the bridging adaptor to the ADOxx-based implementation.

B. Required Interfaces. The module requires the DSL representation as an input, and operates upon the language servers verification and generation results. The visualisation is embedded in the observer of the transformer and performed on changes in the definition.

C. Provided Interfaces. The module provides a graphical user interface to support the metamodel engineer in the design and implementation task. The observer of the Conceptual Graph Transformer triggers the graphical view generation in an asynchronous manner.

D. Functionality. As a support functionality, the visualisation of the conceptual graph is always available on the UI of the IDE. As the visualisation is triggered by the observer, the interaction is on to synchronise programmatic fragements with the visualisation (the visualisation can be used to understand which code fragment is concerned). In addition, color highlights support the cognitive level, marking different metamodels and virtual extensions.



```
class A subclass of B;
class C has characteristic X;
modeltype N is composed of A,B,C;

class A subclass of B;
class C has characteristic X;
modeltype N is composed of A,B,C;

class A subclass of B;
class C has characteristic X;
modeltype N is composed of A,B,C;
class A subclass of B;
class C has characteristic X;
modeltype N is composed of A,B,C;


class A subclass of B;
class C has characteristic X;
modeltype N is composed of A,B,C;

class A subclass of B;
class C has characteristic X;
modeltype N is composed of A,B,C;

class A subclass of B;
class C has characteristic X;
modeltype N is composed of A,B,C;
class A subclass of B;
class C has characteristic X;
modeltype N is composed of A,B,C;


class A subclass of B;
class C has characteristic X;
modeltype N is composed of A,B,C;

class A subclass of B;
class C has characteristic X;
modeltype N is composed of A,B,C;

class A subclass of B;
class C has characteristic X;
modeltype N is composed of A,B,C;
class A subclass of B;
class C has characteristic X;
modeltype N is composed of A,B,C;
```

Figure 6.9: Conceptual Graph Visualizer: Mockup

Figure 6.9 provides a mockup on the visualisation produced by the module.

In the previous section, the architecture on module level of the prototype has been discussed for those elements, that are exclusively assigned to DeMoMa::Harmony. In the next section, the overall architecture of DeMoMa::Intel and its constituting modules are introduced.

6.4 DeMoMa::Intel

DeMoMa::Intel is concerned with the design and enactment of intelligent capabilities within the development environment. As such it builds on the results of the harmonisation and provides engineering functionality to define, implement and deploy intelligence operations on the distributed modelling environment.

A. Purpose/Rationale. The module is responsible to propose, align and define intelligence functionalities through federation concepts on the harmonised modelling ecosystem. It supports the alignment of functionality, defined using the same means applied DeMoMa::Harmony (abstract metamodel fragments as conceptual graphs as functionalities) with the harmonisation results. A focus point of the module relates to enactment, as a deployment connector of the functionality in a distributed manner.

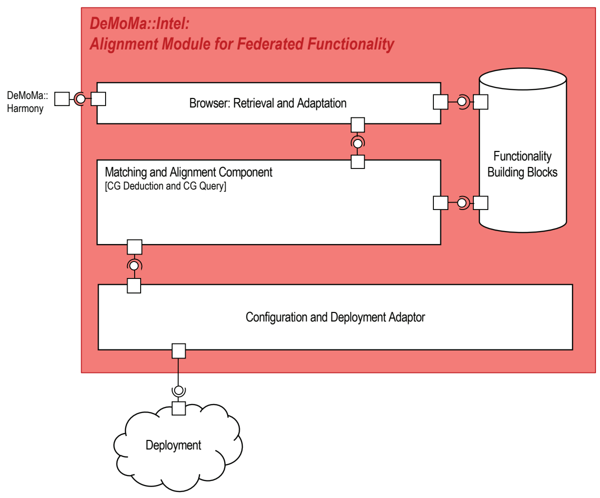

DeMoMa::Intel:
Alignment Module for Federated Functionality
DeMoMa:: Harmony
Browser: Retrieval and Adaptation
Matching and Alignment Component
[CG Deduction and CG Query]
Functionality Building Blocks
Configuration and Deployment Adaptor
Deployment

Figure 6.10: DeMoMa::Intel Architecture

The module is concerned with alignment capabilities; the implementation of these fragments is considered external to the module. This means that intelligence functionality can be realised using any available technology stack or deployment architecture, as long as the interface is of such services are accessible as a conceptual graph that represents the required metamodel structure and semantics. The approach has the advantage that any functionality can be interpreted as federated functionality for modelling ecosystem as soon as the header is defined and made accessible.

B. Required Interfaces. As input to the module, the metamodels and their semantic alignment from DeMoMa::Harmony are required. The input is provided through services exposing the metamodels and its combination as a conceptual graph in CGIF format. In addition, the intelligence capabilities are either designed directly in the module or retrieved as patterns from a functional building block store. The same format (CGIF) is used to represent the functionalities.

C. Provided Interfaces. The module provides interfaces to arbitrary deployment environments. The deployment is considered a configuration of pre-existing services that generically support intelligence mechanisms. The metamodel-based intelligence functionality is concerned with elevating these services with domain-specific semantics.

D. Functionality. The foreseen usage pattern of the module is based on the harmonisation. The designer system proposes potential candidates for inclusion based on the structure and semantic of the metamodels considered. In a next step the engineer iteratively adapts the concerned metamodels, its alignment relations or the configuration of the intelligence functionality. As soon as the results are satisfactory, the deployment interface is responsible to compose, configure and deploy the functionality as a service, in line with adaptors to be included in the harmonisation environment.

E. Submodules. DeMoMa::Intel includes the following submodules:

– Retrieval and Adaptation (see 6.4.2)

– Functional Building Blocks store (see 6.4.3)

– Configuration and Deployment Adapter (see 6.4.4)

DeMoMa::Intel shares the following submodules with DeMoMa::Harmony:

– Conceptual Graph Query Interface (see 6.5.1)

– Conceptual Graph Reasoning Interface (6.5.2)

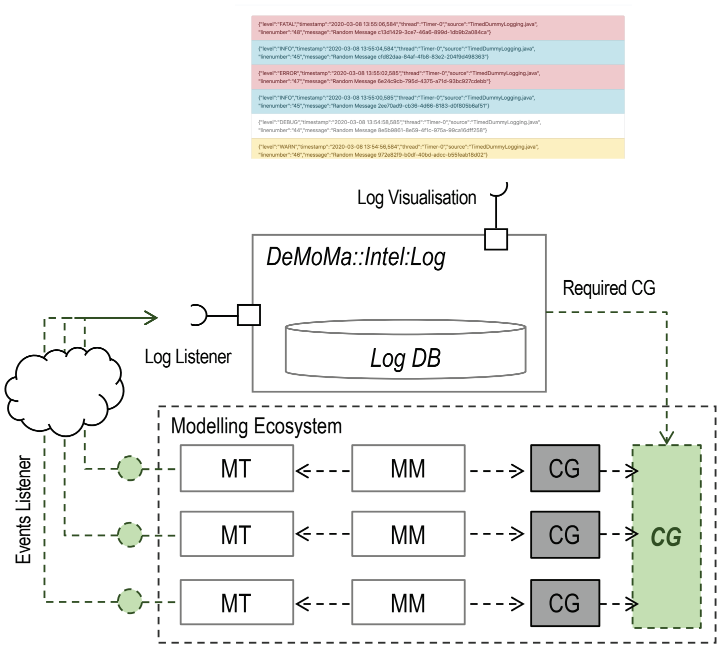
F. Relevant Technologies. The relevant technologies for the module are the same as for DeMoMa::Harmony, with the addition the enable deployment of intelligence services in a dynamic manner. Technologies for the configuration and deployment of micro-services are considered relevant in this field.

G. Evaluation Criteria. The module is evaluated with respect to its capability to define and deploy services. This is considered in line with (Prat et al., 2014) as an evaluation on the usage *environment*, more specifically as *consistency with technology*, as pre-existing technology is re-used for the purpose of intelligence services in distributed modelling environments.

6.4.1 Intelligence Services

Candidate intelligence services (in line with discussed approaches in 5.2 have been identified and implemented. These services are introduced below. Additional service would extend and validate the architectural principles introduces above for the module.

– **DeMoMa::Intel::Log** as general purpose logging functionality that reflects the alignment relations and provides a flat list of time-stamped logging entries. The pre-condition of this service is that metamodel constructs are a) instantiable and consequently b) modifiable through human or machine intervention. Every state change of a modelling element is considered relevant for logging as a behavioural or interaction-based change. The overall structure and semantics of the metamodels concerned is not relevant.



```
{"level":"FATAL","timestamp":"2020-03-08 13:55:06,584","thread":"Timer-0","source":"TimedDummyLogging.java",
"linenumber":"48","message":"Random Message c13d1429-3ce7-46a6-899d-1db9b2a084ca"}
{"level":"INFO","timestamp":"2020-03-08 13:55:04,584","thread":"Timer-0","source":"TimedDummyLogging.java",
"linenumber":"45","message":"Random Message cfd82daa-84af-4fb8-83e2-204f9d498363"}
{"level":"ERROR","timestamp":"2020-03-08 13:55:02,585","thread":"Timer-0","source":"TimedDummyLogging.java",
"linenumber":"47","message":"Random Message 6e24c9cb-795d-4375-a71d-93bc927cdebb"}
{"level":"INFO","timestamp":"2020-03-08 13:55:00,585","thread":"Timer-0","source":"TimedDummyLogging.java",
"linenumber":"45","message":"Random Message 2ee70ad9-cb36-4d66-8183-d0f805b6af51"}
{"level":"DEBUG","timestamp":"2020-03-08 13:54:58,585","thread":"Timer-0","source":"TimedDummyLogging.java",
"linenumber":"44","message":"Random Message 8e5b9861-8e59-4f1c-975a-99ca16dff258"}
{"level":"WARN","timestamp":"2020-03-08 13:54:56,584","thread":"Timer-0","source":"TimedDummyLogging.java",
"linenumber":"46","message":"Random Message 972e82f9-b0df-40bd-adcc-b55feab18d02"}
```

Figure 6.11: DeMoMa::Intel::Log Architecture

Interaction: Interaction behaviour with the intelligence service results are typically filtering and search mechanisms.
Configuration: configuration possibilities are proposed to visually highlight the foundational metamodel (distinguish the different metamodels in the harmonised environment) and "log levels" as semantically rich annotation of each entry produced by the service.
Visualisation: list formats, statistical assessment (charting) and time-series analysis using time-stamped, textual log entries. *Use Case*: Logging all interactions with the modelling environment, chronologically and visualise adaptations and their dependencies.

– **DeMoMa::Intel::Trace** as an elevation of the logging facility above considering traces as subsets of the logged information. Tracing defines a span (trigger to result) and combines the entries in a set of results that can directly be assigned to the trigger. As such, dependencies between metamodels are relevant and need to be reflected in the traces. This is accomplished through the virtual relation concept introduced earlier. Tracing therefore spans across the harmonised metamodels and enables the identification of impact of behaviour changes/model updates performed in one metamodel on the ecosystem.

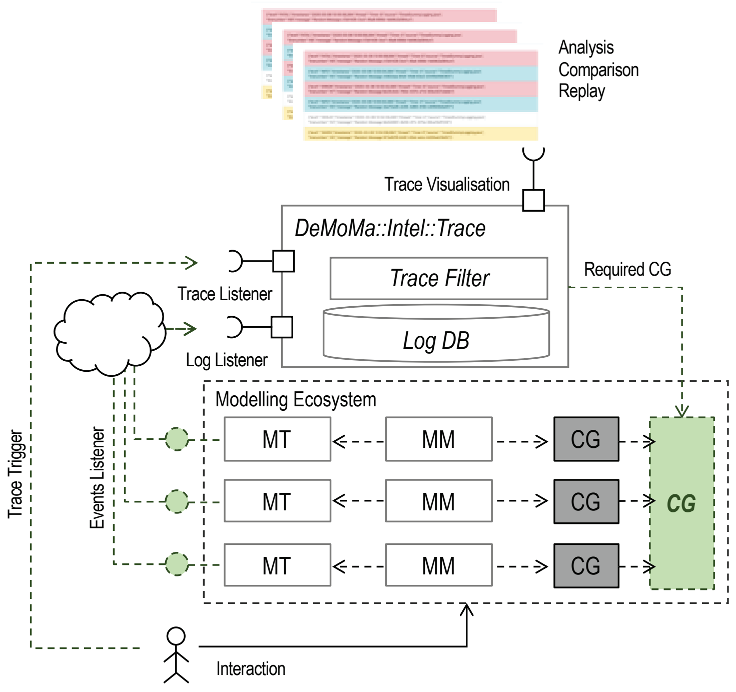


Figure 6.12: DeMoMa::Intel::Trace Architecture

Interaction: Traces are provided as a set of log entries, that is defined by the span they consider. The span is based on logical or temporal events defined by the engineer.
Configuration: in addition to the configuration of logging, spans are specified and configured. Different types are made explicit (e.g. calling, binding, composition)
Visualisation: Spans are typically visualised as directed graphs that represent the logical dependencies of an event and its impact on re-

lated activities. Visualisation support highlighting and animations as a replay functionality based on contextualised, time-stamped textual traces.
Use Case: Logging the effects of a certain modelling functionality on related elements (in the same or different metamodel)

– **DeMoMa::Intel::Impact** impact assessment provide the capability to elevate logs and traces (as subset of log entries) to become metamodel-aware. This implies that the intelligence service not only is aware of temporal or logical context information but also the underlying metamodels and virtual relations for harmonisation are considered and exposed to the intelligence mechanism.

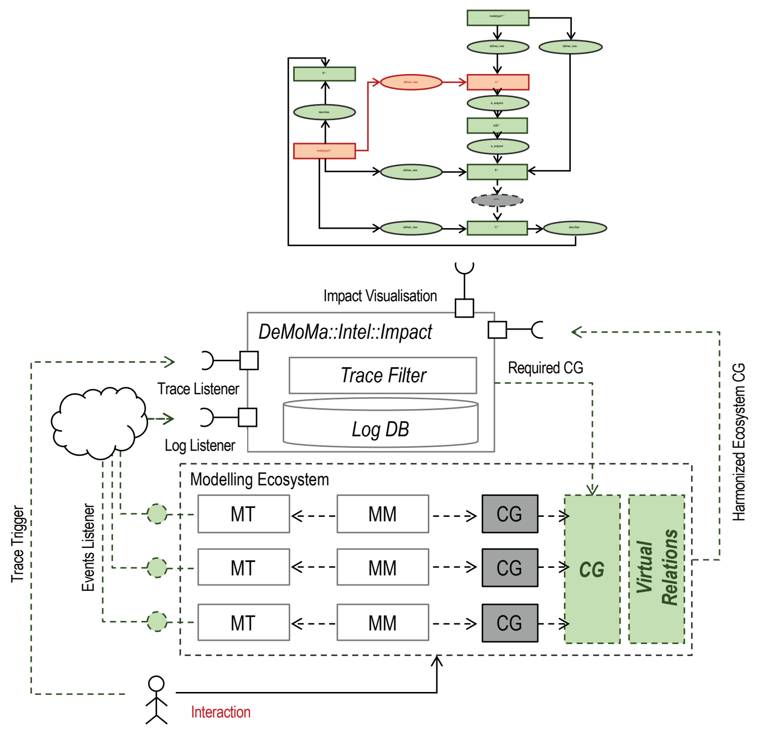


Figure 6.13: DeMoMa::Intel::Impact Architecture

Interaction: Interaction behaviour on model/instance level is set in the context of the harmonisation effort reflecting the underlying metamodels. Any interaction with a specific instance is analysed from a metamodel perspective, assessing and querying the harmonisation

space and returning the invisible dependencies in the ecosystem.
Configuration: The configuration of logging and tracing is used as a foundation, extended by the metamodels and the harmonisation results as conceptual graphs.
Visualisation: The modelling ecosystem is considered a network of concepts and relations in a combined manner. This network is used for visualisation purposes, highlighting visually the interactions and effects.
Use Case: The service considers the harmonisation results consistently and provides means for intelligence analysis. Network nodes as concept derived from the ecosystem can be queried and assessed.

6.4.2 Retrieval and Adaptation

The Retrieval and Adaptation module is the interaction and mitigation element between designed metamodels and applicable functional building blocks. The querying mechanism provided by the representation of metamodels (provided/required) as conceptual graphs supports the alignment task. Resulting projections provide input on the adaptation need required.

A. Purpose/Rationale. The module uses querying techniques and identifies candidate functional building blocks that can be included. The semantic annotation of each building block exposed is defined as the query, the harmonised metamodel as the knowledge base. Projections visualise the overlap for combination of federated functionality.

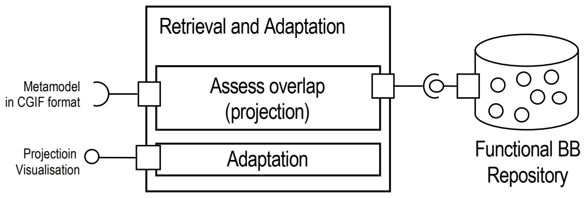


Figure 6.14: DeMoMa::Retrieval and Adaptation

B. Required Interfaces. The module requires, from a data and representation point of view, access to the conceptual graphs and abstract vocabulary of a) the building blocks and b) the design artefacts. Functionally, it tiggers a query and proposes adaptation actions.

C. Provided Interfaces. Adaptation requirements are provided as a result of the retrieval and query/matching mechanisms of the module. Based on the results, requirements can be assessed and mitigated.

D. Functionality. The metamodel engineer triggers a retrieval run by defining the scope and capabilities required. The repository is analysed and query mechanisms (inverse, the required metamodel structures of each building block is retrieved and assessed) provide a matching result.

6.4.3 Functional Building Blocks Repository

Intelligence functionalities are defined as modular building blocks that can be added upon newly developed or exiting metamodels. The purpose of the Functional Building Blocks Repository is to support the reuse of implementations and establish a platform for interchange. The term repository is used as an analogy to source code repositories is applicable as this module enables sharing, re-use and extension similar as in the open-source ecosystem for software engineering.

A. Purpose/Rationale. The module is built upon available source code management systems with extensions to semantically describe the building blocks committed to the repository in a user-friendly manner, including the conceptual graph required dynamically in the front-matter of the documentation files and machine readable to support search, retrieval, adoption or adaptation.

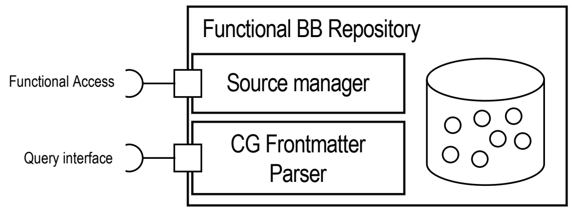


Figure 6.15: DeMoMa::Functional Building Blocks Repository

B. Required Interfaces. The module requires standardised mechanisms to push, pull add and commit changes from the IDE in a non-binary format. It retrieves and versions the artefacts internally.

C. Provided Interfaces. The Functional Building Block Repository expands upon the standard functionality of source code repositories as it provides an elevated interface for search and retrieval. This is achieved through annotation of a building block with the semantic context and required metamodel fragment of the building block with the front-matter of each artefact published. Search mechanisms assess the required structured and provide suggestions based on projection techniques of the front-matter towards the modelling ecosystem designed.

D. Functionality. Based on the development results of metamodels, the engineer can query the repository for adequate (syntactical and semantical fitting) building blocks. The proposed building blocks are further developed and adapted (either on building block level or within the metamodel ecosystem to cover the requirements. The resulting configuration is deployable and re-committed into the repository as a new version for further re-use.

6.4.4 Configuration and Deployment Adapter

The Configuration and Deployment Adapter is responsible to enact the building block in an operational environment. The building block is compiled, packaged and installed within the selected environment.

A. Purpose/Rationale. The Configuration and Deployment Adapter enables the packaging of a building block and its configuration and deployment. The functionality is realised using standard mechanisms for Continuous Integration (CI) and Continuous Deployment (CD). A build and deployment pipeline is defined in YAML Ain't Markup Language (YAML) syntax, executed upon commit of any new configuration made available.

B. Required Interfaces. Interfaces are needed to the metamodel compiler to expose adaptors defined by the federated functionality as the client and the deployment environment to configure, build and expose the functionality.

C. Provided Interfaces. The module provides interfaces in order to trigger and visualise the pipeline operation of configuration and deployment. Monitoring mechanisms report on the use and result of the functionality deployed.

D. Functionality. Following the functional configuration within the Retrieval and Adaptation module, the runtime parameters are defined. The pipeline for deployment is established and the functional building block is made available within the ecosystem.

6.5 DeMoMa::Shared Modules

DeMoMa::Shared Modules are available to both parts of the integrated development environment. They represent the core functionality with respect to functionalities enabled by the notion of conceptual graph. As generic modules, they operate and provide conceptual graph mechanisms for metamodel design.

6.5.1 Conceptual Graph Query Interface

The Conceptual Graph Query Interface realises query mechanismon structural and semantic representations of metamodels as conceptual graphs. The logical foundation enables this functionality. The module is implemented as a general purpose interface for querying conceptual graphs as a service.

A. Purpose/Rationale. The query interface is exposed as a state-less service that consumes the metamodel to be queried, the query graph and the abstract vocabulary. Alignment of the graph is established as all conceptual structures build upon a common abstract vocabulary including the virtual relations defined. Projections as an outcome of the query can be mapped on the inputs logically and graphically.

B. Required Interfaces. Access to the metamodels in CGIF format is required. This concerns the query base, the query itself and the abstract vocabulary that is embedded in the conceptual graphs.

C. Provided Interfaces. Projections as an outcome are returned in CGIF format. The textual representation (mappable to First Order Logic (FOL)) and graphical representation (as overlays on the query base) show the matching results.

D. Functionality. The functionality is applicable for a) matching of metamodel fragments (a fragment of a specific metamodel is extracted and defined as the query against the overall definition or another fragment) or b) matching functional building blocks (the required structure retrieved from the building block repository becomes the query, the metamodel design the query base)

6.5.2 Conceptual Graph Reasoning Interface

Reasoning on conceptual graphs supports the deduction of additional knowledge within the defined structure. The reasoning mechanisms provide the means to extend dynamically using reasoning rules. These rules are defined as required by the platform operating the solution (e.g. metamodel verification for skeleton creation) or the building blocks for federated functionality.

A. Purpose/Rationale. Rules defined for conceptual graphs enable the deduction of knowledge within the graph structure. These rules are based on the abstract vocabulary and formulated as if-then statements. The consequence is that the input graph is extended by the concepts derived by the rule definition. The purpose of this deduction for DeMoMa::* is to a) verify a metamodel composition and b) provide additional structural and semantic concepts/relations for processing.

Following the same strategy as for queries in section 6.5.1, the reasoning interface is implemented as a service, building on the CGIF representation of the conceptual graphs. Results of the reasoning/deduction can be stored and represent a homomorphic transformation of the conceptual graph. Available reasoning engine is re-purposed and adapted to the needs of the service functionality.

B. Required Interfaces. Access to the metamodels in CGIF format is required and are used as an input for reasoning. The rules are also represented in CGIF syntax and are applied by the reasoning engine in iteratively.

C. Provided Interfaces. The result of the reasoning are homomorphic conceptual graphs for each iteration of application. The resulting graph (for each step in the reasoning) is returned.

