



universität
wien

MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

„A Domain Integration Framework for
Business Process Modeling“

verfasst von / submitted by

Alexandra Birkmaier, BSc

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of
Master of Science (MSc)

Wien, 13. Mai 2019 / Vienna, 13th of May 2019

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

A 066 915

Studienrichtung lt. Studienblatt /
degree programme as it appears on
the student record sheet:

Masterstudium Betriebswirtschaft
Master's degree program Business Administration

Betreut von / Supervisor:

o. Univ.-Prof. Dr. Dimitris Karagiannis

Table of Contents

| | |
|---|------|
| LIST OF ABBREVIATIONS..... | III |
| LIST OF FIGURES | V |
| LIST OF TABLES | VIII |
| 1 INTRODUCTION | 1 |
| 1.1 PROBLEM STATEMENT | 1 |
| 1.2 RESEARCH OBJECTIVES | 3 |
| 1.3 RESEARCH APPROACH | 4 |
| 1.4 OUTLINE AND FOCUS OF THE THESIS | 8 |
| 2 STATE OF THE ART | 10 |
| 2.1 BUSINESS PROCESS MODELING | 10 |
| 2.1.1 <i>Concepts</i> | 10 |
| 2.1.2 <i>Requirements</i> | 13 |
| 2.2 DOMAIN-SPECIFIC MODELING | 15 |
| 2.2.1 <i>Degree of Domain-specificity</i> | 16 |
| 2.2.2 <i>Frameworks</i> | 16 |
| 2.3 MODEL AND META-MODEL | 18 |
| 2.4 MODELING METHOD BUILDING BLOCKS | 20 |
| 2.4.1 <i>Modeling Language</i> | 21 |
| 2.4.2 <i>Modeling Procedure</i> | 22 |
| 2.4.3 <i>Modeling Mechanisms & Algorithms</i> | 23 |
| 2.5 INTERIM CONCLUSION | 24 |
| 3 ANALYSIS OF DSML | 27 |
| 3.1 DOMAIN SPECIFICATION | 28 |
| 3.2 QUALITY EVALUATION | 33 |
| 3.3 DS COMPONENT IDENTIFICATION | 36 |
| 4 RESULTS – THE DOMAIN INTEGRATION FRAMEWORK (DIF) | 39 |
| 4.1 DOMAIN-SPECIFIC DESIGN | 41 |
| 4.1.1 <i>Domain Framework (DF)</i> | 41 |
| 4.1.2 <i>Domain-specific User Stories</i> | 46 |
| 4.1.3 <i>Meta-models of Modeling Method Building Blocks</i> | 49 |
| 4.2 DOMAIN-SPECIFIC IMPLEMENTATION | 53 |
| 4.2.1 <i>Linguistic Matching Heuristic</i> | 54 |

| | | |
|-------|---|-------|
| 4.2.2 | <i>Domain-specific Modeling Tool</i> | 55 |
| 4.3 | DOMAIN-SPECIFIC MODELING | 55 |
| 4.3.1 | <i>Quality Criteria Evaluation</i> | 56 |
| 4.3.2 | <i>Adaptation Log</i> | 59 |
| 5 | PROOF OF CONCEPT – AUTOMOTIVE ASSEMBLY LINE CASE STUDY .. | 60 |
| 5.1 | DOMAIN-SPECIFIC DESIGN | 61 |
| 5.1.1 | <i>Domain Specification</i> | 61 |
| 5.1.2 | <i>Domain-specific Requirements</i> | 69 |
| 5.1.3 | <i>Domain-specific Modeling Method</i> | 74 |
| 5.2 | DOMAIN-SPECIFIC IMPLEMENTATION | 83 |
| 5.2.1 | <i>Linguistic Matching Heuristic</i> | 83 |
| 5.2.2 | <i>Domain-specific Modeling Tool – Automotive Assembly Line DSL (AAL-DSL)</i> ... | 86 |
| 5.3 | DOMAIN-SPECIFIC MODELING | 90 |
| 5.3.1 | <i>Models of the AAL-DSL</i> | 90 |
| 5.3.2 | <i>Quality Criteria and New Requirements</i> | 102 |
| 5.4 | LIMITATIONS OF THE PROOF OF CONCEPT | 106 |
| 6 | CONCLUSION AND FUTURE OUTLOOK | 107 |
| | LITERATURE | IX |
| | APPENDIX A | XVII |
| | APPENDIX B | XXII |
| | APPENDIX C | XXXVI |
| | ABSTRACT | XL |
| | ZUSAMMENFASSUNG | XLI |

List of Abbreviations

AAL-DSL – Automotive assembly line domain-specific language

AMME – Agile Modeling Method Engineering

APQC - American Productivity and Quality Center

BP – Business process

BPM(L) – Business process modeling (language)

DF – Domain Framework

DIF – Domain Integration Framework

DS(L) – Domain-specific (language)

DSBPM(L) – Domain-specific business process modeling (language)

DSM(L) – Domain-specific modeling (language)

DSMM – Domain-specific modeling method

DSRM – Design Science Research methodology by [88]

ER diagrams – Entity Relationship diagrams

FCML – Fundamental conceptual modeling languages by [54]

GP – General-purpose

GPM(L) – General-purpose modeling (language)

IS – Information systems

JIS – Just-in-sequence

JIT – Just-in-time

KPI – Key performance indicator

ML – Modeling language

MM – Modeling method

MP – Modeling procedure

OMiLAB – Open Models Initiative LABoratory

PCF® - Process Classification Framework by [7]

RE – Requirements engineering

UML – Unified Modeling Language

List of Figures

| | |
|--|----|
| FIGURE 1 DEGREE OF SOFTWARE CUSTOMIZATION | 2 |
| FIGURE 2 RESEARCH APPROACH | 5 |
| FIGURE 3 TAXONOMY OF LITERATURE REVIEWS..... | 6 |
| FIGURE 4 SEARCH-PROCESS DOCUMENTATION | 7 |
| FIGURE 5 INFORMATION SYSTEMS RESEARCH FRAMEWORK..... | 8 |
| FIGURE 6 ELEMENTS OF A BUSINESS PROCESS..... | 12 |
| FIGURE 7 ROLES IN DOMAIN-SPECIFIC BPM | 15 |
| FIGURE 8 LANGUAGE-BASED META-MODEL CONCEPT | 18 |
| FIGURE 9 MODELING METHOD BUILDING BLOCKS | 20 |
| FIGURE 10 GENERIC META-MODEL OF MODELING LANGUAGE | 22 |
| FIGURE 11 GENERIC META-MODEL OF MODELING PROCEDURE | 23 |
| FIGURE 12 GENERIC META-MODEL OF MECHANISMS AND ALGORITHMS | 24 |
| FIGURE 13 LITERATURE-COLLECTION AND -ANALYSIS PROCEDURE | 27 |
| FIGURE 14 EXAMPLE OF A META-MODEL STRUCTURED ACCORDING TO ITS MODEL TYPES | 36 |
| FIGURE 15 THE DOMAIN INTEGRATION FRAMEWORK (DIF) | 39 |
| FIGURE 16 THE DOMAIN FRAMEWORK (DF) | 41 |
| FIGURE 17 CORRELATIONS BETWEEN THE THREE DIMENSIONS OF THE DF | 42 |
| FIGURE 18 OPTIONS OF MODELING METHOD DESIGN..... | 50 |
| FIGURE 19 DIF MODELING LANGUAGE | 51 |
| FIGURE 20 DIF MODELING PROCEDURE | 52 |
| FIGURE 21 DIF MODELING MECHANISMS & ALGORITHMS | 53 |
| FIGURE 22 MATCHING POSSIBILITIES | 54 |
| FIGURE 23 METRICS FOR COMPLEXITY ASSESSMENT | 57 |