D. Functionality. The metamodel engineer triggers the functionality upon a query or alignment step to perform verification and semantic elevation based on the development results and selected building blocks (that expose rules for reasoning). Additionally, extension and reasoning is imposed by the design techniques specified in section 5.1.1 as well as the execution platform of each metamodel as a verification pattern.

The building blocks of the prototype have been introduced conceptually and logically above. This specification is used as input for the selection of appropriate technologies and the definition of the deployment/use architecture to evaluate the concepts established for harmonisation and federated functionalities in distributed modelling ecosystems. In the next section, the contributing technologies and their assessment as well as the deployment strategy is introduced.

6.6 Technology Assessment

The assessment of available technologies builds on the environment specification and building block structure. The following areas of technology relevant for the implementation are identified and assessed:

– *Integrated Development Environment*: The IDE acts as the umbrella of all building blocks and user interaction layer to provide effective usage patterns of design (harmonisation and federation) functionalities for the metamodel engineer. IDEs typically consist of code editors (including related technologies for code highlighting, auto-complete, syntax verification), compilation and build support and debugging facilities. Criteria for the selection of an adequate technology to realise the DeMoMa::IDE are established as: a) operating system support, b) openness to include modules and extensions, c) language independence and d) deployment support.

– *Language Workbench*: technologies for language-support are needed to implement the CoChaCo grammar and provide programming interactions to the metamodel engineer. Language workbenches enable grammar definition and realise code-highlighting, auto-complete and syntax verification. Evaluation criteria defined for the selection of language workbench technologies are a) functional coverage, b) generation capabilities and c) integration support.

– *Conceptual Graph Query/Reasoning*: Support of conceptual graph representation, transformation and functionality is required for querying and deduction functionality. Available technologies are assessed

– *Visual programming*: The programming approach build on classical coding, elevated with visual support for evaluation and assessment. The conceptual graph is presented in CGIF as well CGDF to provide the means for a visual inspection of dependencies and relation of harmonised metamodels.

– *Deployment support*: Continuous integration and deployment principles are required for an agile adaptation of experiments. Deployment is considered on two levels: skeleton generation for newly design metamodels and extension patterns for existing, operational metamodels.

The assessment of technologies available for the building blocks is discussed in annex B.

Summary. The technological realisation concept presented in this chapter establishes the architecture blue-print for the components defining the metamodel design space. The technical concept proposes an architecture (overall and per building block) to implement the conceptual design presented in chapter 2 3, 4. The technical concept defined clarifies the technical components and architectural consideration to implement an IDE for metamodel design, coined "DeMoMa::IDE". This IDE is considered the container of all implementations and services realised and classifies the services into building blocks of the architecture.

Feasibility of the architecture is assessed during the evaluation: prototypical implementation of selected building blocks aim to demonstrate the applicability of the functionality foreseen. The architectural principles guarantee openness for extension and domain-specific adaptation of existing service functionality.

Both the conceptual design and technical realisation architecture are input to the evaluation approach discussed in chapter 7. Prototypes realised are tested twofold: for completeness with respect to pre-existing metamodels utilising a code base of operational metamodels and environmental evaluation presenting a concrete, domain-specific implementation case within a research project.

Chapter 7

Evaluation

The evaluation approach of research results developed is based on the research objectives defined in section 1.4, following the guiding research question on an adequate knowledge representation for metamodels to support harmonisation and intelligence functionality alignment. The evaluation dimensions are introduced below, based on the criteria-set defined in (Prat et al., 2014, p. 5):

1. *Structural Evaluation - Completeness and Consistency*: the evaluation dimensions target whether the conceptual design proposed is complete to represent metamodels and is applicable in development/implementation projects. The criteria is assessed through the community results established in the OMiLAB as an indication whether the conceptual graph and CoChaCo mapping is adequate for metamodels of different domains and level of abstraction.
2. *Environment Evaluation - Consistency with Technology*: targeting the use of technology to realise the concept technically in a prototypical implementation case derived from a research project,

The evaluation results discussed below have been published and presented in conferences and journals initially, to include feedback from the scientific community during the early stages of concept development. The feedback received is mentioned in the sections below and references to the design outcomes are listed.

7.1 Structural Evaluation: OMiLAB Metamodels

The structural evaluation aims to assess whether the design concept for metamodel using conceptual structures is applicable and can be used to represent metamodels defined in research, academia and industry. For the evaluation of this criteria, the metamodels developed in the OMiLAB at the University of Vienna are used as the evaluation set, applying the concept developed experimentally on these metamodels and applying the prototypical implementation.

As an initial evaluation steps, that has fed back into the conceptual design, the available metamodels (designed and implemented) in various projects by the community have been assessed. The purpose of this evaluation is to clarify whether the grammar developed and its transformation/mapping on conceptual structure is adequate to represent the syntax and semantic of the implementation results and enables an analytical assessment and value-adding functionalities via similarity matching and virtual relations. As a scope of this evaluation case, the transformation rules are explicated and a repository of metamodels as conceptual graphs is established for further evaluation steps.

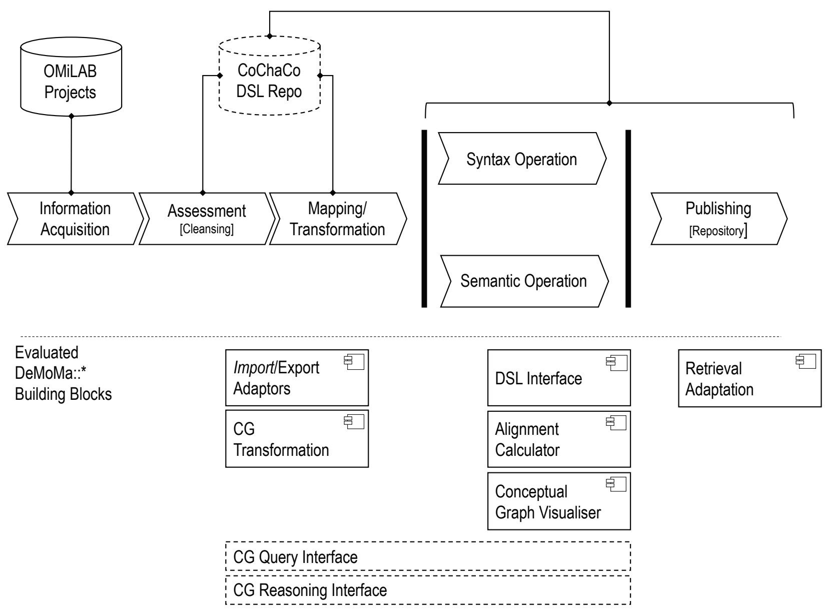


Figure 7.1: Structural Evaluation Process

In the following the steps performed, graphically shown in Fig. 7.1 for the laboratory evaluation are detailed. Insights gained during the steps are discussed and relevance to the concept developed is introduced.

7.1.1 Information Acquisition

In an initial collection step, all implementation results have been assessed and downloaded. The OMiLAB at the University of Vienna provides a dedicated project space for the modelling method design, formalisation, implementation and deployment. The projects are listed publicly by the community for any interested party at `https://austria.omilab.org/psm/exploreprojects`. Each project is structured in a common way: background information and details on the modelling method are described textually, related publications by or relevant for the project owners are referenced, project members and contact details are listed and the download of implementation results is possible. The implementation results are provided as a deployable tool and/or a method configuration file.

7.1.2 Assessment

The assessment of implementation results has been performed in two iterations. An early technical assessment did evaluate the applicable of the research idea, established a first technical concept for the transformation and detailed the grammar specification. The second iteration completed the set of modelling methods available in the lab and applied for evaluation.

The set of modelling methods used for the evaluation is established based on the download packages available. A limitation for this selection relates to the formalisation, that is ready-made available as these modelling methods (and their metamodels) are already available as implementations. For the design process and its decisions following AMME, discussed in and graphically shown in Fig. 2.4, only assumption could have been made. Nevertheless for the purpose of the evaluation cycles presented, the scope relates to the completeness criteria of the grammar established (ChoChaCo and mapping towards the notion of conceptual graphs).

The assessment (as a scoping of available resources) and download resulted in 50 application libraries (implemented on ADOxx) as input for the evaluation (intermediary results are presented in Appendix C see Table C.1, C.2 and C.3).

7.1.3 Mapping and Transformation

The mapping and transformation logic towards the CoChaCo grammar as presented in section 3.3.2 has been evaluated on the set of modelling methods established. The transformation is based on the implementation artefacts (downloaded and assessed in the previous phase) in the form of ABL files. The input are the binary representation of ADOxx Library Language (ALL) files. The import/export building block provides the transformation logic. Following an initial decryption (from ABL to ALL) using a Java based implementation of the meta^2-model of ADOxx, the content of the ALL files is parsed and transformed towards grammar specified. During the binary decryption, 3 libraries did not provide the necessary formality (due to an outdated implementation and have been skipped - see Table C.2, resulting in 47 implementation artefacts as the evaluation base.

The transformation/decryption of the ABL files, performed using their Java Object Model and serialisation resulted in the following quantification:

Criteria	**Quantification**
Number of ALL files	47
Lines of code in ALL files	1064395
External resources parsed (as files)	3927
Lines of code in external files	129538

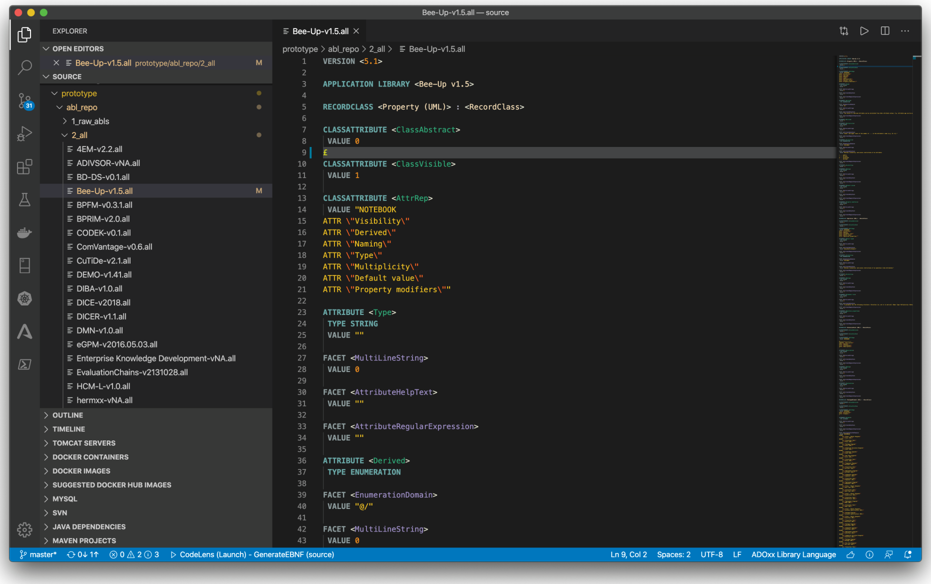
Table 7.1: Overview: ALL Code Base for Evaluation

A detailed analysis of the code base for evaluation is available in Table C.4 in appendix C.

Decrypt ABL Evaluation Code Base

The transformation uses the ADOxx Java Object Model, parsing the ABL files and decrypting them into ALL files. The screenshot shown in Fig. 7.2 shows the results in Microsoft Visual Studio Code; syntax highlighting is enabled by the AdoScript Extension.

The results of the transformation are available in the GitLab project space of this dissertation thesis at `https://gitlab.dke.univie.ac.at/mm_conceptual_graphs/all-repository`. Additional import/export service implementation are available in the source code repository at `https://gitlab.dke.univie.ac.at/mm_conceptual_graphs/import_export`.



```
VERSION <5.1>

APPLICATION LIBRARY <Bee-Up v1.5>

RECORDCLASS <Property (UML)> : <RecordClass>

CLASSATTRIBUTE <ClassAbstract>
 VALUE 0
£
CLASSATTRIBUTE <ClassVisible>
 VALUE 1

CLASSATTRIBUTE <AttrRep>
 VALUE "NOTEBOOK
ATTR \"Visibility\"
ATTR \"Derived\"
ATTR \"Naming\"
ATTR \"Type\"
ATTR \"Multiplicity\"
ATTR \"Default value\"
ATTR \"Property modifiers\""

ATTRIBUTE <Type>
 TYPE STRING
 VALUE ""

FACET <MultiLineString>
 VALUE 0

FACET <AttributeHelpText>
 VALUE ""

FACET <AttributeRegularExpression>
 VALUE ""

ATTRIBUTE <Derived>
 TYPE ENUMERATION

FACET <EnumerationDomain>
 VALUE "@/"

FACET <MultiLineString>
 VALUE 0
```

Figure 7.2: Screenshot: ALL in DSL Programming Interface

Evaluation of ABL Import/Export Service: The import/export service has been developed based on a Java-based transformation logic, applying rules to transform implementation artefacts from a compiled version to a textual representation. The service performs this task in an efficient manner and provides meta-information on the implementation artefact during the transformation as statistics.

Transformation to CoChaCo Grammar

In a second step, the code base has been transformed into the CoChaCo Grammar developed as part of this thesis. The representation is generic with respect to the underlying platform. This implies that dedicated mappings have to be established and implemented as rules.

The mapping logic for the transformation from ALL syntax into CoChaCo grammar is detailed in the following table:

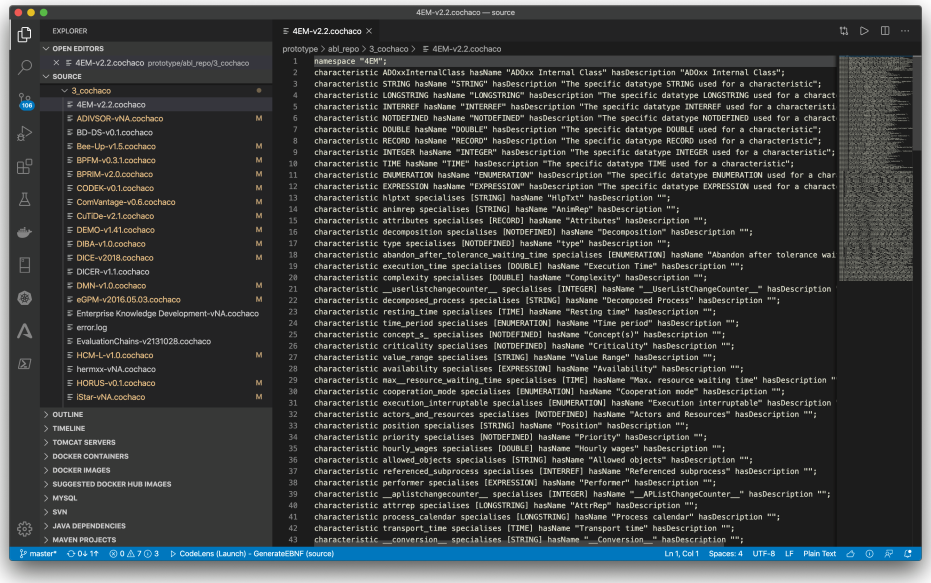
ALL Language Construct	CoChaCo Grammar
CLASS <ModellingClass> : <SuperClass>	concept <classID> specialises [SuperClass] hasName <ModellingClass> has [characteristics];

ALL Language Construct	**CoChaCo Grammar**
RELATIONCLASS <RelationClass> FROM <ModellingClass> TO <ModellingClass>	connector <relationID> specialises [] hasName <RelationClass> fromEndpoint <classID> toEndpoint <classID>;
MODELTYPE <ModelType> INCL "ModellingClass" INCL "RelationClass"	concept <mtID> specialises [] hasName <ModelType> aggregates [classID, relationID];
ATTRIBUTE <AttributeName> TYPE <AttributeType>	characteristic <AttributeType> characteristic <attrID> specialises [AttributeType] hasName <AttributeName>;
ITEM "Functionality" {context ::= menu, notebook}	functionality <context>; functionality <funcID> specialises [context] hasName <Functionality> uses [mtID, classID, relationID, attrID, funcID];
ON_EVENT "EventType" {context ::= event}	functionality <context>; functionality <funcID> specialises [context] hasName <Functionality> has <EventType> uses [mtID, classID, relationID, attrID, funcID];

Table 7.2: Mapping: ALL Syntax to CoChaCo Grammar

This mapping reflect specifics of the ALL syntax and the grammar defined. A trivial reflection relates to the namespace of objects within an implementation: as valid IDs are required by the grammar, the "hasName" element is introduced. This means that an ID is generated based on the object name in the ALL file; the "hasName" element holds the readable name. In addition, a "namespace" element is introduced that supports the identification of a library and provides input to the workspace organiser (distinguish objects with the same name in different implementations).

Evaluation of CoChaCo Transformation Service: An observation relates to the capabilities of specialisation and generalisation of objects. In ADOxx, classes are specialised (single superclass), whereas CoChaCo allows the specialisation of multiple superclasses (conceptually, needs to be resolved in implementation for a concrete implementation platform) for any type of element (relations, characteristics). This is visualised in the mapping table as the set of superclass is either 1 (for class concepts) or empty (for all others) derived from the ALL syntax.



```
namespace "4EM";
characteristic ADOxxInternalClass hasName "ADOxx Internal Class" hasDescription "ADOxx Internal Class";
characteristic STRING hasName "STRING" hasDescription "The specific datatype STRING used for a characteristic";
characteristic LONGSTRING hasName "LONGSTRING" hasDescription "The specific datatype LONGSTRING used for a charact
characteristic INTERREF hasName "INTERREF" hasDescription "The specific datatype INTERREF used for a characteristi
characteristic NOTDEFINED hasName "NOTDEFINED" hasDescription "The specific datatype NOTDEFINED used for a charact
characteristic DOUBLE hasName "DOUBLE" hasDescription "The specific datatype DOUBLE used for a characteristic";
characteristic RECORD hasName "RECORD" hasDescription "The specific datatype RECORD used for a characteristic";
characteristic INTEGER hasName "INTEGER" hasDescription "The specific datatype INTEGER used for a characteristic";
characteristic TIME hasName "TIME" hasDescription "The specific datatype TIME used for a characteristic";
characteristic ENUMERATION hasName "ENUMERATION" hasDescription "The specific datatype ENUMERATION used for a char
characteristic EXPRESSION hasName "EXPRESSION" hasDescription "The specific datatype EXPRESSION used for a charact
characteristic hlptxt specialises [STRING] hasName "HlpTxt" hasDescription "";
characteristic animrep specialises [STRING] hasName "AnimRep" hasDescription "";
characteristic attributes specialises [RECORD] hasName "Attributes" hasDescription "";
characteristic decomposition specialises [NOTDEFINED] hasName "Decomposition" hasDescription "";
characteristic type specialises [NOTDEFINED] hasName "type" hasDescription "";
characteristic abandon_after_tolerance_waiting_time specialises [ENUMERATION] hasName "Abandon after tolerance wai
characteristic execution_time specialises [DOUBLE] hasName "Execution Time" hasDescription "";
characteristic complexity specialises [DOUBLE] hasName "Complexity" hasDescription "";
characteristic __userlistchangecounter__ specialises [INTEGER] hasName "__UserListChangeCounter__" hasDescription
characteristic decomposed_process specialises [STRING] hasName "Decomposed Process" hasDescription "";
characteristic resting_time specialises [TIME] hasName "Resting time" hasDescription "";
characteristic time_period specialises [ENUMERATION] hasName "Time period" hasDescription "";
characteristic concept_s_ specialises [NOTDEFINED] hasName "Concept(s)" hasDescription "";
characteristic criticality specialises [NOTDEFINED] hasName "Criticality" hasDescription "";
characteristic value_range specialises [STRING] hasName "Value Range" hasDescription "";
characteristic availability specialises [EXPRESSION] hasName "Availability" hasDescription "";
characteristic max__resource_waiting_time specialises [TIME] hasName "Max. resource waiting time" hasDescription "
characteristic cooperation_mode specialises [ENUMERATION] hasName "Cooperation mode" hasDescription "";
characteristic execution_interruptable specialises [ENUMERATION] hasName "Execution interruptable" hasDescription
characteristic actors_and_resources specialises [NOTDEFINED] hasName "Actors and Resources" hasDescription "";
characteristic position specialises [STRING] hasName "Position" hasDescription "";
characteristic priority specialises [NOTDEFINED] hasName "Priority" hasDescription "";
characteristic hourly_wages specialises [DOUBLE] hasName "Hourly wages" hasDescription "";
characteristic allowed_objects specialises [STRING] hasName "Allowed objects" hasDescription "";
characteristic referenced_subprocess specialises [INTERREF] hasName "Referenced subprocess" hasDescription "";
characteristic performer specialises [EXPRESSION] hasName "Performer" hasDescription "";
characteristic __aplistchangecounter__ specialises [INTEGER] hasName "__APListChangeCounter__" hasDescription "";
characteristic attrrep specialises [LONGSTRING] hasName "AttrRep" hasDescription "";
characteristic process_calendar specialises [LONGSTRING] hasName "Process calendar" hasDescription "";
characteristic transport_time specialises [TIME] hasName "Transport time" hasDescription "";
characteristic __conversion__ specialises [STRING] hasName "__Conversion__" hasDescription "";
```

Figure 7.3: Screenshot: CoChaCo DSL Programming Interface

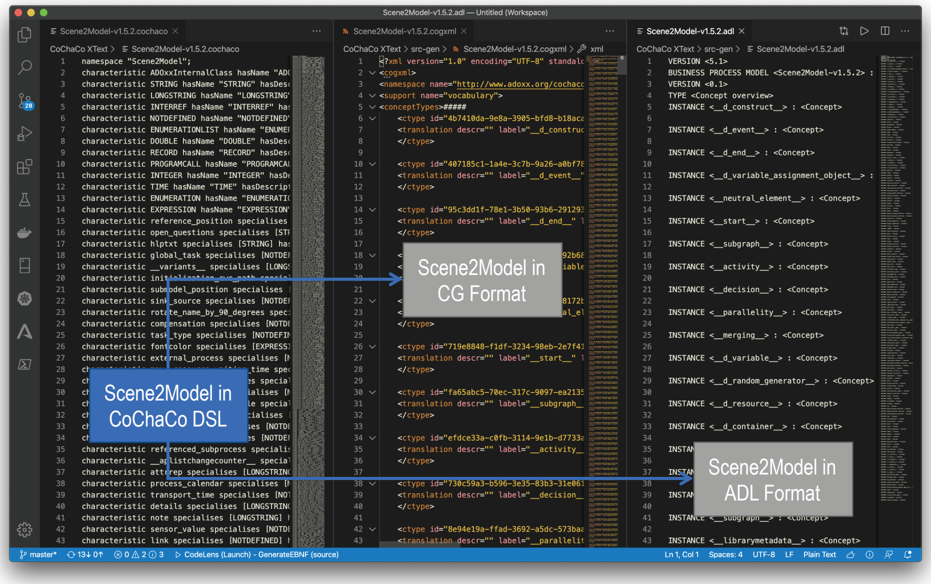
Conceptual Graph Transformation:

The transformation towards conceptual graph builds on a mapping of the grammar towards the constructs available in the abstract vocabulary. Using an observer pattern, the generator is triggered on every change of the metamodel. As a result two outcomes are created on valid metamodels autonomously. The generator for CGIF, as a basis for any knowledge operations, uses the grammar defined as an input and implements the listener to create the output formats. The formats realised are a) CGIF using the syntax established by the CoGUI implementation and b) CoChaCo ADL (for the visualisation of implementation results within the CoChaCo4ADOxx Modelling Toolkit). The mapping to CFIF is presented formally in section 4.2.