| | |
|--|----|
| FIGURE 24 PORTFOLIO-DIAGRAM FOR THE EVALUATION OF MODELING LANGUAGE | |
| CRITERIA | 58 |
| FIGURE 25 LINE ASSEMBLY | 60 |
| FIGURE 26 DOMAIN FRAMEWORK (DF) OF THE AAL-DSL..... | 62 |
| FIGURE 27 DF INDUSTRY SPECIFICATION OF THE AAL-DSL..... | 63 |
| FIGURE 28 DF MCS LEVEL SPECIFICATION OF THE AAL-DSL..... | 63 |
| FIGURE 29 THE PROCESS ARCHITECTURE OF THE AAL-DSL..... | 64 |
| FIGURE 30 DF GRANULARITY SPECIFICATION OF THE AAL-DSL..... | 65 |
| FIGURE 31 INTEGRATED GRANULARITY-LEVELS OF THE AAL-DSL..... | 65 |
| FIGURE 32 PROCESS LANDSCAPE META-MODEL OF THE AAL-DSL..... | 75 |
| FIGURE 33 PROCESS MODEL META-MODEL OF THE AAL-DSL | 77 |
| FIGURE 34 PRODUCT STRUCTURE META-MODEL OF THE AAL-DSL | 78 |
| FIGURE 35 WORKING ENVIRONMENT META-MODEL OF THE AAL-DSL..... | 78 |
| FIGURE 36 COMPLETE META-MODEL OF THE AAL-DSL..... | 79 |
| FIGURE 37 MODELING PROCEDURE META-MODEL OF THE AAL-DSL | 81 |
| FIGURE 38 MODELING MECHANISMS AND ALGORITHMS META-MODEL OF THE AAL-DSL | 82 |
| FIGURE 39 LINGUISTIC MATCHING HEURISTIC – CREATE | 85 |
| FIGURE 40 LINGUISTIC MATCHING HEURISTIC – EXTEND | 86 |
| FIGURE 41 LINGUISTIC MATCHING HEURISTIC – REDUCE..... | 86 |
| FIGURE 42 MODEL OVERVIEW | 91 |
| FIGURE 43 PROCESS LANDSCAPE L0 | 92 |
| FIGURE 44 PROCESS MODEL L1 | 93 |
| FIGURE 45 PROCESS MODEL L2..... | 94 |
| FIGURE 46 PROCESS MODEL L3..... | 96 |
| FIGURE 47 WORKING ENVIRONMENT MODEL..... | 97 |
| FIGURE 48 PRODUCT STRUCTURE MODEL | 98 |

| | |
|---|-----|
| FIGURE 49 DEFINITION OF TASK PARAMETERS | 99 |
| FIGURE 50 QUERY RESULTS | 100 |
| FIGURE 51 ANALYTIC EVALUATION RESULTS | 100 |
| FIGURE 52 PATH ANALYSIS RESULTS | 101 |
| FIGURE 53 SIMULATION RESULTS PATH ANALYSIS | 101 |
| FIGURE 54 CAPACITY ANALYSIS RESULTS | 102 |
| FIGURE 55 QUERIES FOR COMPLEXITY ASSESSMENT | 103 |
| FIGURE 56 COMPLEXITY ASSESSMENT OF THE AAL-DSL | 104 |
| FIGURE 57 QUALITY ASSESSMENT PORTFOLIO-DIAGRAM OF THE AAL-DSL | 105 |

List of Tables

| | |
|--|-----|
| TABLE 1 GENERAL-PURPOSE BPM APPROACHES | 12 |
| TABLE 2 REVELATIONS FROM THE LITERATURE REVIEW FOR THE FURTHER COURSE OF THE THESIS | 24 |
| TABLE 3 DOMAIN CONTEXT OF DSML | 28 |
| TABLE 4 DOMAIN ANALYSIS OF DSML | 30 |
| TABLE 5 QUALITY EVALUATION OF DSML | 34 |
| TABLE 6 DOMAIN-SPECIFIC COMPONENT IDENTIFICATION | 37 |
| TABLE 7 USER STORIES FOR DOMAIN-SPECIFIC DESIGN | 48 |
| TABLE 8 QUALITY CRITERIA COLLECTED | 56 |
| TABLE 9 SUMMARY OF MODEL TYPES FOR THE AAL-DSL | 66 |
| TABLE 10 ELEMENTS IDENTIFIED BY THE DF | 68 |
| TABLE 11 KEY FOR REQUIREMENTS ASSESSMENT | 70 |
| TABLE 12 COLLECTED, ASSESSED, AND CLUSTERED REQUIREMENTS | 70 |
| TABLE 13 ELEMENTS IDENTIFIED BY THE USER STORIES | 73 |
| TABLE 14 MATCHING HEURISTIC APPLIED | 83 |
| TABLE 15 CONCEPTS OF THE AAL-DSL | 87 |
| TABLE 16 COMPLEXITY ASSESSMENT OF THE AAL-DSL | 103 |

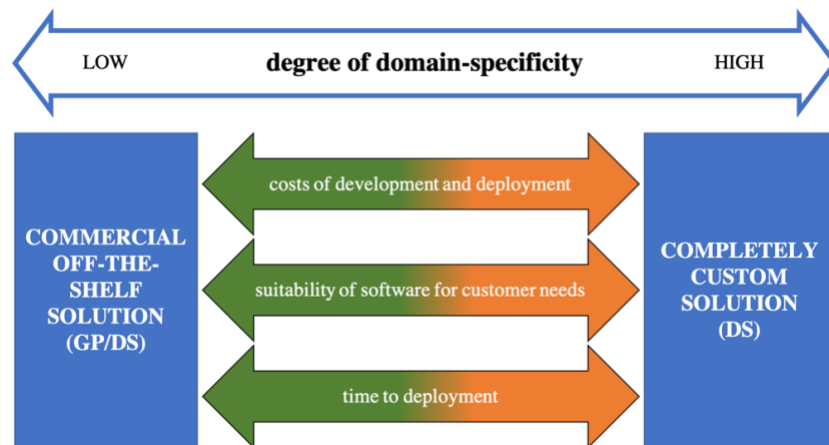
1 Introduction

Business process modeling (BPM)¹ can be seen as the major discipline within business process management, as it is used to graphically describe the as-is business processes of a company as well as the desired to-be business processes after a BP optimization initiative [cf. 32]. According to [75], the business process management market is showing rapid growth with an estimated compound annual growth rate of 14,2 % between 2016 and 2021. Companies spend a considerable amount of resources to decide, which BPM language is the most suitable for their specific company needs. Some argue for the need of general-purpose (GP) BPM languages to save monetary resources by avoiding implementation efforts. Others emphasize the necessity to have domain-specific (DS) BPM languages, which are tailored to the requirements of a specific application area and are more likely to be accepted by the users (cf. [57], [64]). Within an insurance company, for example, the domain of modeling insurance processes requires concepts, which represent risks or insured objects like a car or house. Within business process modeling, domain-specific modeling has experienced a growing research interest, as it helps to raise the level of abstraction and uses the vocabulary necessary to address the respective domain (cf. [60], [30]).

1.1 Problem Statement

General-purpose modeling (GPM) languages often fail to depict domain-specific concepts, rules, and functionalities, which are needed to model the respective domain adequately [cf. 60]. GP and DS software solutions are freely available or offered by IT companies to be sold to the general market or a specific market niche. Also, companies themselves produce their own in-house solutions in order to fit their specific domain needs. Costs for customized IT-solutions are high compared to solutions available on the market [cf. 121]. There is a trade-off between solutions readily available on the market and solutions tailored to the specific enterprise and application area needs (see Figure 1).

¹ Within this thesis, abbreviations are written out the first time of their appearance. The short form is used in the further course of the text.



*Figure 1 Degree of software customization
- own representation based on [120]*

Figure 1 shows the trade-offs between a low degree and a high degree of domain-specificity, adapted from the analysis of software customization by [120]. Completely customized, also called domain-specific, solutions tend to be more costly and their implementation is more time-intensive. Yet, they are most suitable for the specific company needs. This implies a need for solutions, which are domain-specific to the highest-possible extent while making use of existing implementations to reduce time and costs.

If companies opt for choosing commercial DS language solutions, they often need to make adaptations to the actual state of their processes in order to “make them fit” to the language needs. This fact is taken into account by [76], who names *organizational fit* as a main risk in the enterprise wide implementation of ERP projects. A specific risk factor of the organizational fit is the *failure to redesign business processes*. Mitigating this risk by adding regular re-evaluation and control mechanisms helps to keep the fit between the implemented IT solution and the actual domain needs. Therefore, an approach to develop and manage customized solutions, which accounts for regular re-evaluation mechanisms, is needed.

As of now, there is no satisfactory definition of what a domain is and how domains can be differentiated from each other or categorized [cf. 54]. Throughout the literature about modeling methods and languages, the terms domain-general or general-purpose and domain-specific are recognized and used to distinguish between different degrees of applicability of a modeling method in a specific application area. To name some examples, the Unified Modeling Language (UML) is used mainly in the domain of software engineering and Entity Relationship (ER) diagrams in data modeling. When a language for domain-specific

modeling is designed, two kinds of considerations are made by the method engineer [cf. 54]. First, existing languages and notations as well as experiences made with them are considered. Second, modeling requirements are evaluated for the implementation of necessary extensions or specifications. This situation leads to the conclusion that there exists a need for a toolkit for modeling method development, which enables a flexible integration of domain-specific requirements. Examples of such toolkits are the meta-modeling platforms ADOxx, MetaEdit+, and Meta Object Facility (MOF), among others [cf. 56].

Moreover, there is no clear distinction between GP modeling (GPM) and DS modeling (DSM). There exist different characteristics and criteria that enable to classify the quality of modeling methods and languages. In the relevant literature, more inwards looking quality criteria such as consistency, integrity, and performance are considered instead of outwards quality criteria like usefulness, comprehensibility or end-user acceptance, and usability [cf. 52]. By this predominant approach, requirements of different stakeholder groups using the modeling method in the respective domain are often neglected. Companies moving towards higher abstraction levels face considerable advantages. The increased business value described by [60] consists of a higher productivity, quality, way to leverage expertise, and improved economics.

As a conclusion of the above statements, DSM can be seen as a suitable approach to model the processes of a company in a way that is tailored to the specific company needs. In order to obtain a valid graphical, domain-specific modeling tool no solution is known, which combines a procedure for modeling method engineering with an approach to design, implement, model, and re-evaluate the domain-specific solution. Most existing frameworks either focus on domain analysis [cf. 96] or on DS solutions with the goal to generate code [cf. 60]. The research objectives followed within this thesis arise as a consequence of the previous sections and are described within the next chapter.

1.2 Research Objectives

The overall objective of this thesis is to provide a guideline to domain-specific modeling method design and implementation in form of a life-cycle model. The Domain Integration Framework (DIF) developed within this thesis serves as a procedure model, which provides direction on approaching domain-specific BPM initiatives and recommends useful tools for each phase. It provides assistance when classifying domains and helps to consider all

relevant requirements, which are influenced by a certain application. Moreover, it includes chosen quality criteria, which describe the applicability of the designed modeling method in the specific domain and also accounts for regular re-evaluation in order to adapt the language to changing requirements.

The research questions answered within this thesis are summarized as follows:

1. Identify **components**, which make a language domain-specific, through the analysis of existing domain-specific BPM languages
2. Derive a suitable **categorization scheme for domains** in the light of this thesis, based on literature research
3. Construct a **Domain Integration Framework (DIF)** for systematic creation and evaluation of modeling methods
4. Provide **tools** for the design, implementation, and modeling phase of the domain-specific modeling method
5. Prove the **validity** of the DIF in the light of this thesis by conducting a case study on automotive assembly line modeling

In order to provide a proof-of-concept for the DIF developed within this thesis, it is applied on the domain of automotive assembly line modeling. The goal is to test the DIF in one domain-context and prove its validity in the light of this thesis. In case any inconsistencies or unconsidered aspects are identified, the DIF is re-evaluated. This iterative process follows the Design Science Research methodology by [47] and [88]. The applicability of the DIF is regarded as proven within this thesis, if the resulting modeling method fulfills the domain-specific requirements.

1.3 Research Approach

In order to meet the solution objectives described above while taking into account the fact that many domain-specific business processes are created inhouse and therefore not necessarily documented in literature, the research approach has to consider both situations. Figure 2 shows the exploratory research approach followed within this thesis. Exploratory research aims at combining literature review as well as observation and evaluation of the

real modeling environment in order to derive findings [cf. 112]. By evaluating real-life examples, relevant questions are identified. Important parameters are the practicality and usefulness of the solution artifact, which is why specific situations are regarded in order to generalize them in the DIF. The exploratory research methodology is useful, when the objective is to “gain familiarity with a phenomenon or to achieve new insights into it” [65].

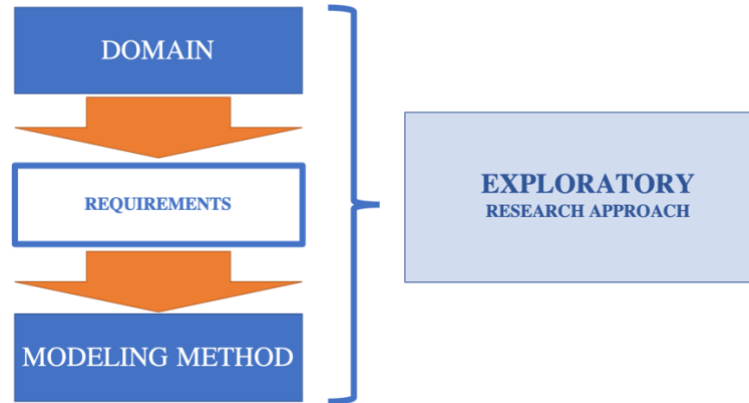


Figure 2 Research approach
– own representation

Figure 2 shows the research approach in a schematic way. As the aim of this thesis is to develop an approach for designing domain-specific modeling methods, the first step is to investigate the term domain. This includes a formal definition and classification of the term as well as the evaluation of components, which make a DS language domain-specific. Requirements, which are distinctive for a certain domain and important for the modeling method to be developed, are investigated and specified. The modeling method as the output of the DIF must have a domain-fit in all its building blocks in order to be applicable and useful for the respective domain.

In order to allow for domain-specific implementation, a meta-modeling platform should rather support a complete modeling method than a modeling language only. On the one hand it should include model-driven functionality based on the requirements and on the other hand, related to the modeling goals and needed functionality, provide guidelines and constraints for modeling scenarios [cf. 54]. ADOxx [16] is used within this thesis as the technology for implementing the DS modeling method.

For the purpose of this thesis, a suitable research method is a literature review to find and analyze existing domain-specific modeling languages (DSML). Especially for the literature

review in the IS domain, the approach by [118] is followed, where a five-phase framework is proposed.

Phase I: To define the scope of the literature review, the taxonomy of Cooper [24] is used, which is comprised of six constituent characteristics (focus, goals, organization, perspective, audience, coverage). Figure 3 shows the categories of these six characteristics used within this thesis marked in blue.

| Characteristic | | Categories | | | |
|----------------|--------------|------------------------|--------------------------|----------------|-----------------|
| (1) | Focus | research outcomes | research methods | theories | applications |
| (2) | Goal | integration | criticism | central issues | |
| (3) | Organization | historical | conceptual | methodological | |
| (4) | Perspective | neutral representation | espousal of position | | |
| (5) | Audience | specialized scholars | general scholars | practitioners | general public |
| (6) | Coverage | exhaustive | exhaustive and selective | representative | central/pivotal |

Figure 3 Taxonomy of literature reviews
– own representation based on [24]

Phase II: This step aims to identify key issues by concept mapping and provide working definitions to key terms [cf. 103, p. 36], which is done within the state of the art analysis in chapter 2.

Phase III: The general literature search process involves database, keyword, backward, and forward search, and an ongoing evaluation of sources. A focus is set on articles published in scholarly journals or proceedings of renowned conferences. The search-process documentation can be seen in Figure 4.

Phase IV: The literature is analyzed and synthesized in a concept matrix. Within this thesis, this is done in chapter 3 – Analysis of DSM languages.

Phase V: The results of the DSM language analysis reveal language-components, which contribute to a high extent to domain-specificity.