1. *Conceptual Graph in CGIF Format*: the metamodel is translated into an extended abstract vocabulary, utilising the "specialised" relation to define the concept and relation type hierarchy. In parallel, the all other relations are instantiated in the graph as concepts and relations and define the baseline for any syntactical or semantical knowledge operation on the metamodel. The output is compatible with existing conceptual graph technologies and can be imported directly.

2. *ADL Format in CoChaCo4ADOxx Format*: the same translation logic is applied to generate an ADL file that is compatible with CoChaCo4-ADOxx and can be imported in the tool. This transformation is required to provide a consistent graphical notation of the metamodel. The output is in textual format defining the conceptual hierarchy model and metamodel as a linked knowledge representation.

The source code fragment for the transformation is available in annex B, code listing B.2. Additional transformation service implementation are available in the source code repository at `https://gitlab.dke.univie.ac.at/mm_conceptual_graphs/transformation`



Scene2Model-v1.5.2.adl — Untitled (Workspace)
Scene2Model-v1.5.2.cochaco
CoChaCo XText > Scene2Model-v1.5.2.cochaco
Scene2Model-v1.5.2.cogxml
CoChaCo XText > src-gen > Scene2Model-v1.5.2.cogxml > xml
Scene2Model-v1.5.2.adl
CoChaCo XText > src-gen > Scene2Model-v1.5.2.adl
Scene2Model in CoChaCo DSL
Scene2Model in CG Format
Scene2Model in ADL Format
master* 13↓ 0↑ ⊗ 0 ⚠ 2 ⓘ 3 ▷ CodeLens (Launch) - GenerateEBNF (source)
Ln 1, Col 1 Spaces: 4 UTF-8 LF Plain Text

```
namespace "Scene2Model";
characteristic ADOxxInternalClass hasName "ADO
characteristic STRING hasName "STRING" hasDesc
characteristic LONGSTRING hasName "LONGSTRING"
characteristic INTERREF hasName "INTERREF" has
characteristic NOTDEFINED hasName "NOTDEFINED"
characteristic ENUMERATIONLIST hasName "ENUMER
characteristic DOUBLE hasName "DOUBLE" hasDesc
characteristic RECORD hasName "RECORD" hasDesc
characteristic PROGRAMCALL hasName "PROGRAMCAL
characteristic INTEGER hasName "INTEGER" hasDe
characteristic TIME hasName "TIME" hasDescript
characteristic ENUMERATION hasName "ENUMERATIO
characteristic EXPRESSION hasName "EXPRESSION"
characteristic reference_position specialises
characteristic open_questions specialises [STR
characteristic hlptxt specialises [STRING] has
characteristic global_task specialises [NOTDEF
characteristic __variants__ specialises [LONGS
characteristic initialization_cwc_path specia
characteristic submodel_position specialises
characteristic sink_source specialises [NOTDEF
characteristic rotate_name_by_90_degrees speci
characteristic compensation specialises [NOTDE
characteristic task_type specialises [NOTDEFIN
characteristic fontcolor specialises [EXPRESSI
characteristic external_process specialises [N
ch                           _time spec
ch                           es specia
ch                           ialises [N
ch                           le specia
ch                           cialises
ch                           ses [NOTDE
ch                           es [NOTDEF
characteristic referenced_subprocess specialis
characteristic __aplistchangecounter__ special
characteristic attrrep specialises [LONGSTRING
characteristic process_calendar specialises [N
characteristic transport_time specialises [NOT
characteristic details specialises [LONGSTRING
characteristic note specialises [LONGSTRING] h
characteristic sensor_value specialises [NOTDE
characteristic link specialises [NOTDEFINED] h
```

```
<?xml version="1.0" encoding="UTF-8" standalo
<cogxml>
<namespace name="http://www.adoxx.org/cochaco
<support name="vocabulary">
<conceptTypes>#####
    <ctype id="4b7410da-9e8a-3905-bfd8-b18aca
    <translation descr="" label="__d_construc
    </ctype>

    <ctype id="407185c1-1a4e-3c7b-9a26-a0bf78
    <translation descr="" label="__d_event__"
    </ctype>

    <ctype id="95c3dd1f-78e1-3b50-93b6-291293
    <translation descr="" label="__d_end__" l
    </ctype>

    <                                   92b68
    <                                   iable


    <                                   8172b
    <                                   al_el
    </ctype>

    <ctype id="719e8848-f1df-3234-98eb-2e7f41
    <translation descr="" label="__start__" l
    </ctype>

    <ctype id="fa65abc5-70ec-317c-9097-ea2135
    <translation descr="" label="__subgraph_
    </ctype>

    <ctype id="efdce33a-c0fb-3114-9e1b-d7733a
    <translation descr="" label="__activity_
    </ctype>

    <ctype id="730c59a3-b596-3e35-83b3-31e061
    <translation descr="" label="__decision_
    </ctype>

    <ctype id="8e94e19a-ffad-3692-a5dc-573baa
    <translation descr="" label="__parallelit
```

```
VERSION <5.1>
BUSINESS PROCESS MODEL <Scene2Model-v1.5.2> :
VERSION <0.1>
TYPE <Concept overview>
INSTANCE <__d_construct__> : <Concept>

INSTANCE <__d_event__> : <Concept>

INSTANCE <__d_end__> : <Concept>

INSTANCE <__d_variable_assignment_object__> :

INSTANCE <__neutral_element__> : <Concept>

INSTANCE <__start__> : <Concept>

INSTANCE <__subgraph__> : <Concept>

INSTANCE <__activity__> : <Concept>

INSTANCE <__decision__> : <Concept>

INSTANCE <__parallelity__> : <Concept>

INSTANCE <__merging__> : <Concept>

INSTANCE <__d_variable__> : <Concept>

INSTANCE <__d_random_generator__> : <Concept>

INSTANCE <__d_resource__> : <Concept>

INSTANCE <__d_container__> : <Concept>

INSTAN

INSTAN

INSTAN

INSTANCE <__subgraph__> : <Concept>

INSTANCE <__librarymetadata__> : <Concept>
```

Figure 7.4: Screenshot: CoChaCo Grammer CG Transformation

Evaluation of CG Transformation Service: the transformation of ALL to CoChaCo has been performed using the textual representation of the library and its Java Object Model. For those aspects that have been covered through parsing in the object model, no deficiency could be observed. A challenge observed, relates to the abstract metamodel of the platform and its use within the implementation projects. This common abstract metamodel provides platform functionality through inheritance/specialisation of structural elements. Nevertheless, only if specifically used and redefined, these concepts are also explicitly available in the textual representation and could be used in the transformation. Therefore, additional escalation and mitigation logic is required to overcome this limitation, specifically for platform-specific abstract relation types that are not redefined during the implementation. This situation occurs when the metamodel engineer builds his/her own metamodel, independent of the available constructs provided by the platform, including all functionalities within the system. An indication and assessment of these metamodel is performed as a structural knowledge operation query.

7.1.4 Knowledge Operations for Metamodelling

Knowledge operations, as introduced in section 5.1, support the design and alignment tasks of a metamodel engineer. These operations are applicable to perform homomorphic transformations of one or more metamodels. This means that the input is assessed, queried and and results are provided as a sub- or superset of the input elements.

The prototypical realisation of these operations builds on the shared modules "Conceptual Graph Query Interface" and "Conceptual Graph Reasoning Interface" that consume transparently the conceptual graph representation of one or more metamodels and operation definition (also as a conceptual graph) and return the results as projections of the original conceptual graph. As a stateless, meta-service, these functionalities can be defined by the metamodel engineer or platform providers freely and specialised to the specific needs. Operations are sequenced and provide the insights required during the design process.

For evaluation purposes, the Scene2Model metamodel is selected and knowledge operations are applied on the metamodel. The results of this evaluation are discussed in the following paragraphs. As a graphical aid, the conceptual graph is visualised either in CoChaCo4ADOxx or CoGUI. Knowledge operations run on these structures. Figure 7.5 shows the result of the import into the CoGUI toolkit, that supports layout algorithms and browsing of the structure as well as a graphical feedback on operations performed.

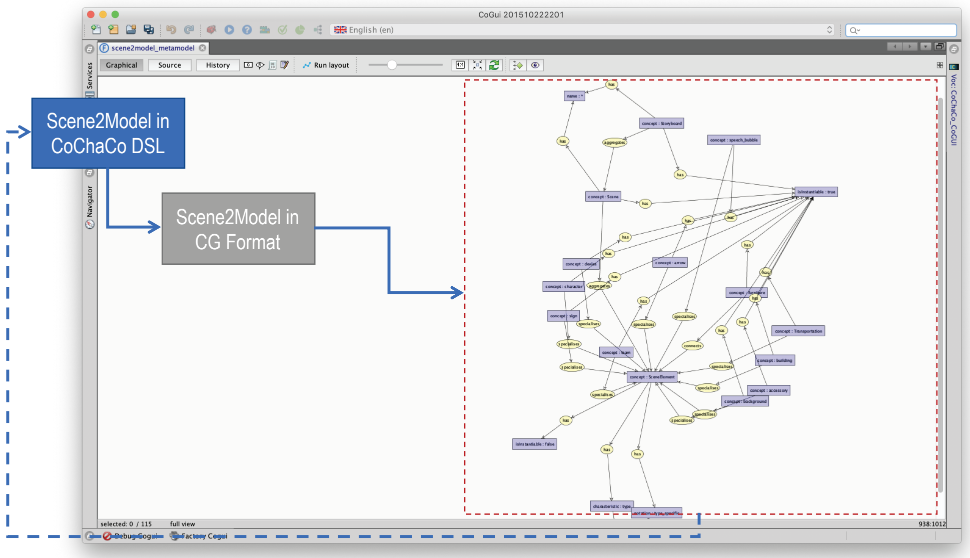


Figure 7.5: Screenshot: Conceptual Graph Visualisation (CoGUI, CGDF)

The CGIF representation of the Scen2Model metamodel above is shown below.

$$
\begin{aligned}
&[scene2model\_metamodel] \qquad (7.1)\\
&aggregates(< Storyboard >, < Scene >),\\
&aggregates(< Scene >, < SceneElement >),\\
&has(< SceneElement >, type_specific),\\
&has(< Scene >, true),\\
&has(< Storyboard >, true),\\
&has(< Storyboard >, X1),\\
&has(< Scene >, X1),\\
&has(< SceneElement >, type),\\
&has(< SceneElement >, false),\\
&specialises(team, < SceneElement >),\\
&has(team, true),\\
&specialises(character, < SceneElement >),\\
&has(character, true),\\
&specialises(device, < SceneElement >),\\
&has(device, true),\\
&specialises(sign, < SceneElement >),\\
&specialises(speech_bubble, < SceneElement >),\\
&specialises(arrow, < SceneElement >),\\
&has(sign, true),\\
&has(speech\_bubble, true),\\
&has(arrow, true),\\
&connects(furniture, < SceneElement >),\\
&has(furniture, true),\\
&has(< Transportation >, true),\\
&specialises(< Transportation >, < SceneElement >),\\
&specialises(building, < SceneElement >),\\
&has(building, true),
\end{aligned}
$$

$$\begin{aligned}
&has(background, true),\\
&specialises(background, < SceneElement >),\\
&specialises(accessory, < SceneElement >),\\
&has(accessory, true),\\
&connects(type, type_specific),\\
&concept(< Scene >),\\
&concept(background),\\
&name(X1), isInstantiable(false),\\
&concept(accessory),\\
&concept(speech_bubble),\\
&concept(device),\\
&notation(type_specific),\\
&concept(< SceneElement >),\\
&characteristic(type),\\
&isInstantiable(true),\\
&concept(sign),\\
&concept(< Transportation >),\\
&concept(character),\\
&concept(furniture),\\
&concept(< Storyboard >),\\
&concept(building),\\
&concept(arrow),\\
&concept(team).
\end{aligned}$$

Syntax Operations / Structural Analysis: structural analysis can be performed on the conceptual graph, whereas "structural" is understood as any operation that assesses the concept/relation hierarchy and its application within the metamodel as a conceptual graph. Queries enable this analytical assessment for e.g. counting the concepts and their dependencies, identification of structures not used. Syntax operations transform the conceptual graphs syntactically structure. This includes operations for normalisation (reducing unused relations or concepts in the graph), slicing/joining of metamodels or subsetting based on specific criteria.

Query Example 1: "Get all concepts defined as containers" (implies meaning/purpose of the concept is defined by included elements), represented using the "aggregates" relation in the CoChaCo grammar.

$$\begin{aligned}[query\_containers]?:- \quad & \\ & aggregates(X1, X2), \\ & concept(X1), \\ & concept(X2).\end{aligned} \tag{7.2}$$

Results for Scene2Model: 11 projections found within the Scene2Model metamodel (2 shown graphically in Fig. 7.6.

Result for all OMiLAB metamodels: 327 projections identified.

Query Example 2: "Get all domain-specific concrete concepts (implies any concept that is directly usable by the modeller/actor, no abstract or system concepts)

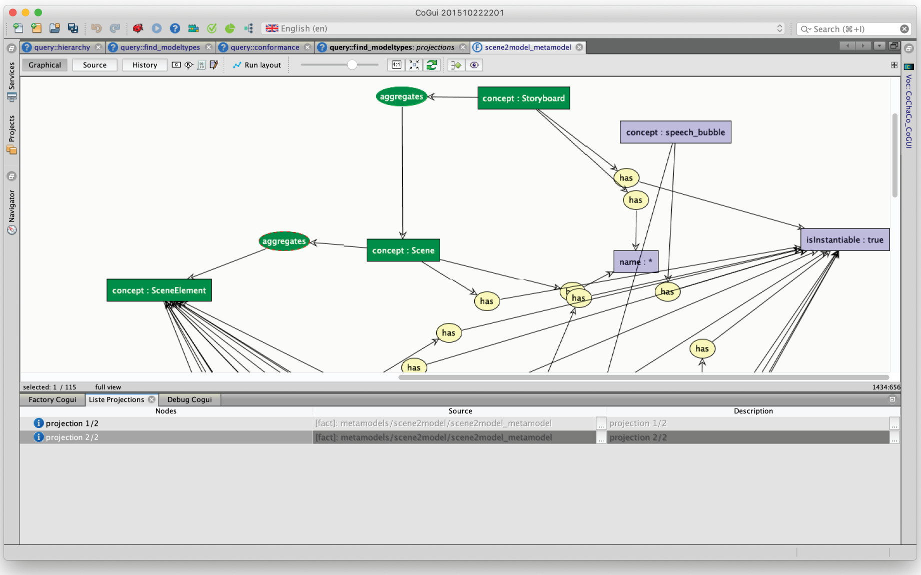
$$\begin{aligned}[query\_concept\_instantiable]?:- \quad & \\ & has(X1, true), \\ & isInstantiable(true).\end{aligned} \tag{7.3}$$

Results for Scene2Model: 134 projections identified

Results for all OMiLAB metamodels: 2270 projections identified.

As the projections identified are homomorphic conceptual graphs, slicing is feasible to e.g. retrieve a specific container type and all contained elements and re-use for design challenge (see section 3.4.2.

Semantic Operation: semantic operations enable the assessment of the meaning included in the metamodel. This includes operations to deduce additional information on the structure (e.g. type identification based on rules) and similarity assessments as input for virtual relations definition in modelling ecosystems.



Nodes	Source	Description
projection 1/2	[fact]: metamodels/scene2model/scene2model_metamodel	projection 1/2
projection 2/2	[fact]: metamodels/scene2model/scene2model_metamodel	projection 2/2

Figure 7.6: Screenshot: Conceptual Graph Projections

Reasoning Example 1: "Any aggregates relation has an inverse counterpart named composed_of" This example of a semantic operation handles the relation definition. Inverse relations are explicitly foreseen in the grammar and are deduced through the reasoning rule below.

$$
\begin{aligned}
&[derived\_composed\_of] \\
&\qquad composed\_of(X2, X1) \\
&\qquad :- \\
&\qquad aggregates(X1, X2), \\
&\qquad concept(X2) \\
&\qquad concept(X1).
\end{aligned}
\tag{7.4}
$$

Reasoning Example 2: "Any aggregating concept is a container that defines a viewpoint/diagram type, all other concepts are modelling constructs" Type identification enable advanced queries. As all operations can be sequenced, a deducing rule is the input for a one of the above syntactical queries.

$$\begin{aligned}
&[derived\_modeltypes] && (7.5)\\
&\quad specialises(X1, modeltype),\\
&\quad specialises(X2, modelling_concept),\\
&\quad concept(modeltype),\\
&\quad concept(modelling\_concept),\\
&\quad :-\\
&\quad composed\_of(X1, X2)\\
&\quad concept(X2)\\
&\quad concept(X1).
\end{aligned}$$

The rule definition above defines the type identification logic. Additional knowledge on the concepts defines is deduced based on the rules, as any "aggregates" relation, and its inverse counterpart "composed_of" are assessed and type specialisation links are established for the related concepts (modelling construct or modeltype).

Evaluation of CG Query and Reasoning Interface: The interface is capable to provide the functionality required for the design process and assesses the conceptual graph structurally/syntactically and semantically. As a stateless service, it allows for a flexible adaptation and configuration to the specific needs of the metamodel engineer

This flexibility though impacts usability considerations as queries and operations need to be established in advance and require knowledge on the techniques and approach to define such queries and reasoning rules using conceptual graphs. Out-of-the-box implementations reflect on the vocabulary in a narrow scope and need adaptation in case of an update in the vocabulary.

7.1.5 Publishing

The publishing phase is concerned with externalising (in a human readable as well as machine processable format) the metamodel code base. The transformation service operates on the grammar established and adapted during the design operations and generates two artefacts:

– RDF Export: the metamodel as a conceptual graph is exposed as a RDF representation. The direct mapping between RDF and CGIF is available and supports this transformation,

– Markdown Generator: for human interpretation, the abstract vocabulary as well as fact base is transformed to markdown dynamically and available as a documentation result from the IDE.

Evaluation of Retrieval and Adaptation: The publication mechanisms is applicable and produces complete results. Nevertheless, an observation relates to the extensive information artefact produces and its visual representation. Navigation, browsing and interactive filtering mechanisms are required to enable proper re-use capabilities on the code repository and its aretfacts.

7.2 Environment Evaluation: IBPM

Environment evaluation is concerned with the applicability of the artefact conceptualised within concrete metamodel design challenges from an industrial and research background. During the course of this dissertation research project, the design approach for metamodels has been applied within the European research project GO0DMAN, concerned with the development of a conceptual modelling approach for industrial business process management. The project context and challenges have partly motivated the problem identification, result development and evaluation phases of the results presented in this thesis.

Intermediate results achieved have been published and are available in the following selected publications:

– Dominik Bork, Hans-Georg Fill, Dinitris Karagiannis and Wilfrid Utz (2018). ‘Simulation of Multi-Stage Industrial Business Process Using Mmetamodelling Building Blocks with ADOxx’. In: *Journal of Enterprise Modelling and Information Systems Architectures* 13.2, pp. 333–344
The paper targets the idea on the novel metamodelling approach, using metamodel building blocks as a means for re-use of implementation results and functionality alignment, based on the project results presented in (Utz, Woitsch, Falcioni et al., 2018).

- Wilfrid Utz and Damiano Falcioni (Sept. 2018). 'Data Assets for Decision Support in Multi -Stage Production Systems Industrial Business Process Management using ADOxx'. In: *Proceedings - IEEE 16th International Conference on Industrial Informatics, INDIN 2018*. Institute of Electrical and Electronics Engineers Inc., pp. 809–814
  Technical assessment of the realisation concept for the use case developed (data streams and propagation along metamodel relations) resulting in the formulation of the modelling ecosystem approach
- Wilfrid Utz (2019a). 'Design of a Domain-Specific Metamodel for Industrial Business Process Management'. In: *Proceedings - 2019 8th International Congress on Advanced Applied Informatics (IIAI-AAI)*. vol. 2019. Toyama: IIAI, pp. 821–826
  Publication on evaluation results on the design approach followed and its implementation results.

The evaluation presented in the following sections has been discussed and introduced within the conference above to gain additional input and reflections on the intended contribution.

7.2.1 Requirements: Industrial Business Process Management

The requirements for the Industrial Business Process Management case presented are derived from a collaborative H2020 European research project with an execution time of 36 month (October 1, 2016 - September 30, 2019) in the field of zero-defect manufacturing. Overall the project aimed at establishing an integrated system on process and quality control in multi-stage manufacturing/production.

For the research performed in this thesis, the project acts as an evaluation environment to test ideas and evaluate the applicability of the results in a concrete environmental setting.

Fig. 7.7 shows the overall architecture of the system realised within the project: operational technologies on the shop-floor of production companies are abstracted (using an implementation of a multi-agent system) and further refined towards a knowledge-based environment. This knowledge-based environment is coined Industrial Business Process Management (IBPM) as it builds on the production processes as a baseline and elevates the definition

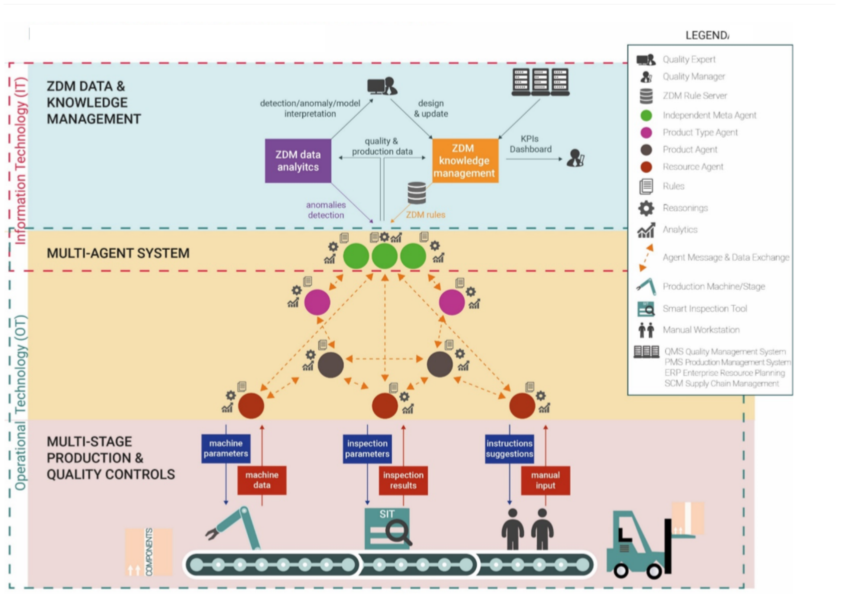


Figure 7.7: GO0DMAN Distributed System Architecture (Utz, Woitsch, Falcioni et al., 2018)

of the processes towards a decision-support system. The artefacts on this level are graphical, smart model that require a set of metamodels combined flexibly based on the characteristics of the production company.

From a user perspective, the following requirements are imposed on the modelling ecosystem:

- Support a standardised process representation: build on existing international standards and elevate with concepts from the production industry. Standardisation is required to provide means for sharing and comparison,
- Assess data streams: streams of data from the multi-agent-system are integrated in the model environment and support processing functionality such as simulation, process and product dashboards and input for smart decision making,
- Propagation rules: define the calculation logic between different abstraction level. These rules are considered the virtual relations between different representation in the knowledge management system,

Further details on the requirements and needs of the end-users are available in the public deliverable of the project ((Utz, Woitsch, Falcioni et al., 2018)) on the knowledge management system architecture.

7.2.2 Metamodelling using Building Blocks for IBPM

Applying the approach of metamodelling using building blocks (formalised as conceptual graphs), the following case/domain-specific procedure has been applied:

1. Transformation of input specification in proposed conceptual structure formalism: using a specific import/export adaptor capable to transform a the standard representation of the BPMN 2.0 metamodel into conceptual structures as a syntactic operation.

   The import service is based on the MOF syntax and its object model, transformation and mapping rules defined create the conceptual graph via the CoChaCo grammar defined. Manual layouting has been performed to increase readability of the graph.

2. Slicing of the metamodel: only the graph-based process structure for propagation is required. The overall BPMN 2.0 metamodel is partitioned and sliced using a query pattern for graph-based structures as a syntactic operation.

   Using queries to identify partitions with in the imported metamodel (type identification and aggreates/composed_of relation), partitions are identified. These partitions are marked first (colour-coded) and then filtered via projections resulting from type-based queries (domain-specific).

3. Discover/Bind Graph Propagation Rules: the conceptual structure of the graph propagation mechanisms is matched against the conceptual structure and refined to map quantitative facets and structure to the required input as a semantic operation.

   Binding of functionality on the metamodel is facilitated by matching functionalities. The required metamodel (as a federated functionality) is exposed as a conceptual graph that is mapped on the result from step 2. The syntactical mapping results in the elevation of the result towards the found concepts, implemented as "specialised" relations between the merged metamodels. In case such a direct mapping is not possible, the virtual relation concept supports a cascading definition of a semantic chain.

4. Elevate with domain-semantics: production/manufacturing specific needs and expressiveness is established as a domain elevation resulting from a semantic operation on concept types.

   This step is concerned with the inclusion of domain semantics. In the case of IBPM, the concepts have been specialised according to the needs of the end-user. Specifically the aspects of multi-stage production have been aligned with the "Task" definition in BPMN and additional flow semantics are added. The effect of this elevation (as functionality is already available on general concepts) is transparent i.e. are reflected in the algorithm accordingly.

5. Generate the skeleton for implementation: Alignment and verification with implementation platform capabilities and generation of skeleton.

   This technical step is concerned with the alignment of the design artefact with the realisation platform used for implementation. Verification queries support the alignment initially and a skeleton generator enables from-scratch implementation or extension techniques based on injections.

Fig. 7.8 shows step 1 to 4 of the procedure graphically. The transformation performed are queries and rules applied on the conceptual structure. Step 5 is discussed in detail within the instantiation section.

7.2.3 Instantiation of Building Blocks

The instantiation of the building blocks resulting from the design chain above defines the implementation process. The three phases in the metamodelling approach are defined and executed for the evaluation

1. Approach: defines the abstract metamodel based on the application case. The challenge is to collect and assess the concepts required based on domain-specific requirements, literature and pre-existing knowledge within an organisation.

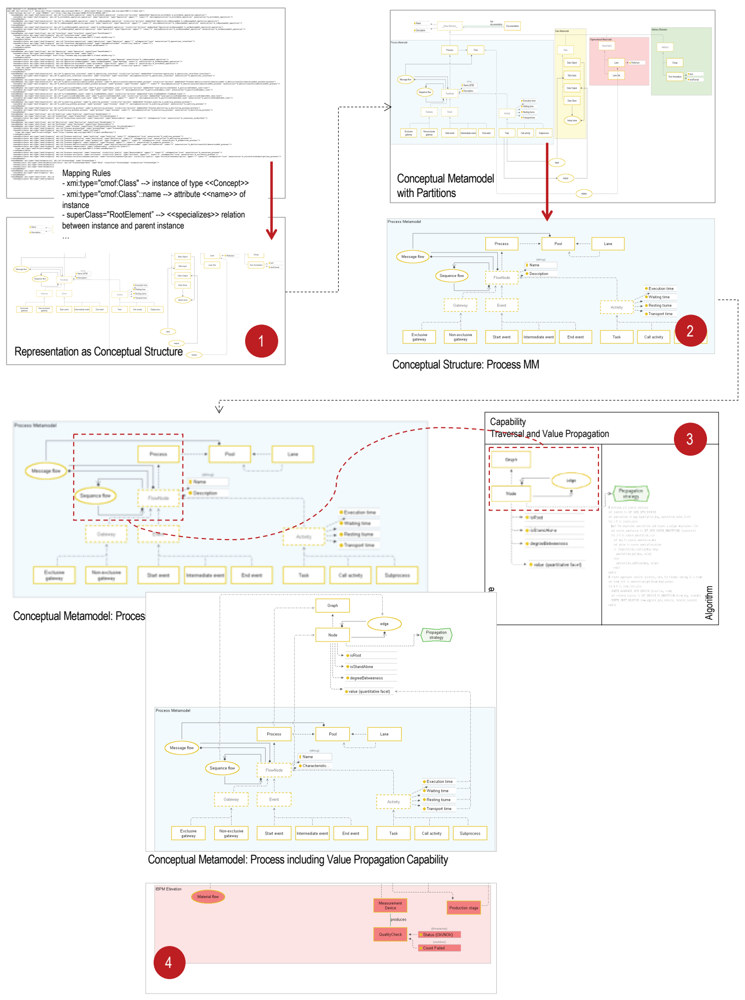


Figure 7.8: Knowledge Operations on Building Blocks for IBPM

For IBPM, hierarchical value propagation networks and strategies define the approach selected. These techniques are applicable for traversing value streams between different levels of abstraction. An example for the approach are low-level sensor streams that are aggregated according to the abstraction/decomposition logic, and result in condensed high-level views with capabilities to drill-down and apply data streams also for purposes such as simulation and analysis.

2. Concept: as a formalisation step, the abstract building block is refined and concretised. This implies that the formal representation is established that considers variants for aligning model processing capabilities. Deficiencies are reflected upon and implementation variants are created. The transformation from approach to concept is considered an instantiation of the approach in a concrete formal framework.

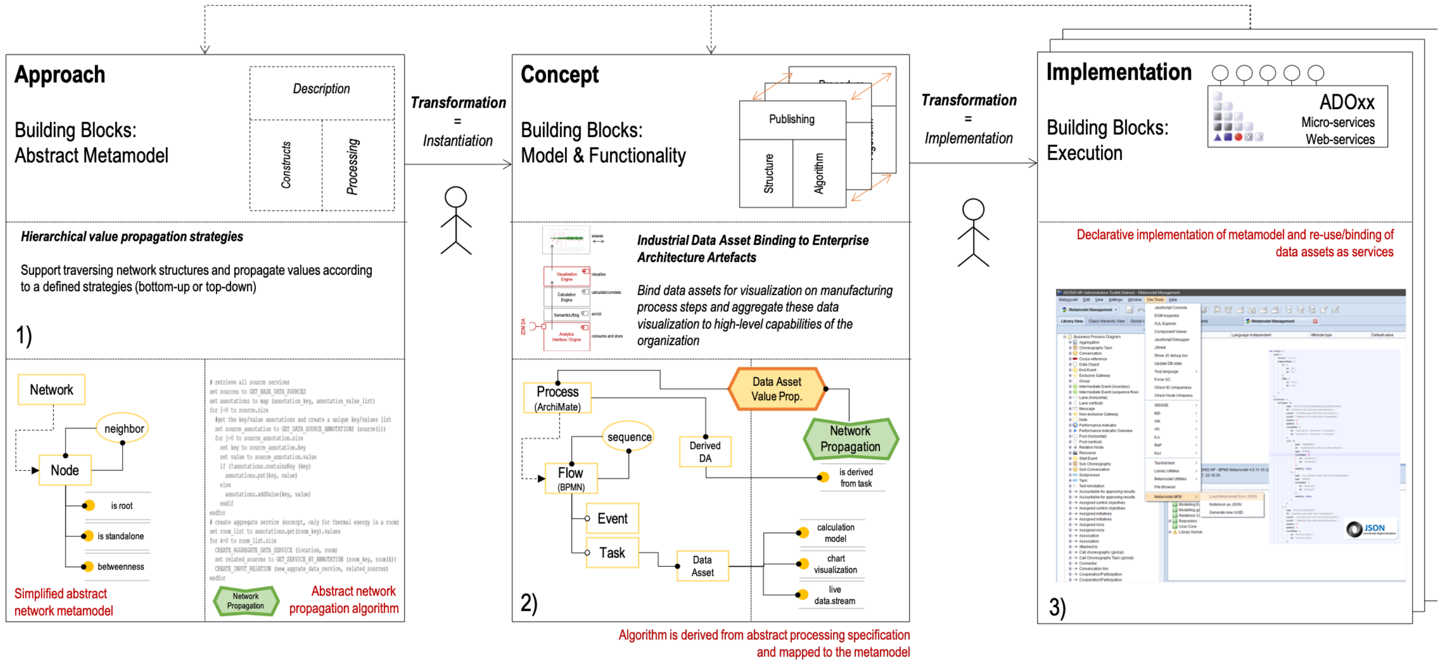
   As a concept, derived from the approach, data binding techniques are designed. This means that data producing entities are provided as concepts in the metamodel and alignment to enterprise assets (such as processes, IT artefacts, systems, etc.) is possible conceptually. The semantics of this relation is precisely defined via calculation models.

3. Implementation: concerned with the actual implementation as tooling and prototypes. Implementation considers also deployment and distribution aspects of the tool realised. The transformation from concept to implementation is considered an implementation of a concept in a runtime environment.

   The concept results are platform-independent, which means that operationalisation as e.g. a tool implementation requires a transformation. This transformation is supported using verification techniques, mapping the capabilities of a metamodelling platform explicitly towards the concept design.

7.2.4 Assessment and Evaluation

This evaluation stream targets the question how (procedural) metamodels can be (co-) designed to provide an efficient implementation approach to metamodel engineers. The concept of metamodelling using building blocks (individually formalised as conceptual structures and beyond as a combination approach) was evaluated. In an initial this evaluation targeted the


Approach
Building Blocks: Abstract Metamodel
Description
Constructs
Processing
Transformation = Instantiation
Concept
Building Blocks: Model & Functionality
Publishing
Structure
Algorithm
Transformation = Implementation
Implementation
Building Blocks: Execution
ADOxx
Micro-services
Web-services
1)
Network
neighbor
Node
is root
is standalone
betweenness
Simplified abstract network metamodel
Network Propagation
Abstract network propagation algorithm
2)
Process (ArchiMate)
Flow (BPMN)
sequence
Event
Task
Derived DA
Data Asset
Data Asset Value Prop.
Network Propagation
is derived from task
calculation model
chart visualization
live data.stream
Algorithm is derived from abstract processing specification and mapped to the metamodel
3)
Declarative implementation of metamodel and re-use/binding of data assets as services
JSON

Hierarchical value propagation strategies

Support traversing network structures and propagate values according to a defined strategies (bottom-up or top-down)

```
# retrieve all source services
set sources to GET_BASE_DATA_SOURCES
set annotations to map (annotation_key, annotation_value_list)
for i=0 to source.size
  #get the key/value annotations and create a unique key/values list
  set source_annotation to GET_DATA_SOURCE_ANNOTATIONS (source(i))
  for j=0 to source_annotation.size
    set key to source_annotation.key
    set value to source_annotation.value
    if (!annotations.containsKey (key)
      annotations.put(key, value)
    else
      annotations.addValue(key, value)
    endif
endfor
# create aggregate service (excerpt, only for thermal energy in a room)
set room_list to annotations.get(room_key).values
for k=0 to room_list.size
  CREATE_AGGREGATE_DATA_SERVICE (location, room)
  set related_sources to GET_SERVICE_BY_ANNOTATION (room_key, room(k))
  CREATE_INPUT_RELATION (new_aggrate_data_service, related_sources)
endfor
```

Industrial Data Asset Binding to Enterprise Architecture Artefacts

Bind data assets for visualization on manufacturing process steps and aggregate these data visualization to high-level capabilities of the organization

Figure 7.9: GO0DMAN IBM Metamodel Building Blocks (Utz, 2019a)

conceptual framework developed and proposed, applied on the application case IBPM thereafter.

This evaluation targeted the metamodelling approach that differs to traditional requirement - specification - implementation - testing and deployment cycles as agile considerations and re-use patterns are positioned at its core. The formal knowledge representation acts in this case as an enable to a) identify adequate structures and discover/bind functionality dynamically. Due to the temporal and organisational/role separation of concerns (approach, concept and implementation), various design variants become testable and feedback by involved stakeholder can be reflected within another design iterations.

An observed drawback of the proposed concept relates to the knowledge and skills required to handle formal knowledge representation techniques. Defining and using a conceptual graph of a metamodel is transparently useable, nevertheless the application in intelligent queries and deductive reasoning involves domain expertise on one hand and a core understanding of formal concepts and their representation. This becomes especially apparent when assessing the publishing features in the retrieval and adaptation building block: the essence of this building block is to maintain and document the outcomes of the development process during each phase and provide these results to the community. Even though graph-based representations are continuously evolving, the abstraction capabilities of a domain-expert need to be met and domain-semantics are needed to enrich these representations.

Summary. During the evaluation phase, two iterations of refinement have been performed: in a first step, the feasibility of the concept and technical realisation has been assessed with selected metamodel implementation. The resulting adaptation in the concept and architecture are reflected in chapter 5 and 6. In a second iteration the repository of metamodel has been used to assess the capabilities of the knowledge representation formalism, the CoChaCo grammar and visualisation options. Highlights and low-lights of this evaluation are articulated for each building block assessed, resulting in an evaluation of the research question and objectives as a conclusion of this thesis in chapter 8.

Chapter 8

Conclusions

This concluding chapter focuses on summarising the results achieved and highlighting/contrasting the contribution established against the research question and objective initially defined. The structure of the chapter follows the work phases, providing a summary on the achievements of each phase of the process that was also applied as the red thread of the thesis. Highlights and accomplishments are identified and discussed. The research question and derived objectives are re-introduced to provide the context of the discussion of results. A conclusive SWOT analysis has been performed to assess the achievements from a critical, self-reflective viewpoint. The threads and opportunities are input for the outlook and further research directions.

8.1 Summary

The motivation aspects for the research work presented in this thesis have been introduced in chapter 1: digital intelligence mechanisms and functionality within ecosystems of information systems are the core aspect this research project aimed to contribute to. This specific interest is derived from observations in today's enterprise settings. Organisations need to be innovative at a fast pace, reflect on intelligent offerings, interactions and processes at a fast pace and have means in their repertoire of tools and methods that enable them to react and evaluate new trends quickly and efficiently. Innovation laboratories provide the physical and virtual space, but the toolset to handle the involved complexity, involve stakeholders from various backgrounds and establish a systematic approach are limited. This deficiency results in ad-hoc decisions that are uninformed, ignore expertise of actors or information in systems and only partially cover innovative solution development processes. The question to work on (defining the objective of this research) relates to whether an adequate formalism can be identified, utilising the baseline on conceptual structure and knowledge representation to support a) design processes of metamodels (based on the assumption that the knowledge in a domain/field is encapsulated within the type level structure) and b) provide means for intelligence functionality and operations already during design.

The baseline for related work is derived from this direction and discussed in chapter 2. The literature on *intelligence* and *ecosystems* define the foundation for the research on related work related to smart models of information systems. During this foundational work, derived definitions have been establish that target intelligence via federation and modelling ecosystems as a distributed system of services that define their interactions and relations in a smart manner via virtual, semantically rich relations. Co-design aspects for different stakeholders acting on varying abstraction levels are considered as an intelligence feature for these ecosystems.

Chapter 3 develops the initial artefact based on these observations: concrete requirements are derived from motivational case studies (bottom-up, top-down approach in distributed modelling systems). A graphical design environment is proposed that builds on the core aspects of concepts - characteristics - connected, coined "CoChaCo". The core aspects are extended with model-processing/model value aspects to align functionalities, their purpose and relevant stakeholders within the design approach. CoChaCo is considered the structural element for the design technique. In parallel the baseline on conceptual structures is assessed, diving back into core aspects of graph-based knowledge representation. The approach developed by Sowa and refined by different research teams has been selected to formalise CoChaCo. This formalisation is required to perform knowledge operations (i.e. intelligent design support functionality) during the design pro cess. The formalisation has been done step-wise: initially the language constructs of CoChaCo have been identified and described as a grammar (ENBF form). This step resulted in the observation, that design freedom limits query and deductive capabilities and resulted in a mapping based approach towards an abstract vocabulary for metamodelling based on CoChaCo and design principles to map results against. This mapping is understood as a harmonisation effort, clarified in chapter 4.

The contribution of chapter 4 is characterised by the notion of conceptual structures: the high-level domain-specific design language for metamodels CoChaCo is proposed as a grammar definition, in the form of a DSL, as well as graphical notation, supported by a modelling tool, to elicit the specific structural and semantic aspects of metamodel. Mapping bidirectionally the language and constructs with a formal knowledge representation based on conceptual graphs, design support functionalities become possible. These functionalities are defined as knowledge operations operating on the conceptual structure of one or more metamodels. A main aspect in the definition of the conceptual graph structure is related to a knowledge representation that provides the means to anchor CoChaCo in a manner that operations required in the design process become feasible. These generic operations are sequenced and define the implementation process by gradually refining the output of the previous phase as input for the next one. Knowledge operations derived from literature and through operations of the work in the community are defined. During the development of the design approach, its formalisation, and realising architecture, the aspects of openness, extendability have been considered. This is specifically relevant for domain-specific extension capabilities, uncertain abstraction techniques and implementation/design strategies. The core outcome of this chapter is the specification of a bidirectional mapping logic between a language-oriented design approach, defined as a grammar and modelling language and its representation as a conceptual graph, as input for the definition of the knowledge operations for harmonisation and functionality alignment.

These operations are targeted in chapter 5 that defines the environment needed and its structure as a procedure model for the design of modelling ecosystems. Harmonisation and federation concepts are introduced for novel, or existing metamodels in light of intelligence operations required within the system. The procedure model builds on the three phases of Approach - Concept - Implementation to systematically structure the design to implementation process. Re-use and agile adaptation concerns are reflected and the generic anatomy for these operations is specified as an input for the technical realisation concept. Specific examples are developed to enable a classification of operations to a) query, b) integrate (via equivalence or similarity relations or virtual/ecosystem relation), c) perform dependency analysis on a conceptual/type level.

The technical realisation (to assess the feasibility of the conceptual design) is discussed in chapter 6. The overall architecture of the design environment is defined and contribution building blocks are introduced. These building blocks are considered as service containers, where actual functionality is provided with a high-degree of flexibility through smart binding techniques. As such each building blocks defines (in abstract terms) its consuming and required interfaces (structure, format). Any service that fulfils the transformation is applicable for inclusion, in case not existing the interfaces define the service implementation skeleton for any interested stakeholder/community to include specific functionalities.

Both the conceptual design and technical realisation architecture are input to the evaluation approach discussed in chapter 7. Prototypes realised are tested twofold: for completeness with respect to pre-existing metamodels utilising a code base of operational metamodels and environmental evaluation presenting a concrete, domain-specific implementation case within a research project. Evaluation results have fed back to the design phases and informed/updated the concept realisation.

8.2 SWOT Analysis

Summarising the evaluation of the design approach for metamodels using conceptual structures, a conclusive SWOT analysis has been performed to provide an assessment of the results achieved based on the criteria of completeness, adequacy and efficiency of the novel design approach proposed. For this analysis the design approach/procedure, the elements defined for CoChaCo, the mapping and transformation concept towards conceptual structures and its technical realisation have been assessed holistically.

Strength. The strength of the novel design approach for metamodels are listed below.

- *Openness and Extendability:* The process/procedure using metamodel building blocks assumes re-use as a design pattern and supports knowledge sharing and co-creation. From a technical point of view, the architecture is designed in a way, that extensions can be added dynamically and elevate the functional capabilities of the development environment. Algorithmic solutions and mechanisms that already exist, can support specific cases without changing the structure.
- *Cognition and Understanding:* the approach developed in this research project centres around intelligence as a cognitive function. This cognition is considered unique as it applies a type-level assessment (discussing concepts, characteristics and connectors instead of concrete implementation cases as instances). The complexity in large systems is manageable as agreements between stakeholders are established in an

ecosystem manner, clearly and explicitly defining the relations between individual nodes in the network.

– *Established Technologies:* The technologies used are mature and well established. As such the risk of uptake is reduced and is considered a strength in its application.

– *Formal Knowledge Representation:* the design approach builds on a combination of freedom (during the ideation and co-creation) phase. This means that the metamodel engineer is not restricted by constraints of a formalism and is free to develop alternative (even syntactically/semantically "wrong") representations. CoChaCo provides this liberating aspect and does not reinforce any restriction. The formalisation is transparent, which means that design results are continuously transformed and add value to the process.

Weakness. The weakness of the approach are mainly attributed to the knowledge and skill set required to maintain a distributed modelling ecosystem:

– *Quality Considerations:* The approach presented prominently discusses adequacy instead of quality. This differentiation is important as the value of a model/metamodel is coupled with the use and applicability. Nevertheless, the assessment whether adequacy is reached using the DeMoMa::* environment is not evaluated formally, but it is assumed that a system that is structured in multiple, distributed entities, considers agile adaptation and considers ecosystem links will evolve in such a direction.

– *Metamodel Engineer knowledge and skill set:* Formalisation requires the skills and knowledge to operate and use it. The different roles involved in the design process are still maintained (domain expert, metamodel engineer); in complex cases extended by a knowledge engineer. The additional effort in establishing such a formal level needs to be assess in context of the value gained, even through the formalisation is transparent and systematically supported.

– *Technical Readiness:* Components and artefacts developed are on a prototype level; this includes re-used libraries (e.g. for conceptual graphs). Mitigation strategies have been though after, especially in the architectural blue print to provide means to replace components with upcoming and novel techniques that might be more appropriate and fitting.

Opportunity. The opportunities established by the results presented and contribution proposed are manifold as the formalisation could result in an extended uptake within the conceptual modelling community and beyond. As a baseline for in-depth analytical support, during the design and operation of information systems artefacts, metamodels are processable design artefact. This includes integration and compatibility aspects.

– *Integration:* as state-of-the-art software development technologies have been selected, integration within other domains becomes feasible, e.g. as embedded smart models within systems, to support the semantic needs of software components, for transformations between operational systems, etc.)

– *Compatibility:* in relation with the strength on openness, the selection of conceptual structures is regarded an opportunity as compatibility is implicitly granted through available mapping to other knowledge representation standards and logic representations.

Thread. Threads identified are closely related to the weakness defined above. The core concern is attributed to semantic expressiveness and its impact on knowledge operations. As the concept builds on a rather flat structure (CoChaCo definition, abstract vocabulary) it is difficult to estimate whether this expressiveness is sufficient to cover increasing and evolving complexity in information systems. Following the development in other domains that moved from design to recognition/mining, which is based on the assumption that the truth and value is established by past events and their representation in data, the conceptual viewpoint (on type level) seems to be contradictory. Marrying these two worlds could potentially lead to an increased reception and validation of designed metamodels.

8.3 Discussion and Outlook

The objectives defined for this research project have been derived as part of the motivation, derived from the research question (repeated below) are applied for the discussion and identification of future research directions based on the outcomes of this research project. This subjective assessment is established based on the experience within the conceptual modelling community and the reception of modelling methods and tooling.