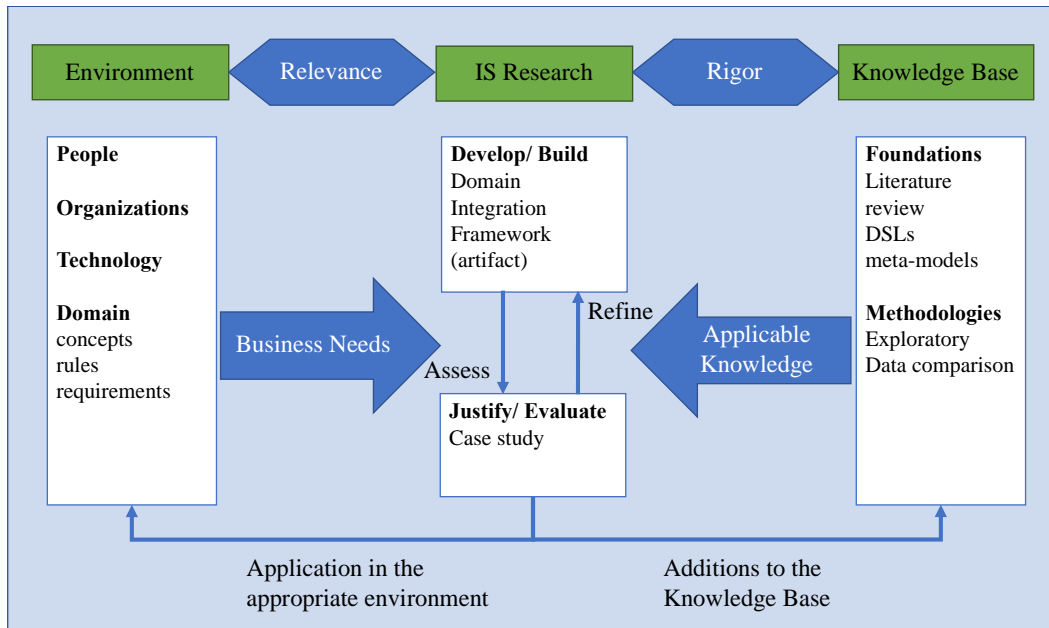
| | | | |
|--|---|---|--|
| Search outlets | Considered are articles from following research databases: Scopus, EBSCOhost, ScienceDirect, IEEE A focus is set on newer publications, starting from year 1990. | | |
| Search terms | BPM related | DSM related | Meta-model related |
| | process model, model-driven engineering, model-driven reengineering | Domain model, domain*specific, domain engineering, requirements engineering, mini language* | dynamic, meta, high-level, abstraction, conceptual |
| Search string | Any of the business process modeling related search terms in abstract AND any of the domain-specific modeling related search terms in abstract AND any of the meta-model related search terms in abstract | | |
| The asterisk (*) can be substituted with several letters | | | |

Figure 4 Search-process documentation

– own representation based on [24]

The Design Science Research methodology (DSRM) after [88] is used widely in information systems research. Design is the “act of creating an explicitly applicable solution to a problem” [88]. Design Science Research differs from other paradigms like theory building and testing and interpretative research, in that the prototype designed can be “any designed object with an embedded solution to an understood research problem” [88, p. 6].

As the above described exploratory research approach describes the way, in which knowledge is gained throughout this thesis, the DSRM provides a whole framework with the goal of developing a prototype. The exploratory research approach can be seen to stand above the DSRM framework on a higher level. Within this thesis it is used as a complementary approach for data collection and research design, which is manifested within the DSRM framework. Figure 5 shows the integrated DSRM framework followed within this thesis based on [46].



*Figure 5 Information systems research framework
– own representation based on [46]*

The information systems (IS) research framework provides an integrated view of the environment and the knowledge base. The two main components of the IS research block are develop/ build and justify/ evaluate. Within this thesis, the developed DIF represents the artifact. The step of evaluation is done by the application of the DIF on a real instance, which is achieved by conducting a case study.

1.4 Outline and Focus of the Thesis

In the previous chapters, the concept of domain-specific business process modeling (DSBPM) was introduced to the reader of this thesis. For one, the importance and potential of this notion within business process management was explained. After that, the research objectives pursued within this paper were defined and explained in the context of DSBPM. In order to answer the research objectives in a scientific and systematic way, the research approaches *Design Science Research* and *literature review in information systems* were introduced.

The succeeding chapters within this thesis are structured as follows. In chapter 2, the state of the art regarding important concepts related to DSBPM is evaluated. Here, the focus lies on more recent developments and the interpretation of their relevance in the light of this thesis. Chapter 3 is dedicated to the analysis of several collected DSBPM languages. Here,

the insights gained throughout the state of the art are used to define focus areas, according to which the DSBPM languages are analyzed. The goal of the analysis process is the extraction of concepts, which significantly affect the domain-specificity of BPM languages. The knowledge gained throughout the analysis is then abstracted in chapter 4, the results section, into the Domain Integration Framework (DIF) as the artifact of this scientific work. Following the Design Science Research methodology, the DIF is tested by conducting a case study in chapter 5. The thesis concludes with chapter 6 by final remarks about the research and implementation project, its opportunities and limitations in form of a SWOT analysis, as well as an outlook for future research directions.

The focus of the work presented within this scientific paper lies on graphical modeling languages as opposed to textual ones. Graphical modeling languages consist of a graphical representation of their syntax [cf. 44]. Here, concepts (or classes) are represented by symbols, and relationships between those symbols are represented by connecting lines. Examples of graphical modeling languages are UML [105] or BPMN [86]. Textual modeling languages, on the other hand, are computer-interpretable and are composed of keywords and parameters [cf. 105]. Examples of textual modeling languages are OCL [85] or PlantUML [101]. The context of modeling within this thesis is exclusively seen in the graphical representation of business processes as language-oriented constructs. Therefore, it shall not be confused with the concepts of textual, mathematical, or statistical modeling. The focus of this thesis lies on the graphical value of DSBPM, not its translation into usable code.

2 State of the Art

The goal of the following chapter is to provide the reader with an understanding of the concepts and related work relevant in the context of this thesis. In the beginning, more general concepts related to the research area of domain-specific business process modeling (DSBPM) are shown and condensed throughout the chapter into more specific topics. Advances in business process modeling (BPM) are shown in chapter 2.1 with a special focus on newer developments of BPM concepts as well as requirements engineering. Approaches towards domain-specific modeling (DSM) as well as categorization schemes of domains are investigated in chapter 2.2. The concept of meta-modeling (chapter 2.3) is crucial when it comes to describe a modeling language on a higher level by abstracting away unnecessary details. Chapter 2.4 introduces the concept of a modeling method and shows existing approaches to design it.

2.1 Business Process Modeling

Business process modeling (BPM) as a research discipline offers opportunities in a variety of contexts, e.g. business management, industrial engineering, and information technology [cf. 32]. Various authors stress the importance of business process modeling within the company to be able to depict activities, roles, and process-dependencies, among others (cf. [102], [102], [10]). Rosemann [102], for example, suggests that a company should establish a business process management center of excellence and provides a portfolio of services to be offered. Within this chapter on BPM, the first part (chapter 2.1.1) deals with the concepts of BPM and sets a special focus on newer developments and research areas. Chapter 2.1.2 is dedicated to the fields specialized in the identification and categorization of requirements, which are a preliminary for high-quality BPs.

2.1.1 Concepts

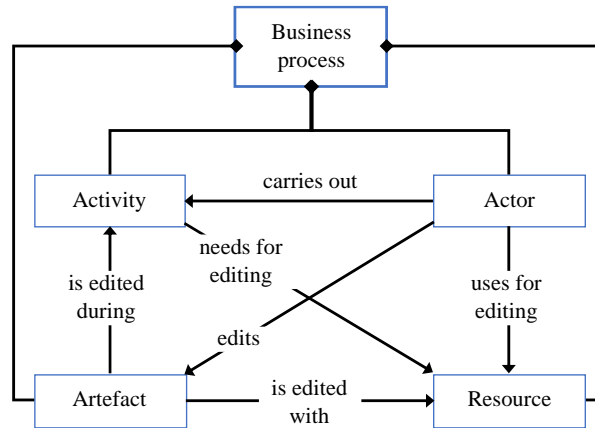
Throughout the past years, research was mainly focused on two fields. The first field being the design of methods for BP modeling (cf. [110], [72]) and the second field engaging in methods for BP reengineering (cf. [26], [42], [125], [19]). Attributable to the growing complexity in today's corporate environment, research towards more flexible and situation-specific concepts is claimed (cf. [19], [10]).