Which form of knowledge representation is appropriate and can be identified to support the analysis of information systems defined by underlying metamodels?

Objective 1: Design Environment. the DeMoMa::* design environment has been conceptually defined and prototypically developed following re-use and combination patterns classified as knowledge operation. As an environment it spans across the results on procedure, tasks and tooling to support the design process. As specific emphasis has been put on re-engineering techniques (as documented in the evaluation chapter) that has been identified as a future research challenge by (Višić, 2016; Visic, H. G. Fill et al., 2015) using the services for operational metamodels, it becomes a) possible to assess metamodels that are operational and detect the concepts defined, even for cases where no direct explication has been performed in the implementation and b) a novel combination technique based on these results as virtual/ecosystem relation is introduced. Both streams contribute to the intelligence consideration. A future research stream identified aims at models, or better metamodels at runtime. This implies that the concretisation of a metamodel does not happen in advance but during its application: types are identified and dynamically change the available concepts. The requirement to have multiple levels of abstraction (similar to the concept of intrinsic attributes introduced in (Frank, 2014)), is supported as the conceptual structures are on a type, rather than instance level.

Objective 2: Harmonisation. the harmonisation objective is omnipresent as a result of an ever changing environment surrounding us. Valid assumption are adapted and modify the conceptual frame organisations operate within. Therefore harmonisation has been defined as concept of conceptual connectivity. Interrelations are between participating nodes in an ecosystem have an assigned lifecycle (come into existence, grow and are decommissioned); the building nodes targeting particular domains and stakeholders experience the same evolution. Harmonisation is therefore considered as a temporal aspect - revision safety, traceability and design trajectories are equally important within the system as are the modelling artefact created with the specific nodes. This harmonisation objective impacts on one hand the definition of a modelling method as a holistic approach would span across multiple participating nodes (see motivational examples in 3.1.1 and 3.1.2). Currently the procedural aspect of the methods is only available implicitly; assessing these aspects and which types of procedures can be recognised is considered a second stream of research - correlating not only concepts within the same metamodels but across them respecting the relations established.

Objective 3: Collection and Community. based on the assumption that a collective, shared approach would support the uptake of the design approach, an initial collection of metamodels/metamodel fragments has been established as part of this research project. Retrieval and adaptation techniques are defined and enable metamodel engineer to discover concepts and structures that are appropriate and adequate for a domain requirement (structure/semantic or functionality). Co-design aspects and methods/tools to support creativity are needed as a frontend, to a) identify adequate metamodels and b) validate the exposed functionalities in a distributed manner. Collective intelligence on concepts and their relations impacts the way the design process is constructed. As such the repository and collection of metamodels requires an extension to also classify design process instances and externalise the knowledge involved by the experts. Changing the ecosystems is then driven by best practices that are evaluated continuously within the community.

Digital intelligence, defined within this research project as a cognitive, analytical capability, requires metamodels that are designed adequately. In a setting that is constantly changing, co-creation has become a design principle and tools and methods are needed to conceptually analyse such complex information systems (in a broad sense, including strategy and business modelling to process and organisational views, legal aspects and technological layers such as cyber-physical systems) to stay competitive, constantly transforming - in an informed way - organisations, enterprise and the society we live in.

There is nothing permanent except change. Heraclitus (circa 500 BC)

References

Adams, Nan B. (2004). 'Digital Intelligence Fostered by Technology'. In: *The Journal of Technology Studies.*

Adams, Nan B. (2010). 'Digital Intelligence'. In: *Teaching through Multi-User Virtual Environments.*

ADOxx.org (2020). *ADOxx Metamodelling Platform.* URL: https://www.adoxx.org/live/home (visited on 22/03/2020).

Almeida, João, Ulrich Frank and Thomas Kühne (2018). 'Multi-Level Modelling (Dagstuhl Seminar 17492)'. In: *Dagstuhl Reports* 7.12, p. 49. URL: http://www.dagstuhl.de/17492.

Alter, Steven. (1991). *Information Systems: A Management Perspective Addison-Wesley.* Addison Wesley, p. 523.

Andal-Ancion, Angela, Phillip A Cartwright and George S. Yip (2003). 'The digital transformation of traditional businesses'. In: *MIT Sloan Management Review* 44.4, pp. 34–41.

Atkinson, Colin and Thomas Kühne (2003). 'Model-driven development: A metamodeling foundation'. In: *IEEE Software* 20.5, pp. 36–41.

Avison, David and Guy Fitzgerald (2006). *Information systems development: methodologies, techniques & tools.*

Baget, Jean François, Madalina Croitoru, Alain Gutierrez, Michel Leclère and Marie Laure Mugnier (2010). 'Translations between RDF(S) and conceptual graphs'. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics).* Vol. 6208 LNAI, pp. 28–41.

Becker, Jörg, Christian Janiesch and Daniel Pfeiffer (2007). 'Reuse mechanisms in situational method engineering'. In: *IFIP International Federation for Information Processing.* Vol. 244, pp. 79–93.

Bee-Up (2019). URL: http://austria.omilab.org/psm/content/bee-up/info (visited on 22/02/2019).

Berman, Saul J. (Mar. 2012). 'Digital transformation: Opportunities to create new business models'. In: *Strategy and Leadership* 40.2, pp. 16–24.

Bihanic, David and Thomas Polacsek (2012). ‘Models for visualisation of complex information systems’. In: *Proceedings of the International Conference on Information Visualisation*, pp. 130–135.

Blanc, Xavier, Marie Pierre Gervais and Prawee Sriplakich (2005). ‘Model bus: Towards the interoperability of modelling tools’. In: *Lecture Notes in Computer Science*. Vol. 3599. Springer, Berlin, Heidelberg, pp. 17–32.

Bock, Alexander and Ulrich Frank (2016). ‘Multi-perspective enterprise modeling-Conceptual foundation and implementation with ADOxx’. In: *Domain-Specific Conceptual Modeling: Concepts, Methods and Tools*. Cham: Springer International Publishing, pp. 241–267.

Bohmann, Tilo, Jan Marco Leimeister and Kathrin Möslein (2014). ‘Service Systems Engineering - A Field for Future Information Systems Research’. In: *Business & Information Systems Engineering.*

Bork, Dominik, Robert Andrei Buchmann, Dimitri Karagiannis, Moonkun Lee and Elena-Teodora Miron (2019). ‘An Open Platform for Modeling Method Conceptualization: The OMiLAB Digital Ecosystem’. In: *Communications of the Association for Information Systems*, pp. 673–679.

Bork, Dominik, Hans-Georg Fill, Dinitris Karagiannis and Wilfrid Utz (2018). ‘Simulation of Multi-Stage Industrial Business Process Using Mmetamodelling Building Blocks with ADOxx’. In: *Journal of Enterprise Modelling and Information Systems Architectures* 13.2, pp. 333–344.

Bork, Dominik, Dimitris Karagiannis and Benedikt Pittl (2018). ‘How are metamodels specified in practice? Empirical insights and recommendations’. In: *Americas Conference on Information Systems 2018: Digital Disruption, AMCIS 2018.*

Bower, J. L. and C. M. Christensen (1995). ‘Disruptive technologies: catching the wave’. In: *Long Range Planning* 28.2, p. 155.

Brinkkemper, Sjaak (1996). ‘Method engineering: Engineering of information systems development methods and tools’. In: *Information and Software Technology.*

Brinkkemper, Sjaak, Motoshi Saeki and Frank Harmsen (1999). ‘Meta-modelling based assembly techniques for situational method engineering’. In: *Information Systems* 24.3, pp. 209–228.

Briscoe, Gerard and Philippe De Wilde (Dec. 2006). ‘Digital ecosystems: Evolving service-orientated architectures’. In: *2006 1st Bio-Inspired Models of Network, Information and Computing Systems, BIONETICS*. arXiv: 0712.4102.

Brodie, Michael L. (1989). ‘Future Intelligent Information Systems: AI and Database Technologies Working Together’. In: *Readings in Artificial Intelligence and Databases*. Elsevier, pp. 623–641.

Brown, Tim (2008). ‘Design thinking’. In: *Harvard Business Review*.

Burkhard, Benjamin, Neville Crossman, Stoyan Nedkov, Katalin Petz and Rob Alkemade (June 2013). *Mapping and modelling ecosystem services for science, policy and practice.*

Buse, Raymond P.L. and Thomas Zimmermann (2012). ‘Information needs for software development analytics’. In: *Proceedings - International Conference on Software Engineering*, pp. 987–996.

Chein, Michel and Marie Laure Mugnier (1992). ‘Conceptual graphs: fundamental notions’. In: *Revue d’intelligence artificielle*.

Chein, Michel, Marie Laure Mugnier and Madalina Croitoru (2013). ‘Visual reasoning with graph-based mechanisms: The good, the better and the best’. In: *Knowledge Engineering Review* 28.3, pp. 249–271.

Chein, Michel and Marie-Laure Mugnier (Oct. 2008). ‘Basic Conceptual Graphs’. In: *Graph-based Knowledge Representation*. Springer London, pp. 21–57.

Chen, David, Guy Doumeingts and François Vernadat (2008). ‘Architectures for enterprise integration and interoperability: Past, present and future’. In: *Computers in Industry* 59.7, pp. 647–659.

Cho, Hyun and Jeff Gray (2011). ‘Design patterns for metamodels’. In: *SPLASH’11 Workshops - Compilation Proceedings of the Co-Located Workshops: DSM’11, TMC’11, AGERE’11, AOOPES’11, NEAT’11, and VMIL’11*, pp. 25–31.

Christensen, Clayton M., Michael Raynor and Rory McDonald (2016). *What is disruptive innovation?* URL: https://hbr.org/2015/12/what-is-disruptive-innovation.

Ciriello, Raffaele Fabio, Alexander Richter and Gerhard Schwabe (Dec. 2018). 'Digital Innovation'. In: *Business and Information Systems Engineering* 60.6, pp. 563–569.

'Conceptual Structure' (2019). In: *Lexicon of Linguistics.* URL: `http://www2.let.uu.nl/uil-ots/lexicon/`.

Croitoru, Madalina, Bo Hu, Srinandan Dashmapatra, Paul Lewis, David Dupplaw and Liang Xiao (2007). 'A conceptual graph based approach to ontology similarity measure'. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics).* Vol. 4604 LNAI. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 154–164.

Crossman, Neville D. et al. (June 2013). 'A blueprint for mapping and modelling ecosystem services'. In: *Ecosystem Services* 4, pp. 4–14.

Cyre, Walling R. (1997). 'Automating System Design with Conceptual Models'. In: Springer, Boston, MA, pp. 73–90.

Czarnecki, Krzysztof and Simon Helsen (2003). *OOPSLA'03 Workshop on Generative Techniques in the Context of Model-Driven Architecture Classification of Model Transformation Approaches.* Tech. rep., p. 17.

Davies, Rob (2018). *Apple becomes world's first trillion-dollar company.* URL: `https://www.theguardian.com/technology/2018/aug/02/apple-becomes-worlds-first-trillion-dollar-company`.

Davis, Randall, Howard Shrobe and Peter Szolovits (1993). 'What is a knowledge representation?' In: *AI Magazine.*

Delmond, Marie-HHllne, Fabien Coelho, Alain Keravel and Robert Mahl (2017). 'How Information Systems Enable Digital Transformation: A Focus on Business Models and Value CooProduction'. In: *SSRN Electronic Journal.*

Diehl, Stephan (2007). *Software visualization: Visualizing the structure, behaviour, and evolution of software*, p. 187.

Dobing, Brian and Jeffrey Parsons (2011). 'Dimensions of UML Diagram Use'. In: *Journal of Database Management.*

DQ Institute (2020). *Digital Intelligence.* URL: `https://www.dqinstitute.org/dq-framework/%7B%5C#%7Ddigital%7B%5C_%7Dintelligence%20https://www.dqinstitute.org/` (visited on 14/04/2020).

Ebert, Jürgen and Angelika Franzke (1995). 'A declarative approach to graph based modeling'. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 903. Springer Verlag, pp. 38–50.

Ebert, Jürgen, Roger Süttenbach and Ingar Uhe (1997). 'Meta-CASE in practice: A CASE for KOGGE'. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 1250. Springer Verlag, pp. 203–216.

Efendioglu, Nesat, Robert Woitsch and Wilfrid Utz (2016). 'A toolbox supporting agile modelling method engineering: ADOxx.org modelling method conceptualization environment'. In: *Lecture Notes in Business Information Processing*.

Efendioglu, Nesat, Robert Woitsch, Wilfrid Utz and Damiano Falcioni (2017). 'A Product-Service System Proposal for Agile Modelling Method Engineering on Demand: ADOxx . org'. In: *Digital Enterprise Computing 2017*, pp. 199–212.

Elmansy, Rafig (2018). *Why Design Thinking Doesn't Work*. URL: `https://www.designorate.com/why-design-thinking-doesnt-work/` (visited on 22/11/2019).

Fellbaum, Christiane (1998). 'WordNet: An electronic lexical database. 1998'. In: *British Journal Of Hospital Medicine London England 2005*.

Fernandez, George, Liping Zhao and Inji Wijegunaratne (2003). 'Patterns for federated architecture'. In: *Journal of Object Technology* 2.3, pp. 135–149.

Frakes, William B. and Kyo Kang (July 2005). 'Software reuse research: Status and future'. In: *IEEE Transactions on Software Engineering* 31.7, pp. 529–536.

Frank, Ulrich (1999). 'Conceptual Modelling as the Core of the Information Systems Discipline - Perspectives and Epistemological Challenges'. In: *Proceedings AMCIS Americas Conference on Information Systems*, pp. 695–697. URL: `http://aisel.aisnet.org/amcis1999/240`.

Frank, Ulrich (2014). 'Multi-perspective enterprise modeling: Foundational concepts, prospects and future research challenges'. In: *Software and Systems Modeling*.

Gallagher, Brian (2006). 'Matching structure and semantics: A survey on graph-based pattern matching'. In: *AAAI Fall Symposium - Technical Report*. Vol. FS-06-02, pp. 45–53.

Gardner, Howard (1993). *Multiple Intelligences Theory to practice*, p. 304.

Gardner, Howard (1999). *Multiple Intelligence, Intelligence Reframed, for the 21st.* Basic Books, p. 292.

Gerbé, Olivier, Rudolf K. Keller and Guy W. Mineau (1998). 'Conceptual graphs for representing business processes in corporate memories'. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics).* Vol. 1453. Springer Verlag, pp. 401–415.

Gerbé, Olivier, Guy W. Mineau and Rudolf K. Keller (2001). 'Conceptual graphs and metamodeling'. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics).* Vol. 2120. Springer Verlag, pp. 245–259.

Gill, Asif Qumer (2015). 'Agile enterprise architecture modelling: Evaluating the applicability and integration of six modelling standards'. In: *Information and Software Technology.*

GitHub (2019). *State of the Octoverse.* Tech. rep. URL: https://octoverse.github.com/.

Goethals, Frank (2011). 'An Overview of Enterprise Architecture Framework Deliverables'. In: *SSRN Electronic Journal.*

Götzinger, David, Elena Teodora Miron and Franz Staffel (2016). 'OMiLAB: An open collaborative environment for modeling method engineering'. In: *Domain-Specific Conceptual Modeling: Concepts, Methods and Tools.* Cham: Springer International Publishing, pp. 55–76.

Gutierrez, Alain, Michel Lecl, Marie-laure Mugnier, Eric Salvat, Michel Chein, Madalina Croitoru and David Genest (2009). 'RDF to Conceptual Graphs Translations'. In: *CS-TIW'09: 3rd Conceptual Structures Tool Interoperability Workshop @ICCS'09: 17h International Conference on Conceptual Struc- tures.*

Guychard, Christophe, Sylvain Guerin, Ali Koudri, Antoine Beugnard and Fabien Dagnat (2013). 'Conceptual interoperability through Models Federation'. In:

Haemisch, York (2013). *From innovation to product: The challenge of disruptive technologies such as Digital Photon Counting (DPC).*

Heimbigner, Dennis and Dennis McLeod (July 1985). 'A Federated Architecture for Information Management'. In: *ACM Transactions on Information Systems (TOIS)* 3.3, pp. 253–278.

Henderson-Sellers, Brian and Jolita Ralyté (2010). *Situational method engineering: State-of-the-art review.*

Hevner, Alan R, Salvatore March, Jinsoo Park and Sudha Ram (2004). 'Design Science in Information Systems Research'. In: *Management Information Systems Quarterly* 28.1, pp. 75–105.

Hevner, Alan and Samir Chatterjee (2010). *Design Science Research in Information Systems.*

Höfferer, Peter (2007). 'Achieving business process model interoperability using metamodels and ontologies'. In: *Proceedings of the 15th European Conference on Information Systems, ECIS 2007*, pp. 1620–1631. URL: http://aisel.aisnet.org/ecis2007/174.

Hrgovcic, Vedran, Dimitris Karagiannis and Robert Woitsch (2013). 'Conceptual modeling of the organisational aspects for distributed applications: The semantic lifting approach'. In: *Proceedings - International Computer Software and Applications Conference*, pp. 145–150.

Hrgovcic, Vedran, Wilfrid Utz and Dimitris Karagiannis (2011). 'Service modeling: A model based approach for business and IT alignment'. In: *Proceedings - International Computer Software and Applications Conference.*

Huibers, Theo, Iadh Ounis and Jean Pierre Chevallet (1996). 'Conceptual graph aboutness'. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics).* Vol. 1115. Springer Verlag, pp. 130–144.

Information Systems (2016). *Oxford Reference - A Dictionary of Computer Science.* Ed. by Andrew Butterfield, Gerard Ekembe Ngondi and Anne Kerr. 7th ed. Oxford University Press. URL: https://www.oxfordreference.com/view/10.1093/acref/9780199688975.001.0001/acref-9780199688975-e-2571.

ISO, IEC and IEE (2011). *ISO/IEC/IEEE 29148: Systems and software engineering — Life cycle processes — Requirements engineering.* Tech. rep.

Jackendoff, R (1992). *Semantic structures.* Cambridge, Mass: MIT Press.

Jeusfeld, Manfred A. (2016). 'SemCheck: Checking Constraints for Multi-perspective Modeling Languages'. en. In: *Domain-Specific Conceptual Modeling.* Cham, pp. 31–53.

Jeusfeld, Manfred A. (2017). *Semcheck - Script for the Case Studies at NEMO Summerschool 2017.* Vienna, Austria.

Jeusfeld, Manfred A., Matthias Jarke and John Mylopoulos (2010). 'Metamodeling for method engineering'. In: *Choice Reviews Online* 47.07, pp. 47–3853–47–3853.

Karagiannis, Dimitris (2015). 'Agile modeling method engineering'. In: *ACM International Conference Proceeding Series.* Vol. 01-03-Octo, pp. 5–10.

Karagiannis, Dimitris, Dominik Bork and Wilfrid Utz (2019). 'Metamodels as a Conceptual Structure: Some Semantical and Syntactical Operations'. In: *The Art of Structuring.* Cham: Springer International Publishing, pp. 75–86.

Karagiannis, Dimitris, Robert Andrei Buchmann, Patrik Burzynski, Ulrich Reimer and Michael Walch (2016). 'Fundamental conceptual modeling languages in OMiLAB'. In: *Domain-Specific Conceptual Modeling: Concepts, Methods and Tools.* Cham: Springer International Publishing, pp. 3–30.

Karagiannis, Dimitris, Patrik Burzynski, Wilfrid Utz and Robert Andrei Buchmann (2019). 'A Metamodeling Approach to Support the Engineering of Modeling Method Requirements'. In: *Proceedings - 2019 27th IEEE International Requirements Engineering Conference*, pp. 199–210.

Karagiannis, Dimitris and Harald Kühn (2002). 'Metamodelling platforms'. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics).*

Karagiannis, Dimitris, Heinrich C. Mayr and John Mylopoulos (July 2016). *Domain-specific conceptual modeling: Concepts, methods and tools.* Springer International Publishing, pp. 1–594.

Karve, Saket, Vasisht Shende and Swaroop Hople (2019). 'Semantic relatedness measurement from Wikipedia and WordNet using modified normalized google distance'. In: *Lecture Notes in Networks and Systems*. Vol. 43. Springer, pp. 143–154.

Kim, Hyeyoung, Jae Nam Lee and Jaemin Han (2010). 'The role of IT in business ecosystems'. In: *Communications of the ACM*.

Koch, Stefan, Stefan Strecker and Ulrich Frank (2006). 'Conceptual modelling as a new entry in the bazaar: The open model approach'. In: *IFIP International Federation for Information Processing* 203, pp. 9–20.

Kocura, Pavel (2000). 'Semantics of attribute relations in conceptual graphs'. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 1867. Springer Verlag, pp. 235–248.

Krcmar, Helmut (2015). *Informationsmanagement*. Berlin, Heidelberg: Springer Berlin Heidelberg, p. 814.

Kruchten, Philippe (1995). 'Architectural Blueprints—The "4+1" View Model of Software Architecture'. In: *IEEE Software*.

Kühn, Harald (2004). *Methodenintegration im Business Engineering*. April. University of Vienna, PhD Thesis.

Litwin, Witold, Leo Mark and Nick Roussopoulos (Jan. 1990). 'Interoperability of Multiple Autonomous Databases'. In: *ACM Computing Surveys (CSUR)* 22.3, pp. 267–293.

Lopez-Fernandez, Jesus J., Esther Guerra and Juan De Lara (2014). 'Assessing the quality of meta-models'. In: *CEUR Workshop Proceedings*. Vol. 1235. CEUR-WS, pp. 3–12.

Ma, Zhiyi, Xiao He and Chao Liu (2013). 'Assessing the quality of metamodels'. In: *Front. Comput. Sci* 7.4, pp. 558–570.

Marker, David (2002). *Model theory an introduction*.

Masuda, Hisashi and Wilfrid Utz (2019). 'Visualization of Customer Satisfaction Linked to Behavior Using a Process-Based Web Questionnaire'. In: *Proceedings - 2019 12th International Conference on Knowledge Science, Engineering and Management*, pp. 596–603.

Masuda, Hisashi, Wilfrid Utz and Yoshinori Hara (2013). 'Context-Free and Context-Dependent Service Models Based on "Role Model" Concept for Utilizing Cultural Aspects'. In: *Proceedings - Knowledge Science, Engin-*

eering and Management. KSEM 2013. Ed. by Mingzheng Wang. Springer Berlin Heidelberg, pp. 591–601.

Matt, Christian, Thomas Hess and Alexander Benlian (Oct. 2015). *Digital Transformation Strategies.*

Mccormick, James, Gene Leganza and Chandler Henning (2019). ‘The Forrester Wave™: Digital Intelligence Platforms, Q4 2019’. In:

Meder, Michael, Brijnesh Johannes Jain, Till Plumbaum and Frank Hopfgartner (2015). *Chapter 9 Gamification ofWorkplace Activities*, pp. 239–268.

Mens, Tom and Pieter Van Gorp (Mar. 2006). ‘A taxonomy of model transformation’. In: *Electronic Notes in Theoretical Computer Science* 152.1-2, pp. 125–142.

Menzies, Tim and Thomas Zimmermann (2013). ‘Software analytics: So what?’ In: *IEEE Software* 30.4, pp. 31–37.