In the context of more flexible and agile business process modeling approaches, Becker [10] argues to move from a functional orientation towards a process orientation. The author states that the previous optimization of single functional areas or departments has only brought marginal improvements. Instead, companies need to focus on cross-departmental processes, which has started around the 1980s. In this context, research on situational models (cf. [122], [107], [98]) and conceptual models (cf. [57], [25]) has emerged.

When it comes to the structuring and categorization of BPs, the emergence of different viewpoints within literature can be observed. One of the originally used categorization schemes of BPs is the differentiation between management and support processes by Michael Porter [93]. Other authors add the instance of management processes (cf. [32], [119]). E.g. Schmelzer et al. [108] add additional viewpoints in the form of a client perspective and a perspective on different organizational functions. Also, a classification according to process structure (ad-hoc processes, weakly-structured processes, structured processes) is proposed by [108]. Melão and Pidd [78] propose a conceptual framework, in which different business process views are organized according to four categories (BPs as deterministic machines, interacting feedback loops, dynamic complex systems, social constructs). Nastansky et al. [81] classify business processes according to their structure into ad-hoc process, open team-process, integrated team-process, integrated cooperative activity, ad-hoc exceptions, and well-structured processes. These classifications vary regarding their degree of flexibility versus structure.

Due to the surging amount of BP languages and methods, evaluation methods have emerged. One category builds BPM maturity models (cf. [27], [72], [41]). DeToro and McCabe [29] propose five process condition ratings and five improvement paths, respectively. Hammer [41], for example, proposes a Process and Enterprise Maturity Model. In this context, certain quality criteria have been investigated within literature. Quality criteria for BP metrics are investigated e.g. by [127], [32], and [52]. A framework, which enables the evaluation of conceptual BPM languages is proposed by [72]. The authors use the meta-models of seven BPM languages and compare them to their own generic meta-model based on the four BP perspectives organizational, functional, informational, and behavioral plus the additional perspective BP context [cf. 25].

Curtis et al. [25] extract the most commonly used BP constructs as being agent, role, and artifact. Junginger [51] defines BPs by four elements and their interrelations, which can be seen in Figure 6.



*Figure 6 Elements of a business process
- own representation based on [51]*

Lu and Sadiq [74] conduct a comparative analysis of the two predominant business process modeling approaches, namely graph-based approaches and rule-based approaches. According to the authors, most graph-based modeling approaches originated in Petri Net theory and its extensions. Different graph-based general-purpose modeling approaches are shown in Table 1.

*Table 1 General-purpose BPM approaches
– own representation*

| GP BPML | Elements | Model types |
|--|---|--|
| Entity-relationship (ER) diagrams [21] | Entity, action, attribute, cardinality, connecting line | Data model |
| Unified Modeling Language (UML) [105] | Element, relationship, directed relationship, named element, redefinable element, type, feature, instance and instance specification, comment | Class diagram, package diagram, object diagram, component diagram, composite structure diagram, deployment diagram, activity diagram, sequence diagram, use case diagram, state diagram, communication diagram, interaction overview diagram, timing diagram |

| | | |
|---|--|---|
| Business Process Model & Notation (BPMN) [86] | Flow object, connecting object, swim lane, artifact, event, activity, gateway | Process model |
| Event-driven Process Chain (EPC) [59] | Function, event, gateways, control flow, process navigator, organizational unit, information object, information flow, link between organizational units | Data model, process model, IT system model, organizational model, product model |
| Petri Nets [126] | Places, transitions, arcs | Process model |
| FlowMake [106] | Task, coordinator, and transition | Process model |
| EXPRESS [50] | Datatype, entity-attribute, supertypes and subtypes, algorithmic constraints | Data model |

2.1.2 Requirements

In order to allow for a working as well as accepted BPM approach, the right requirements need to be fulfilled. In the field of requirements engineering (RE), different approaches to identify, evaluate, and implement requirements are investigated. [84], [90], [69], for example, provide an overview of RE in the field of information systems.

Within literature, there exist different approaches to categorize requirements. Pohl and Rupp [91] categorize requirements into functional and quality requirements, and into boundary conditions. Nuseibeh and Easterbrook [84] differentiate between five core RE-activities, namely eliciting requirements, modeling and analyzing requirements, communicating requirements, agreeing requirements, and evolving requirements. Eliciting requirements as the relevant activity in the context of this thesis is composed of identifying system boundaries, stakeholders, goals, and tasks by use-cases and scenarios [cf. 84]. It should be noted, that the authors explicitly state the evolution of requirements within their RE core-activities, which constitutes an integral part of the artifact developed within this thesis, too. Buchmann and Karagiannis [20] emphasize the evolutionary nature of requirements by categorizing modeling requirements into those originating from design-time needs (directly) and those originating from run-time needs (indirectly) (cf. [52], [20]).

Visic et al. [117] distinguish between primary, secondary, tertiary, and quaternary requirements. The *primary requirements* (language requirements) are derived from the application domain of the language. As already discussed in chapter 2.1.1, the four basic constructs of class, relation class, attribute, and model-type build the foundation of the syntax and semantics of the ML. In order to allow for needed extensions, extensibility is considered as an additional key requirement. Another requirement is derived from the need for a suitable notation, which has the ability to represent the specific application domain with a set of expressive graphical symbols. Last but not least, further requirements are derived from the modeling algorithms needed, for instance analysis and simulation functionalities. *Secondary requirements* are derived from the respective meta-modeling platform the solution shall be implemented on. Here, the functional as well as non-functional requirements of meta-modeling platforms are considered. *Tertiary requirements* focus on commonly accepted principles and best practices for the design of DSLs. Following desirable features, adapted from [77], are proposed: user-expectation conformity, readable and consistent syntax, small and orthogonal set of features, and error diagnosis. The *quaternary requirements* refer to the possible evolution of the ML in the future due to emerging technologies. On the one hand, the future development of meta-modeling platforms has to be considered, and on the other hand, changes within the application domain. Regarding the second point, minimum requirements are an extensible abstract syntax, concrete syntax, and semantics, as well as an extensible execution engine. *Non-functional* requirements are extensibility, interoperability, and scalability².

Different approaches to identify requirements can be distinguished. Bortz and Döring [17] name surveys as the most common method to collect data in social sciences. These can be either oral by conducting interviews or in a written manner, e.g. by questionnaires. When it comes to agile software development, e.g. in Scrum [111] and DSDM [1], requirements are depicted by tasks in the product backlog, which are prioritized and realized throughout the sprints. Those task items can be categorized into features, bugs, technical work, and knowledge acquisition. The approach of collecting features is by stakeholder-centric user

² For a detailed description of non-functional requirements the reader is referred to [117] N. Visic, H.-G. Fill, R. A. Buchmann, and D. Karagiannis, "A domain-specific language for modeling method definition: From requirements to grammar," in *Research Challenges in Information Science (RCIS)*, 2015 IEEE 9th International Conference on, 2015, pp. 286-297: IEEE.

stories. User stories can be defined by a given sequence of words with blank spaces in the form: “As a (role) I want (something) so that (benefit)” [cf. 22].

2.2 Domain-specific Modeling

A challenge within domain-specific modeling (DSM) initiatives constitutes the collaboration between three distinguishable roles (see Figure 7). Within literature, there exist slightly different specifications of those roles (cf. [32], [40]).

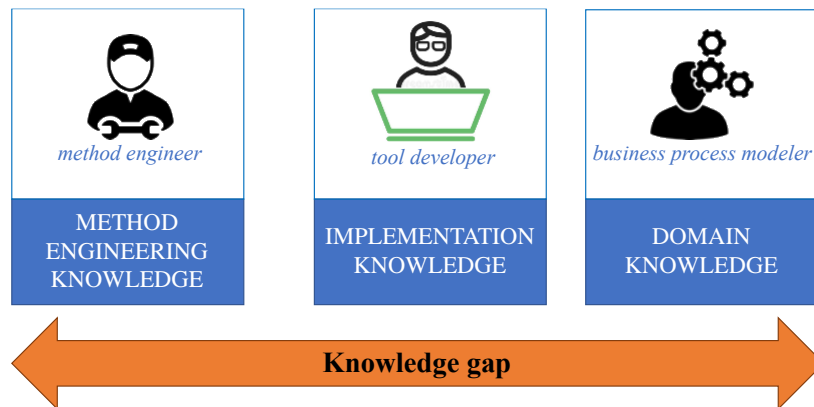


Figure 7 Roles in domain-specific BPM

– own representation, icons from [63] and [31]

All three roles are responsible for different parts within the BPM initiative. Whereas the method engineer designs the modeling method, the business process modeler is the person with knowledge of the specific application domain. The tool developer constitutes the connecting link, as he or she is responsible for implementing the designed modeling method. There exists a knowledge gap regarding design and implementation and the expert knowledge of the respective domain.