Miller, George A. (1995). ‘WordNet: A Lexical Database for English’. In: *Communications of the ACM.*

Mineau, Guy, Gerd Stumme and Rudolf Wille (1999). ‘Conceptual structures represented by conceptual graphs and formal concept analysis’. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics).* Vol. 1640. Springer Verlag, pp. 423–441.

Miron, Elena Teodora, Christian Muck, Dimitris Karagiannis and David Götzinger (Jan. 2018). ‘Transforming storyboards into diagrammatic models’. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics).* Vol. 10871 LNAI, pp. 770–773.

Montes-Y-Gómez, M., Alexander Gelbukh, Aurelio López-López and R. Baeza-Yates (2001). ‘Flexible comparison of conceptual graphs’. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics).* Vol. 2113, pp. 102–111.

Moore, James Frederick (1993). ‘Predators and prey: a new ecology of competition.’ In: *Harvard Business Review* 71.3, pp. 75–86.

Moser, Christoph, Robert Andrei Buchmann, Wilfrid Utz and Dimitris Karagiannis (2017). 'CE-SIB: A modelling method plug-in for managing standards in enterprise architectures'. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics).*

Muck, Christian, Elena-Teodora Miron, Dimitris Karagiannis and Lee Moonkun (Sept. 2018). 'Supporting Service Design with Storyboards and Diagrammatic Models: The Scene2Model Tool'. In: *Joint International Conference of Service Science and Innovation (ICSSI 2018) and Serviceology (ICServ 2018).*

Mylopoulos, John (1992). 'Conceptual modelling and Telos'. In: *Conceptual Modeling, Databases, and Case An integrated view of information systems development.* pp. 49–68.

Mylopoulos, John, Alex Borgida, Matthias Jarke and Manolis Koubarakis (1990). 'Telos: Representing Knowledge About Information Systems'. In: *ACM Transactions on Information Systems (TOIS).*

Negash, Solomon and Paul Gray (2008). 'Business Intelligence'. In: *Handbook on Decision Support Systems 2.* Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 175–193. URL: http://link.springer.com/10.1007/978-3-540-48716-6%7B%5C_%7D9.

Nguyen, Philip H.P. and Dan Corbett (2006). 'A basic mathematical framework for conceptual graphs'. In: *IEEE Transactions on Knowledge and Data Engineering.*

Object Management Group (OMG) (2011a). 'Business Process Model and Notation (BPMN) Version 2.0'. In:

Object Management Group (OMG) (2011b). *Unified Modeling Language - Superstructure, 2.4.1.* Tech. rep., p. 748.

OECD (2019). *Measuring the Digital Transformation.*

Olle, T. William., Jacques Hagelstein, Ian G. Macdonald, Colette Rollands, Henk G. Sol, Frans J. M. Van Assche and Alexander A. Verrijn-Stuart (1988). *Information Systems Methodologies: A Framework for Understanding (2nd Edition).* 2. Addison-Wesley Pub. Co, p. 268.

Open Group (2019). *ArchiMate® 3.1 Specification.* URL: https://pubs.opengroup.org/architecture/archimate3-doc/ (visited on 14/11/2019).

Österle, Hubert et al. (Sept. 2010). 'Memorandum zur gestaltungsorientierten Wirtschaftsinformatik'. In: *Schmalenbachs Zeitschrift für betriebswirtschaftliche Forschung* 62.6, pp. 664–672.

Osterwalder, Alexander and Yves Pigneur (2010). *Business Model Generation - Canvas.*

Pacione, Michael J., Marc Roper and Murray Wood (2004). 'A novel software visualisation model to support software comprehension'. In: *Proceedings - Working Conference on Reverse Engineering, WCRE*, pp. 70–79.

Patwardhan, Siddharth, Satanjeev Banerjee and Ted Pedersen (2003). 'Using measures of semantic relatedness for word sense disambiguation'. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 2588, pp. 241–257.

Pedersen, Ted, Siddharth Patwardhan and Jason Michelizzi (2004). *WordNet:: Similarity - Measuring the Relatedness of Concepts.* Tech. rep.

Petre, Marian and Ed De Quincey (2006). 'A gentle overview of software visualisation'. In: *Engineering* September, pp. 1–10.

Petry, Martin (2019). 'Intelligent Customer Interactions Require an Intelligent Enterprise Architecture'. In: *Presentation at NEMO Summerschool 2019.* Hilti, pp. 1–18.

Plattner, Hasso, Christoph Meinel and Larry J. Leifer (2011). *Design thinking : understand, improve, apply*, p. 236.

Poole, Jonathan and J. A. Campbell (1995). 'A novel algorithm for matching conceptual and related graphs'. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics).* Vol. 954, pp. 293–307.

Prat, Nicolas, Isabelle Comyn-Wattiau and Jacky Akoka (2014). 'Artifact Evaluation in Information Systems Design-Science Research - A Holistic View'. In: *Proceedings - 2014 Pacific Asia Conference on Information Systems (PACIS).* AIS (Association for Information Systems).

PriceWaterhouseCooper (2019). *Global Top 100 Companies by market capitalisation*, p. 36.

Ralyté, Jolita and Colette Rolland (2001). 'An assembly process model for method engineering'. In: *Lecture Notes in Computer Science (including* *subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics).*

Reimann, Peter and Wilfrid Utz (2016). 'Modeling learning data for feedback and assessment'. In: *Domain-Specific Conceptual Modeling: Concepts, Methods and Tools.* Cham: Springer International Publishing, pp. 555–574.

Rohr, Matthias, André Van Hoorn, Jasminka Matevska, Nils Sommer, Lena Stoever, Simon Giesecke and Wilhelm Hasselbring (2008). 'Kieker: continuous monitoring and on demand visualization of java software behavior'. In: *Proceedings of the IASTED International Conference on Software Engineering, SE 2008*, pp. 80–85.

Rospocher, Marco, Chiara Ghidini and Luciano Serafini (2014). 'An ontology for the business process modelling notation'. In: *Frontiers in Artificial Intelligence and Applications.*

Roussopoulos, Nick and Dimitris Karagiannis (2009). 'Conceptual modeling: Past, present and the continuum of the future'. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics).* Vol. 5600 LNCS, pp. 139–152.

Rowe, Peter G. (1987). *Design thinking.* MIT Press, p. 229.

SAP User Experience Design Services (2019). *Scenes.* URL: https://experience.sap.com/designservices/approach/scenes (visited on 07/05/2019).

Selway, Matt, Markus Stumptner, Wolfgang Mayer, Andreas Jordan, Georg Grossmann and Michael Schrefl (2015). 'A conceptual framework for large-scale ecosystem interoperability'. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics).* Vol. 9381. Springer Verlag, pp. 287–301.

Silver, Mark S., M. Lynne Markus and Cynthia Mathis Beath (1995). 'The information technology interaction model: A foundation for the MBA core course'. In: *MIS Quarterly: Management Information Systems.*

Sowa, John F. (1979). 'Semantics of conceptual graphs'. In:

Sowa, John F. (1984). *Information Processing in Mind and Machine.*

Sowa, John F. (2009). 'Conceptual Graphs for Representing Conceptual Structures'. In:

Stachowiak, Herbert (1973). *Allgemeine Modelltheorie.*

Strahringer, Susanne (1995). *Zum Begriff des Metamodells.* Tech. rep.

Strahringer, Susanne (1998). 'Ein sprachbasierter Metamodellbegriff und seine Verallgemeinerung durch das Konzept des Metaisierungsprinzips.' In: *CEUR Workshop Proceedings zur Modellierung '98*, pp. 1–6.

Thalheim, Bernhard (2011). 'The Theory of Conceptual Models, the Theory of Conceptual Modelling and Foundations of Conceptual Modelling'. In: *Handbook of Conceptual Modeling.* Springer Berlin Heidelberg, pp. 543–577.

Thomas, Oliver (May 2005). 'Das Modellverständnis in der Wirtschaftsinformatik : Historie , Literaturanalyse und Begriffsexplikation'. In: Institut für Wirtschaftsinformatik (IWi) im Deutschen Forschungszentrum für Künstliche Intelligenz (DFKI GmbH), Universität des Saarlandes, Saarbrücken - Publikationen.

UNCTAD (2019). *Digital Economy Report 2019 Value Creation and Capture: Implications for Developing Countries.*

Utz, Wilfrid (2018a). *D4.2 ZDM Data and Management Environment Implementation.* Tech. rep.

Utz, Wilfrid (2018b). 'Design metamodels for domain-specific modelling methods using conceptual structures'. In: *CEUR Workshop Proceedings.* Vol. 2234, pp. 47–60.

Utz, Wilfrid (2019a). 'Design of a Domain-Specific Metamodel for Industrial Business Process Management'. In: *Proceedings - 2019 8th International Congress on Advanced Applied Informatics (IIAI-AAI).* Vol. 2019. Toyama: IIAI, pp. 821–826.

Utz, Wilfrid (2019b). 'Support of Collaborative Design Thinking using ADOxx'. In: *Proceedings - 2019 International Conference on Innovation and Management.* Hiroshima.

Utz, Wilfrid and Damiano Falcioni (Sept. 2018). 'Data Assets for Decision Support in Multi -Stage Production Systems Industrial Business Process Management using ADOxx'. In: *Proceedings - IEEE 16th International Conference on Industrial Informatics, INDIN 2018.* Institute of Electrical and Electronics Engineers Inc., pp. 809–814.

Utz, Wilfrid and Dimitris Karagiannis (2009). 'Towards business and it alignment in the future internet; managing complexity in e-business'. In: *Proceedings - 2009 1st International Conference on Advances in Future Internet, AFIN 2009*.

Utz, Wilfrid and Moonkun Lee (July 2017). 'Industrial Business Process Management Using Adonis Towards a Modular Business Process Modelling Method for Zero-Defect-Manufacturing'. In: *2017 International Conference on Industrial Engineering, Management Science and Application, ICIMSA 2017*. Institute of Electrical and Electronics Engineers Inc.

Utz, Wilfrid, Peter Reimann and Dimitris Karagiannis (2014). 'Capturing learning activities in heterogeneous environments: A model-based approach for data marshalling'. In: *Proceedings - IEEE 14th International Conference on Advanced Learning Technologies, ICALT 2014*.

Utz, Wilfrid and Robert Woitsch (2017). 'A model-based environment for data services: Energy-aware behavioral triggering using ADOxx'. In: *IFIP Advances in Information and Communication Technology*.

Utz, Wilfrid, Robert Woitsch, Damiano Falcioni and Cristina Cristalli (2018). *D4.1 ZDM Data and Knowledge Management Environment Specification*. URL: `http://go0dman-project.eu/wp-content/uploads/2016/10/Abstract-D4.1-ZDM-Data-and-Management-Environment-Specification.pdf` (visited on 13/03/2018).

Utz, Wilfrid, Robert Woitsch and Dimitris Karagiannis (2011). 'Conceptualisation of hybrid service models: An open models approach'. In: *Proceedings - International Computer Software and Applications Conference*.

Utz, Wilfrid, Robert Woitsch and Zbigniew Misiak (Nov. 2016). 'Planning for integration: A meta-modelling approach using ADOxx'. In: *Measuring and Visualizing Learning in the Information-Rich Classroom*. Routledge, pp. 183–195.

Vázquez, José Manuel González (2012). *Ein Referenzmodellkatalog für die Energiewirtschaft : Management von Informationsmodellen für Softwareproduktmanager bei der Anforderungsanalyse*. Oldenburg: Dissertation Thesis, p. 376.

Višić, Nikša (2016). *Language-Oriented Modeling Method Engineering*. University of Vienna, PhD Thesis, p. 253.

Visic, Niksa, Hans Georg Fill, Robert Andrei Buchmann and Dimitris Karagiannis (2015). ‘A domain-specific language for modeling method definition: From requirements to grammar’. In: *Proceedings - International Conference on Research Challenges in Information Science.*

Visic, Niksa and Dimitris Karagiannis (2014). ‘Developing conceptual modeling tools using a DSL’. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics).*

Vom Brocke, Jan (2004). ‘Internetbasierte Referenzmodellierung - State-of-the-art und Entwicklungsperspektiven’. In: *Wirtschaftsinformatik.*

Walch, Michael (2019). *A conceptual modelling approach for design and use in cyber-physical environments: the s*IoT modelling method.* Vienna, Austria: University of Vienna, PhD Thesis, p. 282.

Walch, Michael and Dimitris Karagiannis (Jan. 2019). ‘How to connect design thinking and cyber-physical systems: the s*IoT conceptual modelling approach’. In: *Proceedings of the 52nd Hawaii International Conference on System Sciences*, pp. 7242–7251.

Walter, Tobias and Jürgen Ebert (2009). ‘Combining DSLs and ontologies using metamodel integration’. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics).* Vol. 5658 LNCS, pp. 148–169.

Wand, Yair (1996). ‘Ontology as a foundation for meta-modelling and method engineering’. In: *Information and Software Technology* 38.4 SPEC. ISS. pp. 281–287.

Wand, Yair, David E. Monarchi, Jeffrey Parsons and Carson C. Woo (1995). ‘Theoretical foundations for conceptual modelling in information systems development’. In: *Decision Support Systems* 15.4, pp. 285–304.

Wang, Lidong and Xiaodong Liu (2008). ‘A new model of evaluating concept similarity’. In: *Knowledge-Based Systems.*

Wermelinger, Michel (1995). ‘Conceptual graphs and first-order logic’. In: *Conceptual Structures: Applications, Implementation and Theory: Third International Conference on Conceptual Structures.* Santa Cruz: Springer Berlin Heidelberg, pp. 323–337.

Wieringa, Roel J. (Jan. 2014). *Design science methodology: For information systems and software engineering.* Springer Berlin Heidelberg, pp. 1–332.

Woitsch, Robert (2013). 'Hybrid modelling with ADOxx: Virtual enterprise interoperability using meta models'. In: *Lecture Notes in Business Information Processing.* Vol. 148 LNBIP. Springer Verlag, pp. 298–303.

Woitsch, Robert and Wilfrid Utz (Feb. 2016). 'Business Process as a Service: Model Based Business and IT Cloud Alignment as a Cloud Offering'. In: *Proceedings - 2015 3rd International Conference on Enterprise Systems, ES 2015.* Institute of Electrical and Electronics Engineers Inc., pp. 121–130.

World Economic Forum (2019). *Shaping the Future of Digital Economy and Society.* URL: `https://www.weforum.org/platforms/shaping-the-future-of-digital-economy-and-new-value-creation` (visited on 06/11/2019).

Wu, Zhibiao and Martha Palmer (1994). 'Verbs semantics and lexical selection'. In: *Proceedings of the 32nd annual meeting on Association for Computational Linguistics -.* Morristown, NJ, USA: Association for Computational Linguistics (ACL), pp. 133–138. arXiv: `9406033 [cmp-lg]`.

Xia, Yong and Martin Glinz (2003). 'Rigorous EBNF-based definition for a graphic modeling language'. In: *Proceedings - Asia-Pacific Software Engineering Conference, APSEC.* IEEE Computer Society, pp. 186–196.

Yoo, Youngjin, Richard J. Boland, Kalle Lyytinen and Ann Majchrzak (2012). 'Organizing for innovation in the digitized world'. In: *Organization Science.*

Yoo, Youngjin, Ola Henfridsson and Kalle Lyytinen (2010). 'The new organizing logic of digital innovation: An agenda for information systems research'. In: *Information Systems Research.*

Zhang, Kang, Peter Eades and Peter Young (Nov. 1996). *Software Visualisation.* Vol. 7. Series on Software Engineering and Knowledge Engineering vol. 7. World Scientific, pp. 1–23.

Zhu, Ganggao and Carlos A Iglesias (2017). 'Computing Semantic Similarity of Concepts in Knowledge Graphs'. In: *IEEE Transactions on Knowledge and Data Engineering* 29.1, pp. 72–85.

Živkovic, Srdan and Dimitris Karagiannis (2015). 'Towards metamodelling-in-the-large: Interface-based composition for modular metamodel development'. In: *Lecture Notes in Business Information Processing*. Vol. 214. Springer, Cham, pp. 413–428.

Zor, Sema, David Schumm and Frank Leymann (2011). 'A Proposal of BPMN Extensions for the Manufacturing Domain'. In: *3rd International Conference on Model-Driven Engineering and Software Development (MODELSWARD)* 16.1, pp. 468–481.

Zwarts, Joost and Henk Verkuyl (Feb. 1994). 'An algebra of conceptual structure; An investigation into Jackendoff's conceptual semantics'. In: *Linguistics and Philosophy* 17.1, pp. 1–28.

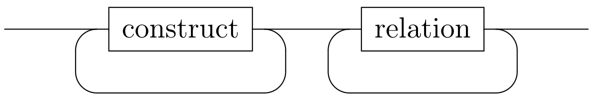
Zwass, Vladimir (2017). 'Information System'. In: *Encyclopædia Britannica*. URL: https://www.britannica.com/topic/information-system.

Appendix A

CoChaCo EBNF Railroad Diagrams

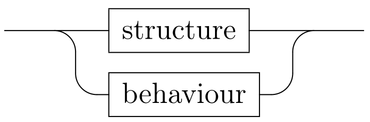
The grammar derived from CoChaCo is graphically shown in this annex, applying the EBNF format to describe context-free grammars. The definition within this appendix is provided as a baseline for the syntactical definition of a metamodel. Railroad diagrams are provided for better readability. Terminals have been introduced only for aspects related to relation definitions and endpoints and are disregarded for compositions (e.g. sets of constructs and relations).

metamodel



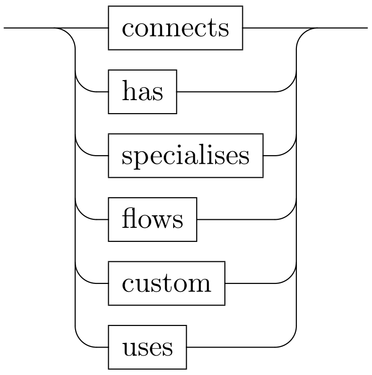
⟨metamodel⟩ ::= ⟨construct⟩+ ⟨relation⟩+

construct



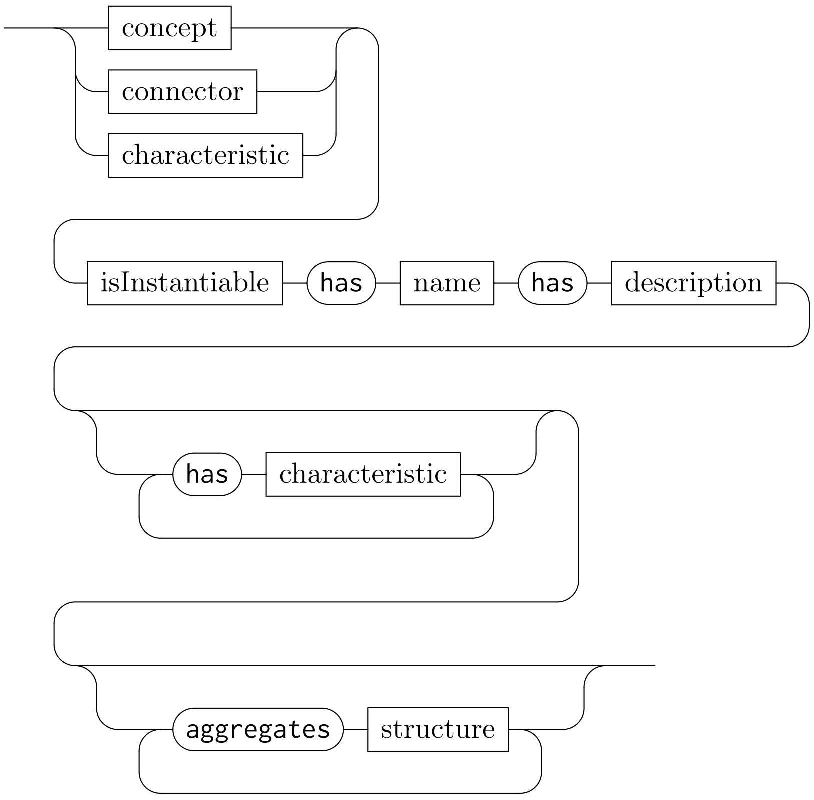
⟨construct⟩ ::= ⟨structure⟩ | ⟨behaviour⟩

relation



⟨relation⟩ ::= ⟨connects⟩ | ⟨has⟩ | ⟨specialises⟩ |
⟨flows⟩ | ⟨custom⟩ | ⟨uses⟩

structure



⟨structure⟩ ::= (⟨concept⟩ | ⟨connector⟩ | ⟨characteristic⟩)

⟨isInstantiable⟩

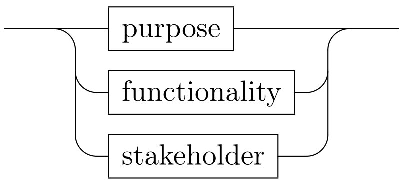
`has` ⟨name⟩

`has` ⟨description⟩

(`aggregates` ⟨structure⟩)∗

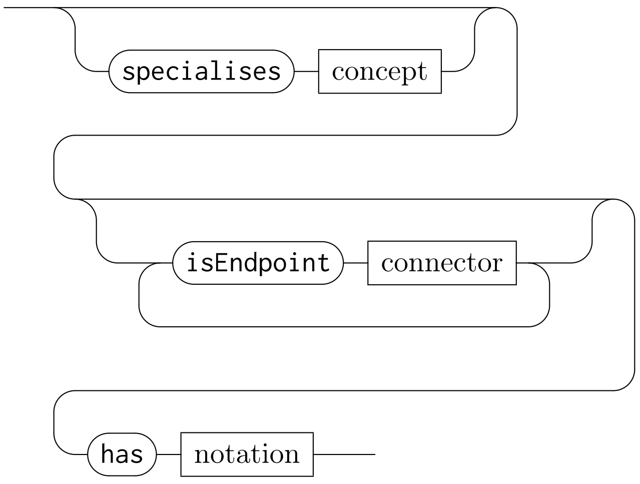
(`has` ⟨characteristic⟩)∗

behaviour



⟨behaviour⟩ ::= ⟨functionality⟩ | ⟨stakeholder⟩ | ⟨purpose⟩

concept

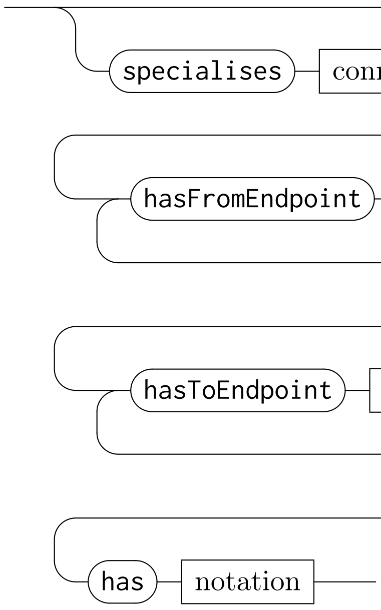


⟨concept⟩ ::= `specialises`⟨concept⟩)?

(`isEndpoint`⟨connector⟩)*

(`has`⟨notation⟩)

connector



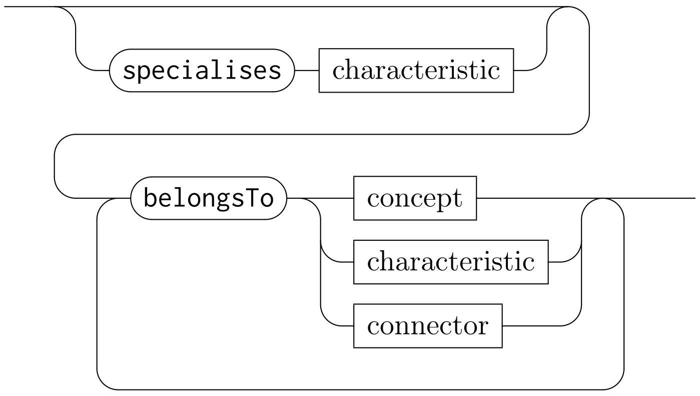
⟨connector⟩ ::= `specialises`⟨connector⟩)?

(`hasFromEndpoint`⟨concept⟩)+

(`hasToEndpoint`⟨concept⟩)+

(`has`⟨notation⟩)

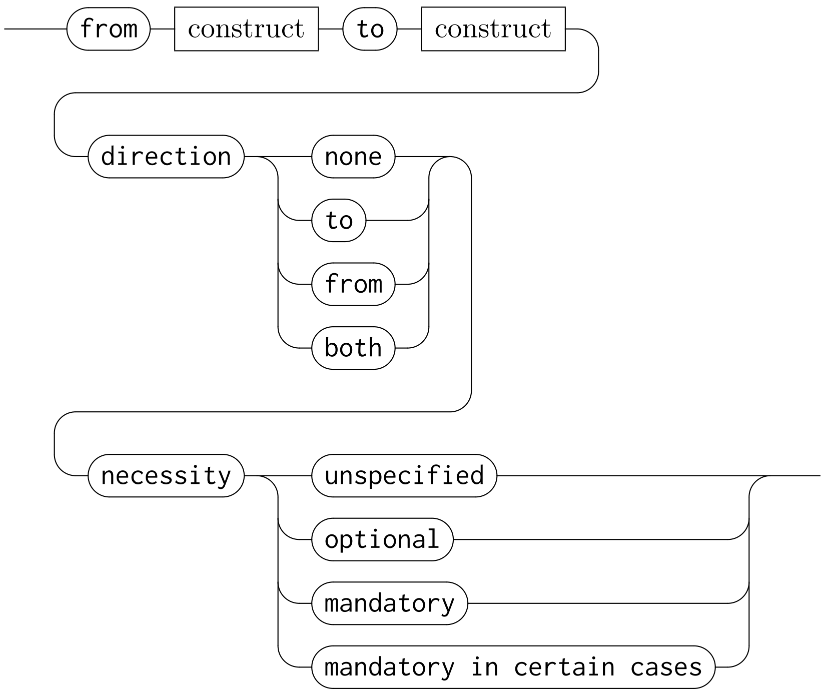
characteristic



⟨characteristic⟩ ::= `specialises`⟨characteristic⟩)?

(`belongsTo`(⟨concept⟩ | ⟨characteristic⟩ | ⟨connector⟩)+

connects



⟨connects⟩ ::= `from`⟨construct⟩)

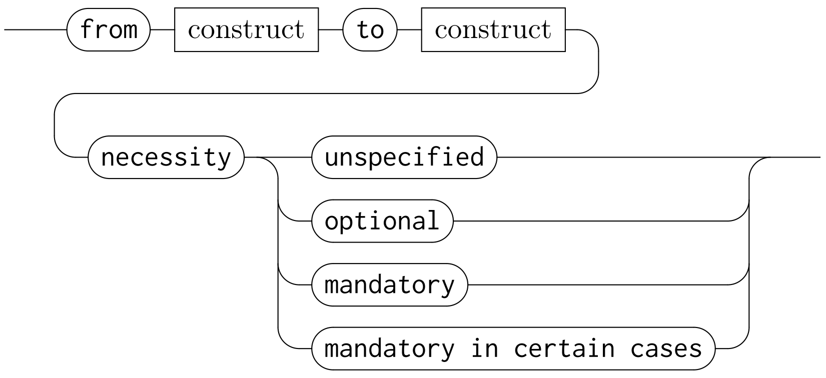
`to`⟨construct⟩

`direction`(`none` | `to` | `from` | `both`)

`necessity`(`none` | `unspecified` | `optional` |

`mandatory` | `mandatory in certain cases`)

specialises



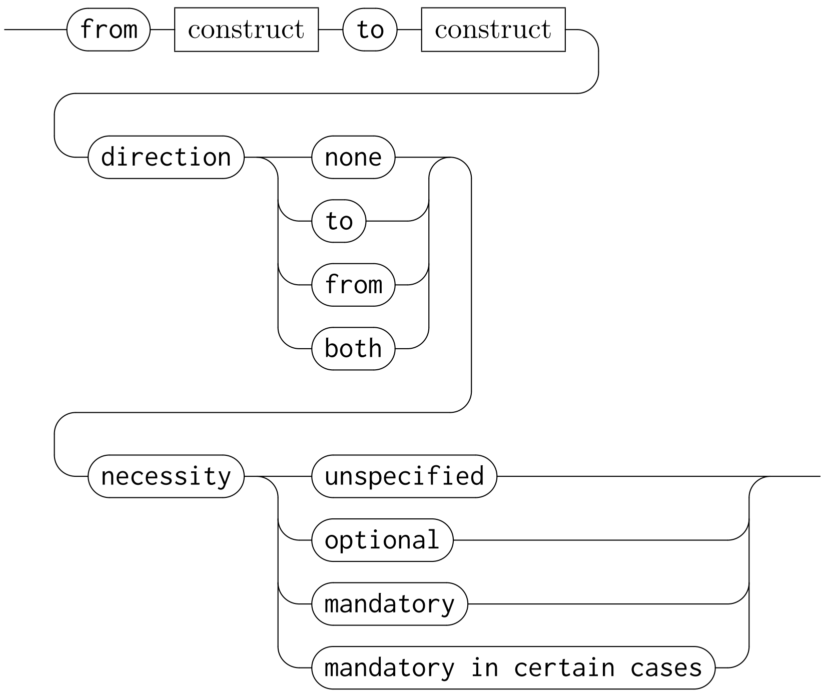
⟨specialises⟩ ::= `from`⟨construct⟩)

`to`⟨construct⟩

`necessity`(`none` | `unspecified` | `optional` |

`mandatory` | `mandatory in certain cases`)

custom



⟨custom⟩ ::= `from`⟨construct⟩)

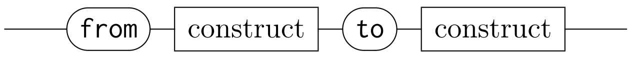
`to`⟨construct⟩

`direction`(`none` | `to` | `from` | `both`)

`necessity`(`none` | `unspecified` | `optional` |

`mandatory` | `mandatory in certain cases`)

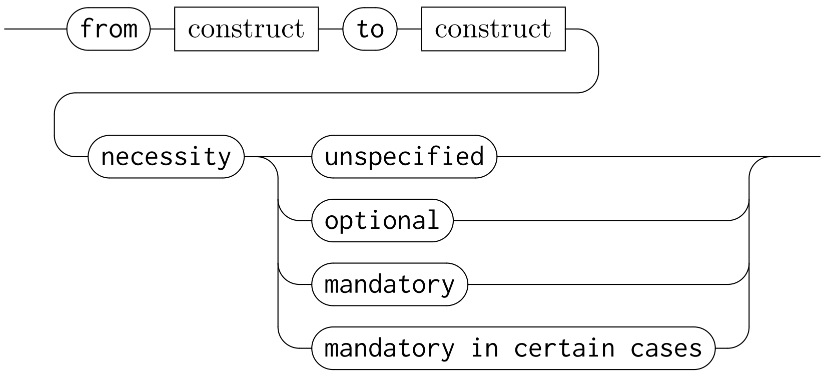
flows



⟨flows⟩ ::= `from`⟨construct⟩)

`to`⟨construct⟩

has



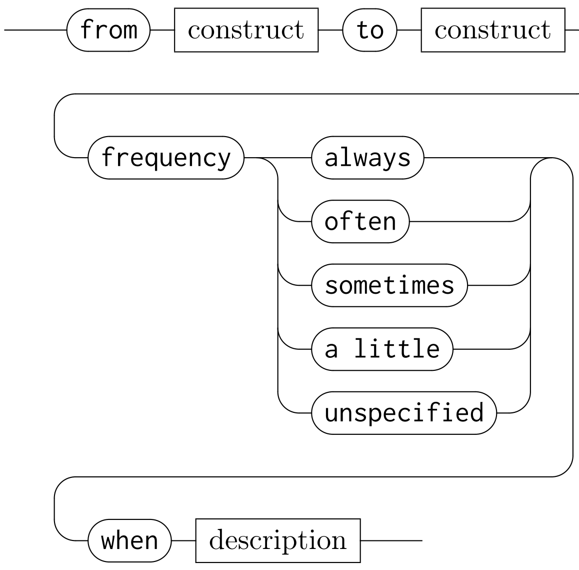
⟨has⟩ ::= `from`⟨construct⟩)

`to`⟨construct⟩

`necessity`(`none` | `unspecified` | `optional` |

`mandatory` | `mandatory in certain cases`)

uses



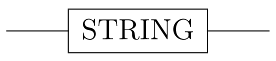
⟨uses⟩ ::= `from`⟨construct⟩)

`to`⟨construct⟩

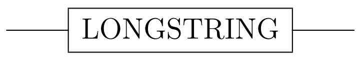
`frequency`(`always` | `ofter` | `sometimes` |

`a little` | `unspecified`)

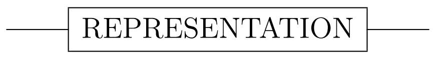
name



description



notation



Appendix B

Prototype Technology

Within this chapter, the prototype technologies selected are introduced and code fragments for the resulting service implementation as building blocks of the DeMoMa::* building blocks are provided.

B.1 Overview Prototype Technologies

This list summarises the technologies selected and used for the realisation of the proof-of-concept implementation during the evaluation phase of the research project. The collected technologies have been assess by the criteria of a) functional coverage (whether and to what extend the required functionality can be realised), b) openness (to extensions and modification) and c) community engagement.

As this assessment always historic and never complete, a living document is established at `https://gitlab.dke.univie.ac.at/mm_conceptual_graphs/technology_assessment` that summarises the changing landscape of applicable technologies. The following list provides an overview on the technology concretely used for the realisation of the building blocks and services

A Development Environments

The following IDEs have been used during the implementation:

1. The Microsoft Visual Studio Code `https://code.visualstudio.com/` environment has been selected for the realisation of the DeMoMa::IDE due to its growing popularity and ease of use to integrate extensions. The integration and deployment process is support by the marketplace `https://marketplace.visualstudio.com/vscode` of extensions and modules. Specifically the AdoScript extension has been used and extended.

2. Eclipse Java Development Environment `https://www.eclipse.org/downloads/packages/` is applicable for the development of the service interactions (functionality, logic and endpoints)

3. Eclipse Language Workbench `https://www.eclipse.org/Xtext/` has been used to realise the language and grammar, including generators. The workbench enables packaging of the language server and integration in Microsoft Visual Studio Code.

4. CoGUI platform `https://www.lirmm.fr/cogui/` provides a graphical UI to develop and operate conceptual graph functionality.

B Development Languages

The following development languages and frameworks are utilised for the implementation of the prototype:

1. Java `https://www.oracle.com/java/` is the language of choice to implement the service logic. This is due to the experience and availability of modules to support the implementation (libraries, extensions and build system),

2. XText `https://www.xtext.org` to realise the abstract syntax tree and parser of the CoChaCo grammar,

3. XTend `http://www.xtend-lang.org` to implement the generator for import/export and transformation logic,

4. Maven `http://maven.apache.org/` applied for dependency analysis and dynamic loading of libraries as well as continuous integration,

5. JAX-RS `https://github.com/jax-rs` uses as a framework to generate service endpoint dynamically based on Java annotations,

6. XSL `https://www.w3.org/standards/xml/transformation` as an embedded language to perform model transformations.

B.2 Prototype Code Fragments

In this section the most relevant code fragment required for the realisation of the prototype are listed. All prototype implementation results can be download at https://gitlab.dke.univie.ac.at/mm_conceptual_graphs.

The code fragment below in listing B.1 shows the implementation of the grammar in XText. The definition of the grammar is used to generate a) the language server (for syntax highlighting, code completion and cross-referencing) and is input for the code generator.

```
grammar org.adoxx.cochaco.Metamodel with
   org.eclipse.xtext.common.Terminals

generate metamodel
   "http://www.adoxx.org/cochaco/Metamodel"

Metamodel:
   'namespace' namespace=STRING ';' NL+
   (constructs+=(StructuralConstruct |
      BehaviouralConstruct))+
   (relations+=Relation)*;


StructuralConstruct:
  (Concept | Connector | Characteristic)
     ('isInstantiable')? ('hasName' className=STRING)
     ('hasDescription' classDescription=STRING)
     ('has' '[' characteristic+=[Characteristic]+
     ']')? ('aggregates' '['
     composition+=[StructuralConstruct]+ ']')? ';'
     NL+;


BehaviouralConstruct:
  Purpose | Functionality | Stakeholder ';' NL+;


Relation:
  virtual_relation ';' WS;

virtual_relation:
  'virtual_relation' name=ID ('specialises' '['
     superType+=[Connector]+ ']')? ('hasFromEndpoint'
     from=[Concept]) ('hasToEndpoint' to=[Concept])
     ('hasNotation' notation=STRING)?;

Concept:
    'concept' name=ID ('specialises' '['
      superType+=[Concept]+ ']')? ('hasNotation'
      notation=STRING)?;

Connector:
  'connector' name=ID ('specialises' '['
     superType+=[Connector]+ ']')? ('hasFromEndpoint'
     from=[Concept]) ('hasToEndpoint' to=[Concept])
     ('hasNotation' notation=STRING)?;

Characteristic:
  'characteristic' name=ID ('specialises' '['
     superType+=[Characteristic]+ ']')?;


Purpose:
  'purpose' name=ID ('specialises' '['
     superType+=[Purpose]+ ']')?;

Functionality:
  'functionality' name=ID ('specialises' '['
     superType+=[Functionality]+ ']')?;

Stakeholder:
  'stakeholder' name=ID ('specialises' '['
     superType+=[Stakeholder]+ ']')?;

terminal NL : ('\r'? '\n')+;
```

Listing B.1: CoChaCo Grammar in XText

The generator towards conceptual graphs uses the above grammar as an input and performs three transformations in parallel on every valid change of the source file:

1. *.voc file: this file represents the abstract vocabulary the metamodel is build upon
2. *.fact file: references the concept and relation types in the vocabulary and represents the implemented metamodel
3. *.adl file: represents two CoChaCo4ADOxx Models (hierarchy and metamodel)

```
package org.adoxx.cochaco.generator

import org.eclipse.emf.ecore.resource.Resource
import org.eclipse.xtext.generator.AbstractGenerator
import org.eclipse.xtext.generator.IFileSystemAccess2
import org.eclipse.xtext.generator.IGeneratorContext
import org.adoxx.cochaco.metamodel.Concept
import java.util.UUID
import org.adoxx.cochaco.metamodel.Metamodel
import org.adoxx.cochaco.metamodel.Connector

/**
 * Generates ADL and CoGUI code from your metamodel
   files on save.
 *
 */
class MetamodelGenerator extends AbstractGenerator {

  static String namespace = "";

  override void doGenerate(Resource resource,
     IFileSystemAccess2 fsa, IGeneratorContext
     context) {
    fsa.generateFile(resource.URI.trimFileExtension
                  .lastSegment + ".voc",
                     resource.compileCGVoc)
    fsa.generateFile(resource.URI.trimFileExtension
                .lastSegment + ".fact",
                   resource.compileCG)
    fsa.generateFile(resource.URI.trimFileExtension
                    .lastSegment + ".adl",
                       resource.compileADL)
    namespace = resource.allContents.toList
              .filter(Metamodel)
              .get(0)
              .namespace.toLowerCase;
  }

  def compileCGVoc(Resource r) '''
    <?xml version="1.0" encoding="UTF-8"
       standalone="no"?>
    <cogxml>
    <namespace
       name="http://www.adoxx.org/cochaco/«namespace»/#"
       prefix="«namespace»"/>
    <support name="vocabulary">
    <conceptTypes>
      «FOR c :
         r.allContents.toIterable.filter(Concept)»
        «c.compileCGConcept»
      «ENDFOR»
      «FOR c :
         r.allContents.toIterable.filter(Concept)»
        «c.compileCGConceptHierarchy»
      «ENDFOR»
    </conceptTypes>
    <relationTypes>
      «FOR c :
         r.allContents.toIterable.filter(Connector)»
        «c.compileCGConnector»
      «ENDFOR»
      «FOR c :
         r.allContents.toIterable.filter(Connector)»
        «c.compileCGConnectorHierarchy»
      «ENDFOR»
    </relationTypes>
    <nestingTypes>
    </nestingTypes>
    <conformity/>
    <modules/>
    </support>
    </cogxml>
   '''

  def compileCG(Resource r) '''
    <?xml version="1.0" encoding="UTF-8"
       standalone="no"?>
    <cogxml>
    <namespace
       name="http://www.adoxx.org/cochaco/«namespace»/#"
       prefix="«namespace»"/>
    <support name="vocabulary">
    <conceptTypes>
      «FOR c :
         r.allContents.toIterable.filter(Concept)»
        «c.compileCGConcept»

      «ENDFOR»
      «FOR c :
         r.allContents.toIterable.filter(Concept)»
        «c.compileCGConceptHierarchy»
      «ENDFOR»
    </conceptTypes>
    <relationTypes>
      «FOR c :
         r.allContents.toIterable.filter(Connector)»
        «c.compileCGConnector»
      «ENDFOR»
      «FOR c :
         r.allContents.toIterable.filter(Connector)»
        «c.compileCGConnectorHierarchy»
      «ENDFOR»
    </relationTypes>
    <nestingTypes>
    </nestingTypes>
    <conformity/>
    <modules/>
    </support>
    </cogxml>
   '''

  def compileCGConceptHierarchy(Concept c) '''
    «FOR superType : c.superType»
      <order
         id2="«generateUniqueName(superType.name.getBytes())»"
             id1="«generateUniqueName(c.name.getBytes())»"/>
    «ENDFOR»

  '''

  def compileCGConnectorHierarchy(Connector c) '''
    «FOR superType : c.superType»
      <order
         id2="«generateUniqueName(superType.name.getBytes())»"
             id1="«generateUniqueName(c.name.getBytes())»"/>
    «ENDFOR»

  '''

  def compileCGConcept(Concept c) '''
    <ctype
       id="«generateUniqueName(c.name.getBytes())»"
     label="«c.name»"
     namespace="«namespace»">
    <translation descr="«c.getClassDescription»"
     label="«c.name»"
     lang="en"/>
    </ctype>
  '''

  def compileCGConnector(Connector c) '''
    <rtype
       id="«generateUniqueName(c.name.getBytes())»"
     label="«c.name»" namespace="«namespace»">
    <translation descr="«c.getClassDescription»"
     label="«c.name»"
     lang="en"/>
    </rtype>
  '''

  /* CoChaCo ADL Converter */
  def compileADL(Resource r) '''
    VERSION <5.1>
    VOCABULARY
       <«r.URI.trimFileExtension.lastSegment»> :
       <CoChaCo - Prototype Dynamic 0.5>
    VERSION <0.1>
    TYPE <Concept overview>
    «FOR c : r.allContents.toIterable.filter(Concept)»
      «c.compileADLConcept»

    «ENDFOR»
  '''

  def compileADLConcept(Concept c) '''
    INSTANCE <«c.name»> : <Concept>
  '''

  def String generateUniqueName(byte[] bytes) {
    return generateUniqueName(bytes).toString()
  }

}
```

Listing B.2: CoChaCo CG Generator in XTend

Appendix C

OMiLAB Modelling Methods

The following tables classify the modelling methods, languages and/or notations available in the OMiLAB at the University of Vienna. All resources developed by the community are publicly available at `https://austria.omilab.org/psm/exploreprojects`.

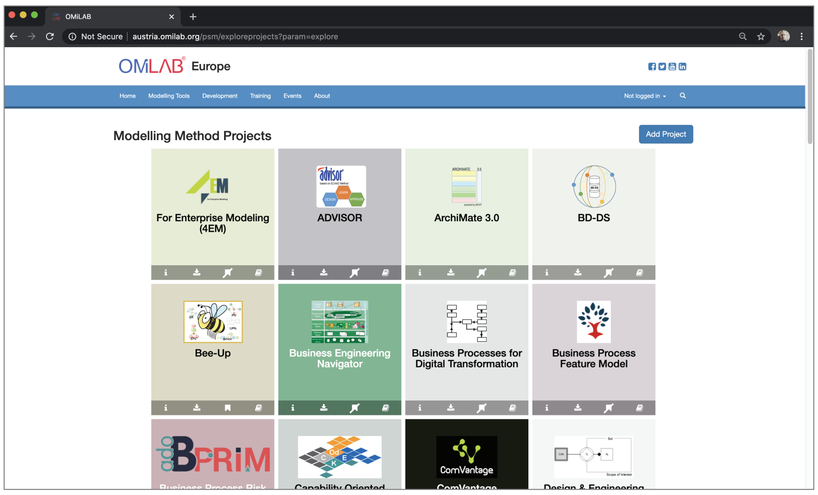

OMiLAB
Not Secure
austria.omilab.org/psm/exploreprojects?param=explore
OMiLAB Europe
Home
Modelling Tools
Development
Training
Events
About
Not logged in
Modelling Method Projects
Add Project
For Enterprise Modeling (4EM)
ADVISOR
ArchiMate 3.0
BD-DS
Bee-Up
Business Engineering Navigator
Business Processes for Digital Transformation
Business Process Feature Model
BPRIM
ComVantage

Figure C.1: OMILAB Project Space: Modelling Methods and Tools

In an initial step the modelling methods/languages/notations (an assessment and categorisation is not performed), organisational background and self-declared description as the purpose of the language are listed (Fig. C.1). The assessment of modelling methods has been performed in two iterations (November 2019 for technical evaluation and testing and April 2020 for completeness evaluation) resulting in 56 implementation artefacts collected.

Following this collection of projects and meta-information about them, the download and extraction phase was conducted to structure the evaluation dataset. The set consists of 50 library files that are used to assess the completeness and adequacy of the conceptual design and prototypical realisation.