Within literature, different definitions of a domain can be found. In Collins [23], a domain is defined as a “particular field of thought, activity, or interest, especially one over which someone has control, influence, or rights”. Prieto-Díaz [96] provides a framework for domain-analysis and defines a domain in the context of software engineering “as an application area, a field for which software systems are developed” [96]. Karagiannis et al. [54] state that there is no clear boundary between general-purpose and domain-specific and that a domain might refer to a narrow application area as well as a whole business sector.

2.2.1 Degree of Domain-specificity

Within literature, a distinction is made between general-purpose modeling languages (GPML) and domain-specific modeling languages (DSML). The word *domain* can refer to a business sector, a community-driven paradigm, narrow application area, or a single enterprise case [cf. 54]. In the context of this thesis, the focus is set on industries as well as intra-company application areas. Karagiannis et al. [54] propose an approach to use different domain-specific modeling languages within one tool without having the trade-off of choosing between them. The underlying method is called Fundamental Conceptual Modeling Languages (FCML). Application domains can be divided into verticals (financial services, telecommunications, public administration, manufacturing) and horizontals (business process modeling, application development, workflow management, knowledge management) [cf. 56, p. 5].

Mernik et al. [79] divide the design of domain-specific modeling languages into the three phases *decision phase*, *analysis phase*, and *design- and implementation phase*. Whereas the three phases described by [79] constitute a possible starting point for the development of a DSML, this procedure lacks the need that arises due to the dynamic nature of domains and their environment. As requirements inevitably change in the course of time, the phases should include a re-evaluation component. The artifact developed within this thesis aims to explicitly address the need for change and re-evaluation. DSLs appear in the form of textual or graphical languages³. The focus of this thesis is set on the design of graphical or visual modeling languages.

Within literature, different approaches towards DSBPM can be found. The lowest degree of domain-specificity is to use a GPML. Some authors use GPML and provide guidance on how to adapt it to specific application domains (cf. [66], [45]). Other authors argue that only a DSL can fully grasp the context of the domain [cf. 60].

2.2.2 Frameworks

There exist different acknowledged frameworks, which aim at providing a categorization scheme for processes or application domains. Some of them are described in the following

³ For a comparison the reader is referred to [89] S. Pissierssens, "Revealing the scientific basis of graphical representation design."

and serve as a data source for the development of the Domain Framework (DF) as part of the DIF within this thesis. The goal of this section is to show similarities and differences between the varying categorization schemes and to provide an overview.

Heitkötter [45] proposes a framework for creating domain-specific process modeling languages. The author's argument is to generalize DSLs and introduces a transformation mechanism to convert DSLs into the BPMN2.0 standard. DSLs4BPM builds on a minimal set of common concepts, which is extended for specific domains. Kelly and Tolvanen [60] describe a domain framework as an interface connecting the platform components with generated code for the purpose of automation.

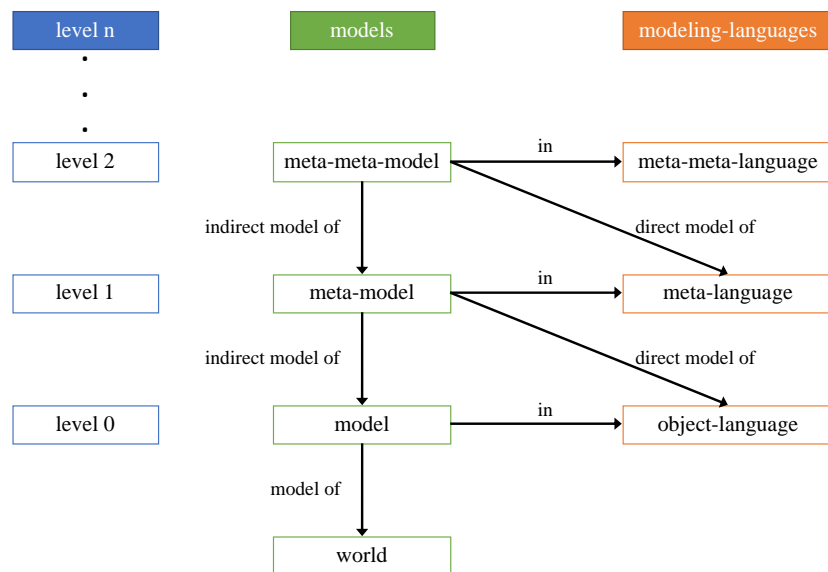
The American Productivity and Quality Center (APQC) [7] has developed the Process Classification Framework® (PCF) for business process modeling and mapping. There exists a cross-industry version of the PCF, which aims at being applicable to all kinds of enterprises and can be used whenever no industry-specific PCF® is available. In order to allow for industry specific representation, additional versions for 19 industries have been issued up to now: aerospace and defense, airline, automotive, banking, broadcasting, city government, consumer electronics, corrosion, consumer products, downstream petroleum, education, health insurance payor, healthcare provider, insurance, pharmaceutical, retail, telecommunications, upstream petroleum, and utilities. An example for a domain-specific process classification is e.g. eTom [48], which is specifically known for telecom operations. Aitken et al. [2] propose a process classification framework, which provides guidance on finding the appropriate model types. The authors distinguish between the levels of contextual, conceptual, logical, and physical and propose certain model types for each level on the case of a health agency implementing an e-health initiative. Other process classifications can be found, e.g. for change management [cf. 3] or strategic decision-making [cf. 68]. APICS [6] offer frameworks for supply chain management specific subjects, like the SCOR (supply chain operations reference), DCOR (design chain operations reference), CCOR (customer chain operations reference), M4SC (managing for supply chain performance), and PLCOR (product lifecycle operations reference model).

It is noticeable, that there exist different viewpoints on and degrees of domain-specificity within literature. Different views on BPs or the use of different model types may account for domain-specificity [cf. 2], or the creation of completely new DSBPM languages with an own syntax, semantics, and notation.

2.3 Model and Meta-model

Stachowiak [113] defines three fundamental model properties, namely mapping, reduction, and pragmatism [cf. 113, p. 313]. The mapping property focuses on the original, which is mapped by the model. The reduction property stresses the incompleteness of the model, which shows only the relevant aspects of the specific viewpoint. Pragmatism is the property specifying the usage aspect of the model for a certain application area within a specific time-frame. Whereas different definitions of models exist (cf. [123], [104]), the model properties by [113] are seen as relevant within the context of this thesis, as they complement the subject of business process modeling.

Strahringer [114] defines a meta-model as a model of a model, which is a linguistic-descriptive model describing the language of the subordinate model. The concept of metaization is depicted in Figure 8, where the recursive nature of describing a model in a meta-language is shown. The advantage of meta-modeling is the reduction of complexity by abstraction (cf. [56], [114], [113]). As an example of this reduction of complexity in meta-models, classes may represent the superordinate of activities of different kinds, or tasks carry attributes like duration, costs, descriptions, or organizational assignments, among others [cf. 109]. Zacarias et al. [124] identify general business process concepts of BP meta-models, namely the meta-elements process components, process connectors, process resources, organizational features, and specific features.



*Figure 8 Language-based meta-model concept
– own representation based on [114, p. 24]*

In the context of BPM, meta-modeling is a widely used concept for depicting BPs on a higher abstraction level and to enable the comparison between different BPs (cf. [62], [55], [58]). Karagiannis and Woitsch [58] emphasize the strong interrelation between BPM and knowledge engineering and propose meta-modeling as an approach to integrate the two disciplines. Other application fields of meta-models are design (in the sense of an inheritance mechanism or reference structure) and integration (mapping of meta-models by the use of the meta² model) [cf. 55].

In recent developments, agile meta-modeling approaches are gaining more and more momentum in order to address the complexity of BPM (cf. [124], [52], [115], [33]). For the purpose of business process flexibility and agility in order to address the issue of changing requirements, [124] propose a meta-model. Raschke [97] defines operational agility in the context of BPM as a construct of the four components reconfigurability, responsiveness, employee adaptability, and process-centric view. Thiemich and Puhlmann [115] provide a meta-model for BPM, which is not by itself agile but based on agile software development methods.

Enablers of flexible meta-modeling are meta-modeling platforms like e.g. ADOxx [16], MetaEdit+ [80], or OpenPonk [116], which provide a meta² structure and allow for the design of modeling methods on the meta level. On the meta layer, concepts, notations, semantics, and syntactic and semantic constraints are implemented. In this layer, also domain-specific concepts are taken into account. The modeling layer depicts the concrete model using a specific modeling language.

In the context of this thesis, the ADOxx [16] meta-modeling platform is used as an implementation platform, as it includes all necessary concepts needed for meta-model creation [cf. 34]. The meta-model builds the base for the implementation of domain-specific concepts. In their work, [61] compare and analyze several meta-modeling languages by comparing their meta-meta-models⁴. As the result of the investigation, several commonly used notions were identified. All of the analyzed meta-meta-models have the elements object, relationship, and attribute, whereas differences were found in structuring and reuse capabilities of meta-model elements. In the ADOxx meta² model, these elements are represented by class (corresponding to the object element from above), relationship, and

⁴ In the general literature, meta-meta-model and meta² model are used as equivalents

attribute. The reuse and structuring capabilities are represented by the additional element model-type. With these four notions, the syntax and semantics of the modeling language can be defined [cf. 117].

2.4 Modeling Method Building Blocks

A modeling method can integrate several modeling languages (cf. [54], [36]), and can be designed for specific needs of a certain domain. A modeling method enables the combination of all concepts of the previous chapters into one framework, combining language, procedure, and mechanisms & algorithms requirements (cf. [54], [34], [51], [36]).

Karagiannis and Kühn [56] define a modeling method by its components modeling technique on the one side, and mechanisms and algorithms on the other side. The modeling technique can further be divided into modeling language and modeling procedure. Figure 9 depicts these building blocks, and their components are shown in greater detail within the subchapters. The colors emphasize, whether the components belong to the modeling language (yellow), procedure (red), or functionality (green).

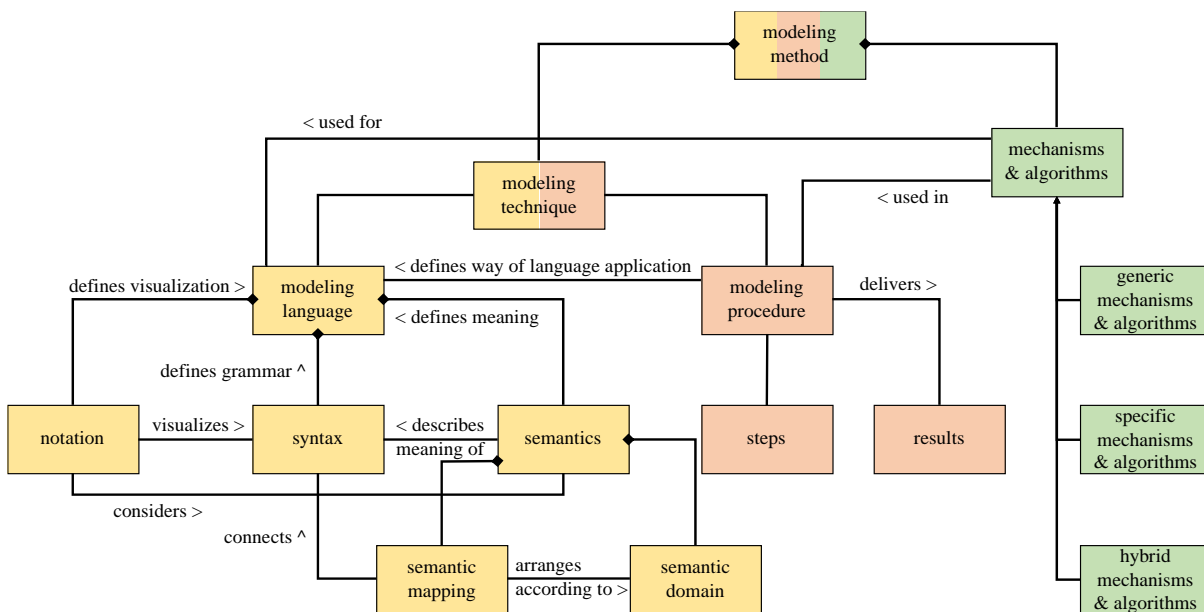


Figure 9 Modeling method building blocks

– own representation based on [56]

Whereas [56] provide the structure definition of a modeling method, [40] propose Agile Modeling Method Engineering (AMME) as a methodology to develop a modeling method. It builds the foundation of the OMiLAB environment, which promotes using metamodeling

for the design of DSLs [cf. 40]. It is domain-independent and focuses on the interaction between modeling and machine-processing abilities of those models. The characteristics of this methodology are based on changing requirements, namely adaptability, extensibility, integrability, operability, and usability [cf. 52].

Within this thesis, the AMME approach is seen as the fundamental structure for modeling method engineering. As it is conceptualized on the meta² level (see chapter 2.3 for an explanation of the different meta-modeling concepts), it is seen as a given orientation for modeling method development and deployment.

Visic et al. [117] recognize the dynamic nature of modeling method engineering due to changing requirements and therefore propose a domain-specific language for modeling method engineering. The phases are inspired by the principles of agile software engineering and based on the framework by the Open Model Initiative Laboratory [87].

2.4.1 Modeling Language

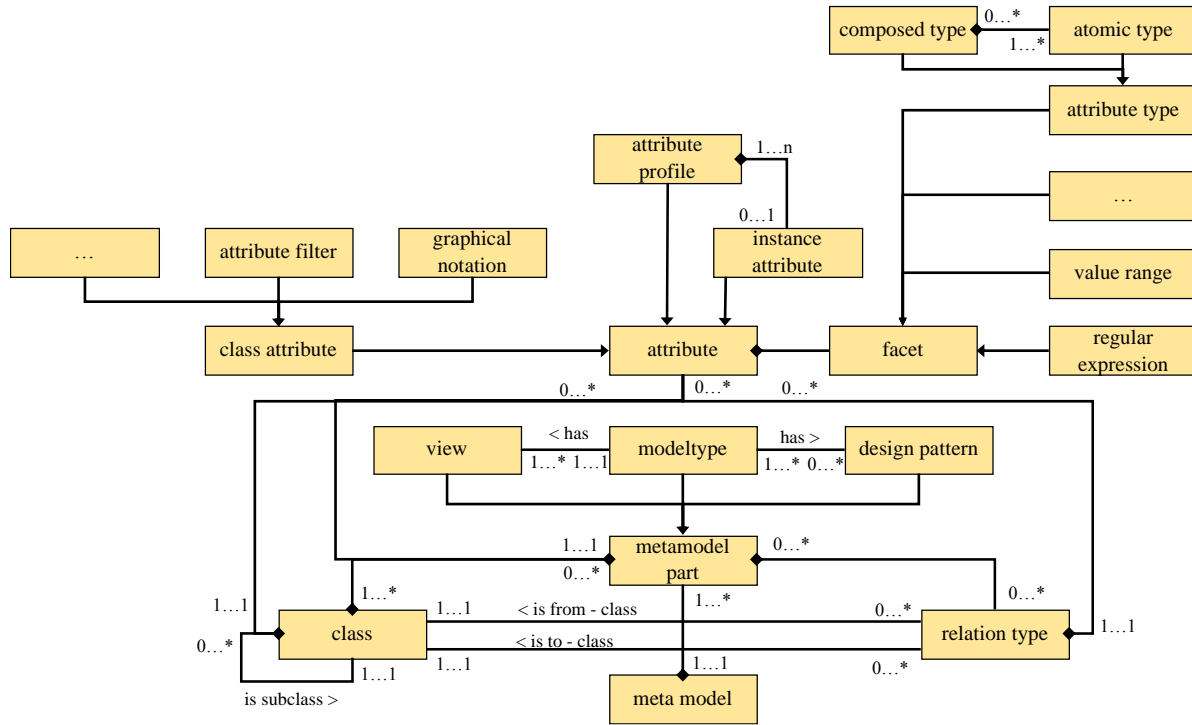
The modeling language (ML) (see Figure 9) constitutes the primary building block and is further divided into syntax, semantics, and notation. The syntax of the ML establishes the grammar by the definition of a set of rules. The semantics of the ML provides the syntax with meaning. The notation gives the graphical representation of the ML by domain-specific symbols [cf. 117].