ID	**Project**	**Akronym**	**Organisation**	**Description (self-declared)**
1	4EM	4EM	University of Rostock Stockholm University University of Skovde	The 4EM method is comprised of three core elements, which can also be regarded as basic principles and are closely interwoven:- A defined procedure to modeling using a fixed notation (defined procedure and notation)- Performance of Enterprise Modeling in the form of a project with predetermined roles (project organization and roles)- A participatory process to involve enterprise stakeholders and domain experts (stakeholder participation)
2	BPRIM	BPRIM	ISITCom Tunisia - INP Toulouse	BPRIM (Business Process Risk Management – Integrated Method) is a methodological framework that consists of the BPRIM process, the BPRIM conceptual models and the BPRIM modelling language.
3	ADVISOR	ADVISOR	University of Syndney	The ADVISOR tool implements the ECAAD (Evidence-Centered Actvitiy and Assessment Design) method for learning design and management.
4	ArchiMate	ArchiMate	York University University of Vienna	Enterprise Architecture Modelling using ArchiMate 3.0

ID	Project	Akronym	Organisation	Description (self-declared)
5	BD-DS	BD-DS	University of Maryland	Big Data – Data Service (BD-DS) enables to semantically map data sources of different kinds, formats and representations with data demands on application/business level using conceptual models as a means for mediation.
6	Bee-Up	Bee-Up	University of Vienna	Bee-Up is a prototypical implementation which supports the BPMN, EPC, ER, UML and Petri Nets modelling languages as well as additional processing functionality in one tool.
7	BEN	BEN	University of St. Gallen	The goal of this project is to provide modeling means for the holistic design and management of an organization.
8	Business Process Feature Model	BPFM	University of Camerino	Business Process Feature Model notation (BPFM) is an extended version of Feature Models named BPFM proposed to deal with Business Process Variability.
9	Business Processes for Digital Transformation	NA	University of Zagreb	Process-oriented approach to business improvement is closely related to digital technologies, whereby not all business processes have the same significance as they contribute differently to the basic pillars of digital transformation.

ID	Project	Akronym	Organisation	Description (self-declared)
10	CODEK	CODEK	University of the Aegean	A conceptual framework for capability driven development of enterprise knowledge. It uses ‘business capabilities’ as the fundamental abstraction to describe the business requirements, and then to map from capabilities to services and / or systems.
11	ComVantage	ComVantage	University of Vienna	The ComVantage modelling method aims to support the ComVantage project and its end-users, by positioning the subject under study in the wider context of supply chain management.
12	CuTiDe Large Scale Collaborative Processes	CUTiDe	University of Technology Sydney	
13	DEMO: Design & Engineering Methodology for Organizations	DEMO	University of Pretoria	DEMO was developed by Prof Jan Dietz, consisting of 4 aspect models to represent a the essence or blueprint of enterprise operation. The models are concise, consistent and comprehensive.
14	DIBA: Data Integration for Business	DIBA	University of Vienna	The DIBA method reflects a modelling language that provides the modeller with the possibility to include detailed knowledge about the data and to give support for the data transformation process by including dependencies and side effects.

ID	Project	Akronym	Organisation	Description (self-declared)
15	DICE: Data Integration and Cleansing Environment	DICE	University of Vienna	A modelling method that supports the data understanding and data preparation phases for business analytics endeavours and subsequently decision making in business process activities.
16	DICER	DICER	Politecnico di Milano	The scope of this product is modelling data-intensive applications and their deployment based on the created visual model utilizing model-to-model transformations and infrastructure-as-a-code approach.
17	DMN	DMN	KU Leuven	Decision Model and Notation
18	eduWeaver	eduWeaver	Universtät Klagenfurt	eduWeaver supports teachers by the individual design courses and the distribution of high quality multimedia teaching materials among higher educational institutes.
19	Electric Vehicle Testbed Modeler	EVTM	AVL List GmbH Alpen-Adria-Universität Klagenfurt	The Electric Vehicle Testbed Modeling Tool is used to model the structure of the high-voltage components installed in the electric vehicle testing facility.
20	ENTERKNOW	ENTER-KNOW	Babes-Bolyai University	The project aims to investigate the concept of information system aware of enterprise semantics, with the semantics captured in hybrid knowledge bases mixing models, graph databases and OWL axioms.

ID	Project	Akronym	Organisation	Description (self-declared)
21	Enterprise Knowledge Development	EKD	University of Paris	EKD is an approach that provides a controlled way of analysing and documenting an enterprise, its objectives and support systems.
22	Evaluation Chains	EC	DHBW Mannheim	The ‚evaluation chains‘ incorporate the diverse factors of the enterprise modelling process. They start with business goals, then derive essential modelling goals and link them with the crucial factors to realize the desired outcomes.
23	exemplarische Geschäftsprozessmodellierung	eGPM	Universität Hamburg	eGPM has been used in academia and industry in domains such as banking, insurance, logistics and health care. It is suitable for several purposes, including requirements engineering and organisational development.

ID	Project	Akronym	Organisation	Description (self-declared)
24	HCM-L	HCM-L	Universität Klagenfurt	HCM-L Modeler provides a comprehensive modelling tool for HCM-L based on ADOxx . In particular, the HCM-L Modeler covers the entirety of HCM-L concepts, including syntax, semantics and consistency checking. In the next development stage HCM-L Modeler modules will support complex scenarios, model optimization and advanced reasoning techniques.Various functionalities provided by ADOxx will also be available at the HCM-L Modeler interface: archiving, analyses, querying, creation of tables, user rights management, ready-made solutions for publishing opportunities, and graphical tool support for the representation and manipulation of objects.
25	Hermxx	Hermxx	Christian-Albrechts-Universität zu Kiel (CAU)	Modeling is a science and an art. Choosing the right tool(s) as an instrument is important for the communication between developers and users. This tool HERMxx allows the user to get access of the following arts of modelling for developing and designing systems and their processes.

ID	Project	Akronym	Organisation	Description (self-declared)
26	HORUS	HORUS	Karlsruhe Institute of Technology (KIT)	The Horus modeling method for business process engineering comprises steps for an integrated modeling of business processes and for the improvement and further use of the created models.
27	iStar	iStar	University of Vienna	The i* Method has been developed at the University of Toronto and facilitates the showing of social relationships and their analysis.
28	Japanese Creative Services	JCS	Kyoto University JAIST	The project analyses in-depth the value creation for Japanese Creative Services namely the areas of Japanese Cuisine, Shinise (organisations of long-standing), Traditional Cultural Activities, and Cool Japan.
29	KAMET	KAMET	Instituto Tecnológico Autónomo de México	KAMET is a methodology based on models designed to manage knowledge acquisition from multiple knowledge sources (KS). The method provides a strong mechanism to achieve knowledge acquisition in an incremental fashion, in a cooperative environment, and in a shared context for knowledge creation. KAMET represents a modernapproach to building diagnosis-specialized knowledge models.

ID	Project	Akronym	Organisation	Description (self-declared)
30	Knowledge Work Designer	KWD	University of Applied Sciences Northwestern Switzerland	The Knowledge Work Designer is a modelling method for knowledge work, which includes business logic and process logic. It offers a deep integration of BPMN and CMMN and supports the representation of decision logic with DMN.
31	LearnPAD	LeanPad	University of Camerino	The Learn PAd project supports process-driven collaborative knowledge sharing and process improvement on a user-friendly basis of wiki pages together with guidance based on formalized models.
32	MEMO4ADO	MEMO	University of Duisburg-Essen	MEMO is a method for Multi-Perspective Enterprise Modeling. MEMO encompasses a set of integrated domain-specific modeling languages to describe and interrelate varied facets of an organization's action system and information systems.
33	MoSeS4eGov MDSDL	MoSeS4eGov MDSDL	Austrian Institute of Technology	Our approach is based on the application of model-driven architecture (MDA) paradigms (Frankel, 2010) in the design and implementation of security requirements in e-Government applications.
34	MoSeS4eGov Project	MoSeS4eGov Project	Austrian Institute of Technology	Our approach is based on the application of model-driven architecture (MDA) paradigms (Frankel, 2010) in the design and implementation of security requirements in e-Government applications.

ID	Project	Akronym	Organisation	Description (self-declared)
35	MultiModal Interface Modeling Language	MMI-ML	Alpen Adria Universität Klagenfurt Carinthia University of Applied Sciences	Multimodal at-home-support for people with dementia e.g. during their morning routine.
36	MuVieMoT Conceptual Design of Multi-View Modeling Tools	MuVieMoT	University of Vienna	The MuVieMoT modeling method is aimed to help method owners and tool developers in creating Conceptual Design specifications for multi-view modeling tools.
37	Open Knowledge Models	OKM	University of Applied Sciences Northwestern Switzerland	A community which is interested in improvements of graphical knowledge modelling
38	PetriNets	PetriNets	Humboldt Universität Berlin	The main goal of this project is the implementation of a simulation tool which provides the user with the capability of describing or studying any information system with the use of Petri Nets through its simulation

ID	Project	Akronym	Organisation	Description (self-declared)
39	Petrixx	Petrixx	Humboldt-Universität zu Berlin	Petrixx enables to model and play the tokengame in order to simulate certain behaviour of a Petri Net. Plus it displays the current status of the model as a graph. Furthermore, Petrixx analyze Petri Nets through integration of the Low Level Petri Net Analyzer (LoLA). A converter functionality is provided within Petrixx to transform the active model into a format which is required by LoLA and can be stored as a file for other purposes. Also, the composition operator according to the Paper Simple Composition of Nets by Wolfgang Reisig was realized which makes Petrixx to an unique tool.
40	PGA - Process-Goal Alignment	PGA	Ghent University	It is important for a company to align the organizational strategy with its operational processes. To realize this, we developed the PGA modeling technique, which employs heat mapping in combination with performance measurement and prioritization.

ID	Project	Akronym	Organisation	Description (self-declared)
41	PRINTEPS	PRINTEPS	Keio University	PRINTEPS (PRactical INTElligent aPplicationS) is a total intelligent application development platform that integrates 5 types of sub systems (knowledge based reasoning systems, speech dialogue systems, human and environment sensing systems, and machine learning systems). PRINTEPS supports end users to participate in AI applications design (user participation design) and to develop applications easily (within hours to days) by combining software modules from PRINTEPS.
42	Probability Visualized	ProVis	University of Vienna	ProVis is a tool for the visualization of conditional probabilities and sequences of events as tree diagrams and unit squares. ProVis has been designed for the use in school to support students and teachers in creating diagrams in a fast and easy way. It enables dynamic interaction, provides reusability of diagrams, and allows for the creation of complex diagrams based on real world situations. The immediate availability of graphical representations encourages students to experiment with tree diagrams and unit squares fostering the explorative acquisition of knowledge.

ID	Project	Akronym	Organisation	Description (self-declared)
43	PSS Scenario Modeller	PS3M	Mines Saint-Etienne	PSS Scenarios Modeller aims to provide a visual representation of the Product-Service Systems (PSS) organizational scenarios and a starting point for analysing and discussing them.
44	RUPERT Process Excellence and Reengineering	RUPERT	University of Regensburg	RUPERT can be used in any quality management project in the service as well as production sector that aims at a systematic improvement of business processes.
45	SAVE	SAVE	Chonbuk National University	SAVE is a tool to model distributed mobile real-time systems with d-Calculus and GTS Logic in a visual specification and verification environment. It consists of ITL and ITS Modelers, EM Generator, Simulator and Analyzer.
46	Scene2Model	S2M	University of Vienna	Product, service and business model innovation using design thinking methods and tools
47	Secure Tropos	SecTro	University of Brighton	Secure Tropos provides a modelling language that represents security requirements through security constraints, allowing developers to model multi-agent, software systems and their organisational environment.

ID	Project	Akronym	Organisation	Description (self-declared)
48	Semantic Database Design (SDBD)	SDBD	University of Maryland	The SDBD method assumes that the primary purpose of the future system is to automate current or planned activities of the enterprise.
49	Semantic Object Model (SOM)	SOM	University of Vienna	SOM is a comprehensive methodology for modeling business systems. The acronym means Semantic Object Model, expressing that the SOM methodology is both fully object-oriented and designed to capture business semantics explicitly.

ID	Project	Akronym	Organisation	Description (self-declared)
50	Semantic-based Modeling Framework for Information Systems (SEMFIS)	SEMFIS	University of Fribourg	The aim of the Semantic-based Modeling Framework for Information Systems (SemFIS) is to allow for the semantic enrichment of conceptual modeling languages that represent a particular knowledge area. For this purpose it provides configurable meta models, mechanisms and algorithms that extend the modeling language of the knowledge area and enable the user to apply advanced semantic processing.Due to the flexible approach of SeMFIS, it can be easily adapted to arbitrary knowledge areas. The current implementation of the SeMFIS framework provides an exchange mechanism with the Protégé toolkit developed by Stanford University to integrate data from OWL ontologies in the modeling framework and make it available for the annotation of model elements.
51	SemCheck	SemCheck	University of Skovde	Semcheck is providing tools and methods for checking the integrity of models and meta models. SemCheck is based on the ConceptBase system, which allows to store models and meta models in a uniform representation, called Telos.

ID	Project	Akronym	Organisation	Description (self-declared)
52	SIMchronization	SIMchronization	University of Vienna	Efficient and responsive Supply Chains require a synchronous interaction of information and material flows. SIMchronization provides a domain-specific, graphical modeling method supporting documentation, analysis and improvement of Supply Chains.
53	sIOT	sIOT	University of Vienna	The sIoT project tackles a challenge in the digital transformation age, which is to bridge the gap between design concepts from design thinking and functional capabilities from Cyber Physical Systems.
54	Structured Entity Relationship Modeling Method (SERM)	SERM	University of Bamberg	In this project an ADOxx-based implementation of the Structured Entity Relationship Model (SERM) approach as developed by Prof. Dr. Elmar Sinz is provided. It includes functions for generating SQL code directly from SERM data models.
55	TEAM	TEAM	University of Pretoria University of Vienna	This project focuses on the realization of an Enterprise Architecture Management toolbox that enables the TOGAF based design and analysis of enterprise archtiectures.

ID	Project	Akronym	Organisation	Description (self-declared)
56	User Story Mapping	USM	École Polytechnique Fédérale de Lausanne	User story mapping (USM) for domain modelling is a method derived from software functionality definition domain that puts end-user and his perspective in focus and results in multi-purpose domain ontology design.

Table C.1: Descriptive Overview: OMiLAB Modelling Methods

In a next phase, the availability of development artefacts has been assessed. The criteria for assessment are a) whether an implementation result can be downloaded (in the form of a configuration library or deployable tool) and b) the characteristic of the project in focus (implementation of a modelling method, or general functionality for metamodelling).

ID	**Acronym**	**Version**	**Scope?**	**Comment**
1	4EM	2.2	YES	
2	BPRIM	2.0	YES	
3	ADVISOR	N/A	YES	
4	ArchiMate	N/A	NO	No implementation available, only usage provided in the projects context
5	BD-DS	0.1	YES	
6	Bee-Up	1.5	YES	
7	BEN		YES	
8	BPFM	0.3.1	YES	
9	NA	N/A	NO	No implementation available
10	CODEK	0.1	YES	
11	ComVantage	0.6	YES	
12	CUTiDe	2.1	YES	
13	DEMO	1.4.1	YES	
14	DIBA	1.0	YES	
15	DICE	2018	YES	
16	DICER	1.1	YES	
17	DMN	1.0	YES	
18	eduWeaver	N/A	NO	No implementation available
19	EVTM	8.2	YES	
20	ENTERKNOW	N/A	YES	
21	EKD	N/A	NO	No implementation available
22	EC	20131028	YES	
23	eGPM	2016.05.03	YES	
24	HCM-L	1.0	YES	
25	Hermxx	N/A	YES	
26	HORUS	0.1	YES	
27	iStar	N/A	YES	

ID	Acronym	Version	Scope?	Comment
28	JCS	N/A	YES	
29	KAMET	0.1	YES	
30	KWD	N/A	YES	
31	LeanPad	5.0	YES	
32	MEMO	1.1	YES	
33	MoSeS4eGov MDSDL	1.0	NO	Implementation language DE, does not transform
34	MoSeS4eGov Project	1.0	NO	Implementation language DE, does not transform
35	MMI-ML	0.2	YES	
36	MuVieMoT	1.0	YES	
37	OKM	N/A	NO	Collection of ideas, no implementation available
38	PetriNets	N/A	YES	
39	Petrixx	N/A	YES	
40	PGA	1.2	YES	
41	PRINTEPS	1.1	YES	
42	ProVis	1.1	YES	
43	PS3M	1.1	YES	
44	RUPERT	2.0	YES	
45	SAVE	3.0	YES	
46	S2M	1.5.2	YES	
47	SecTro	2.1.1	YES	
48	SDBD	1.0	YES	
49	SOM	3.0	YES	
50	SEMFIS	0.42	YES	
51	SemCheck	N/A	NO	Available as metamodelling technique, not a metamodel
52	SIMchronization	1.0	YES	
53	sIOT	N/A	YES	
54	SERM	1.0	YES	
55	TEAM	0.5	YES	
56	USM	1.0	YES	

Table C.2: Availability Assessment: OMiLAB Modelling Methods

The availability assessment resulted in 50 projects that did fulfil the requirement to provide an implementation.

Following this scoping phase, the implementation results have been downloaded and harmonised. During this step, the library files (in ALL format) were retrieved, the namespace (as input for the workplace organiser) has been established based on the project acronym, the artefacts where renamed to a common naming convention ($< ACRONYM > -v. < VERSIONNUMBER >$) and an initial transformation from the binary format to the textual representation is performed (as part of the import/export adaptor building block)

ID	**Acronym**	**Version**	**Artefact**	**Namespace**
1	4EM	2.2	4EM-v2.2.abl	4em
2	BPRIM	2.0	BPRIM-v2.0.abl	bprim
3	ADVISOR	N/A	ADIVSOR-vNA.abl	advisor
5	BD-DS	0.1	BD-DS-v0.1.abl	bdds
6	Bee-Up	1.5	Bee-Up-v1.5.abl	beeup
7	BEN		BEN-v1.0.abl	ben
8	BPFM	0.3.1	BPFM-v0.3.1.abl	bpfm
10	CODEK	0.1	CODEK-v0.1.abl	codek
11	ComVantage	0.6	ComVantage-v0.6.abl	comvantage
12	CUTiDe	2.1	CuTiDe-v2.1.abl	cutide
13	DEMO	1.4.1	DEMO-v1.41.abl	demo
14	DIBA	1.0	DIBA-v1.0.abl	diba
15	DICE	2018	DICE-v2018.abl	dice
16	DICER	1.1	DICER-v1.1.abl	dicer
17	DMN	1.0	DMN-v1.0.abl	dmn
19	EVTM	8.2	Testbed-v82.abl	testbed
20	ENTERKNOW	N/A	Enterprise Knowledge Development-vNA.abl	enterknow
22	EC	20131028	EvaluationChains-v2131028.abl	evaluation-chains
23	eGPM	2016.05.03	eGPM-v2016.05.03.abl	egpm
24	HCM-L	1.0	HCM-L-v1.0.abl	hcml
25	Hermxx	N/A	hermxx-vNA.abl	hermxx
26	HORUS	0.1	HORUS-v0.1.abl	horus
27	iStar	N/A	iStar-vNA.abl	istar
28	JCS	N/A	JCS-vNA.abl	jcs
29	KAMET	0.1	KAMET-v0.1.abl	kamet

ID	Acronym	Version	Artefact	Namespace
30	KWD	N/A	Knowledge Work Designer-vNA.abl	kwd
31	LeanPad	5.0	LearnPAD-v5.0.abl	learnpad
32	MEMO	1.1	MEMO4ADO-v1.1.abl	memo
35	MMI-ML	0.2	MMI-ML-v0.2.abl	mmi
36	MuVieMoT	1.0	MuVieMoT-v1.0.abl	muviemot
38	PetriNets	N/A	PetriNets-vNA.abl	petrinets
39	Petrixx	N/A	Petrixx-vNA.abl	petrixx
40	PGA	1.2	PGA-v1.2.abl	pga
41	PRINTEPS	1.1	PRINTEPS-v1.1.abl	printeps
42	ProVis	1.1	ProVis-v1.1.abl	provis
43	PS3M	1.1	PSSScenario-v1.1.abl	pss
44	RUPERT	2.0	RUPERT-v2.0.abl	rupert
45	SAVE	3.0	SAVE-v3.0.abl	save
46	S2M	1.5.2	Scene2Model-v1.5.2.abl	s2m
47	SecTro	2.1.1	SecTrop-v2.1.1.abl	sectrop
48	SDBD	1.0	SDBD-v1.0.abl	sdbd
49	SOM	3.0	SOM-v3.0.abl	som
50	SEMFIS	0.42	SEMFIS-v.0.42.abl	semfis
52	SIMchronization	1.0	SIMchronization-v1.0.abl	sim-chronization
53	sIOT	N/A	sIoT-vNA.abl	siot
54	SERM	1.0	SERM-v1.0.abl	serm
55	TEAM	0.5	TEAM-v0.5.abl	team
56	USM	1.0	User Story Mapping-v1.0.abl	user_story-_mapping

Table C.3: Artefacts and Namespaces: OMiLAB Modelling Methods

Analysing the code base for evaluation the following statistics could be derived. This table presents the counting of code lines, external files (that are provided as part of the implementation for additional functionality and embedded code) and their line count. This statistical analysis has been performed to establish an understanding of the scope of the evaluation and extend of metamodels implemented.

ID	ALL File	Code Lines	File count	Code Lines in Files
1	4EM-v2.2.all	6733	5	169
2	BPRIM-v2.0.all	4716	19	566
3	ADIVSOR-vNA.all	10521	34	928
5	BD-DS-v0.1.all	4143	2	65
6	Bee-Up-v1.5.all	68993	61	1311
8	BPFM-v0.3.1.all	49330	119	3716
10	CODEK-v0.1.all	45496	121	3825
11	ComVantage-v0.6.all	16981	64	1563
12	CuTiDe-v2.1.all	60493	60	1016
13	DEMO-v1.41.all	47442	127	4266
14	DIBA-v1.0.all	5616	34	548
15	DICE-v2018.all	33246	124	3831
16	DICER-v1.1.all	2682	6	82
17	DMN-v1.0.all	3270	1	1
19	Testbed-v82.all	22019	4	11
20	Enterprise Knowledge Development-vNA.all	4840	32	1263
22	EvaluationChains-v2131028.all	14937	9	10
23	eGPM-v2016.05.03.all	30181	353	9506
24	HCM-L-v1.0.all	5031	43	1670
25	hermxx-vNA.all	17613	32	1299
26	HORUS-v0.1.all	10716	28	334
27	iStar-vNA.all	14513	289	6880
28	JCS-vNA.all	1365	0	0
29	KAMET-v0.1.all	2775	5	292
30	Knowledge Work Designer-vNA.all	57255	146	4040
31	LearnPAD-v5.0.all	80001	178	5242
32	MEMO4ADO-v1.1.all	53111	493	31485
35	MMI-ML-v0.2.all	2953	9	101
36	MuVieMoT-v1.0.all	4616	14	444
38	PetriNets-vNA.all	9050	80	4940
39	Petrixx-vNA.all	2389	11	692
40	PGA-v1.2.all	2920	15	595
41	PRINTEPS-v1.1.all	8133	19	523

ID	**ALL File**	**Code Lines**	**File count**	**Code Lines in Files**
42	ProVis-v1.1.all	3474	15	915
43	PSSScenario-v1.1.all	10404	0	0
44	RUPERT-v2.0.all	39588	59	3625
45	SAVE-v3.0.all	7329	29	1698
46	Scene2Model-v1.5.2.all	55938	299	4840
47	SDBD-v1.0.all	58327	314	6475
48	SecTrop-v2.1.1.all	7916	44	1903
49	SEMFIS-v.0.42.all	65401	60	3372
50	SERM-v1.0.all	1998	5	214
52	SIMchronization-v1.0.all	5217	41	950
53	sIoT-vNA.all	48527	114	1484
54	SOM-v3.0.all	40102	397	11974
55	TEAM-v0.5.all	8596	6	401
56	User Story Mapping-v1.0.all	7498	7	473
	Sum	**1064395**	**3927**	**129538**

Table C.4: Code Statistics: ALL Repository for Evaluation