Kelly and Tolvanen [60] propose several language definition guidelines in their book. Following these guidelines to create a domain-specific ML significantly increases acceptance and likelihood of usage within the company area where the DSL is deployed. The language definition guidelines are as follows [cf. 60]:

- Use the same names and naming conventions already in use within the domain.
- Keep the language simple in order to allow a high level of abstraction. The goal is to satisfy the identified needs first and if necessary, additional concepts can be added at a later point.
- A precise definition of each modeling concept is key. Examples should show alternative cases, concepts, and behavior.
- Language extension possibilities can be included by adding extension concepts, e.g. an object, which can be connected to all other objects and has only one

description property. This makes sense, if the modeling language is still incomplete or the domain relatively new.

Figure 10 depicts the general concepts of a modeling language on the meta-level, as proposed by [67].



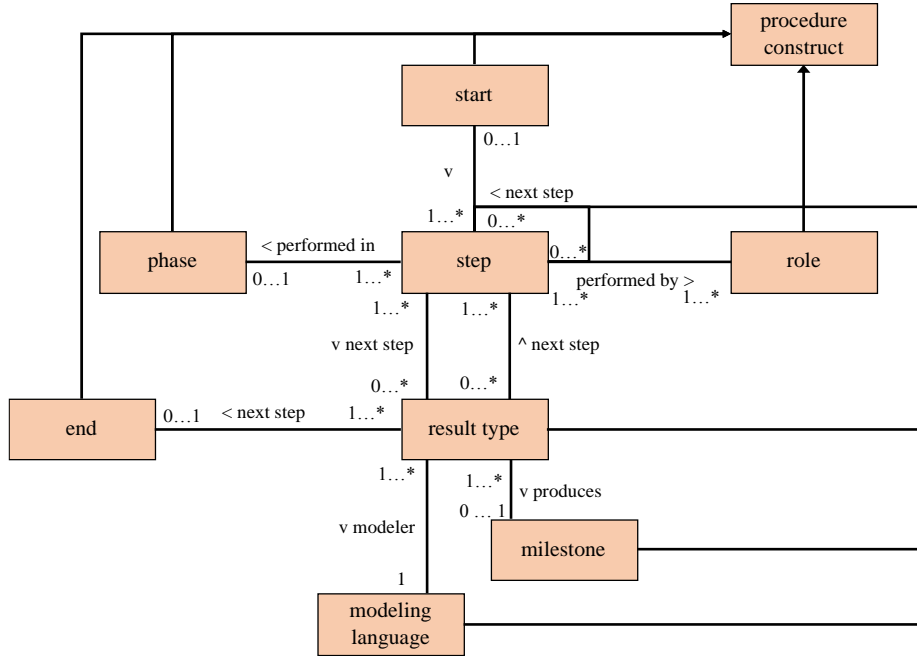
*Figure 10 Generic meta-model of modeling language
– own representation based on [67]*

The generic meta-model shown in Figure 10 includes all possible elements, which can be used within the language of a modeling method. It is noticeable, that not all of these elements might be necessary within a certain domain.

2.4.2 Modeling Procedure

The modeling procedure is defined for the business process modeler and provides guidance on how to produce a valid model. It formally describes the sequence in which certain model types should be created and formulates the steps necessary to produce a coherent model [cf. 54]. The sequence, in which model types are used, can for instance range from generic to specific (i.e. from modeling the process landscape to modeling single processes and sub-processes).

The generic meta-model of the modeling procedure after [67] is depicted in Figure 11. It is an essential part of the modeling method as it serves as a guideline for business process modeling within the specific domain. It makes the task of modeling more user friendly and reduces the risk of producing false models.



*Figure 11 Generic meta-model of modeling procedure
– own representation based on [67]*

2.4.3 Modeling Mechanisms & Algorithms

The purpose of the mechanisms and algorithms building block (see Figure 9) is to provide the modeling method with functionality. Here, [54] propose a generic-to-specific description of the mechanisms & algorithms integrated within a modeling method. In the context of DSMLs, authors commonly argue that the language must provide some kind of services or functionality, like e.g. simulation features, debugging functions, or a compiler to some target environment (cf. [9], [60]). Figure 12 depicts the general modeling concepts in the meta-model of the mechanisms and algorithms building block.

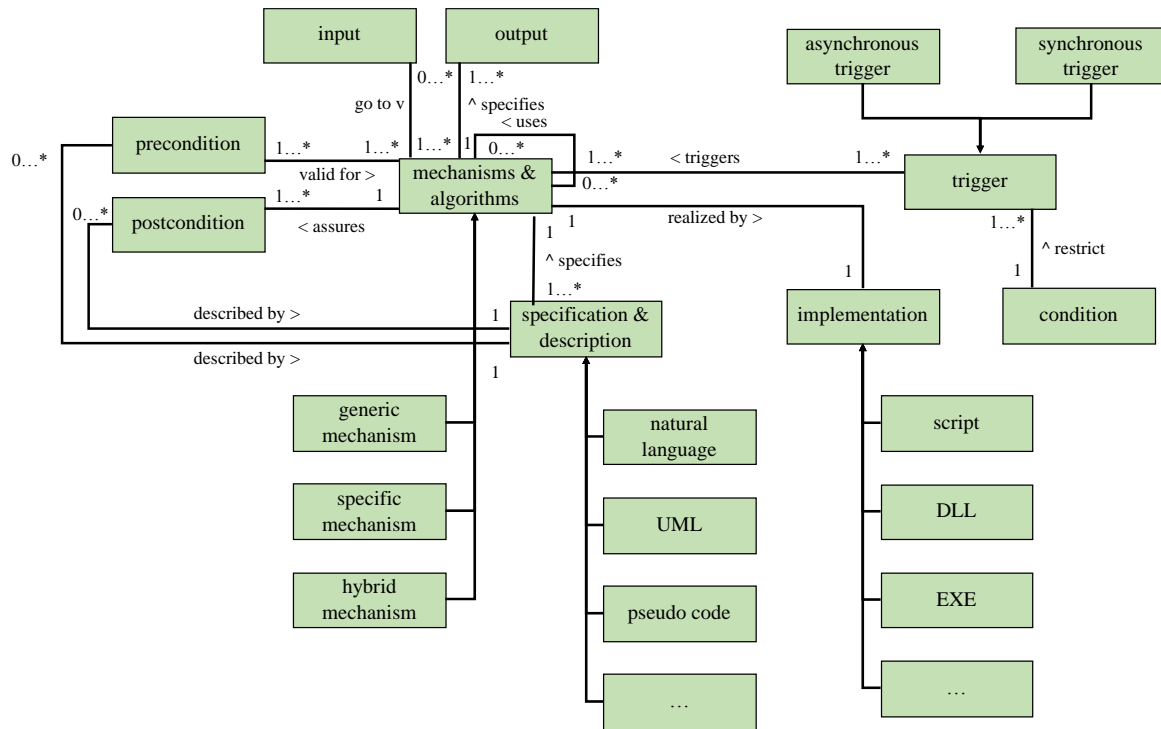


Figure 12 Generic meta-model of mechanisms and algorithms

– own representation based on [67]

2.5 Interim Conclusion

Within this chapter, relevant literature in the field of domain-specific business process modeling has been collected, summarized, and analyzed in the light of its relevancy for the further course of this thesis. The focus was set on the extraction of crucial concepts, which are used in chapter 3 to analyze existing DSM languages and to derive a basis for the development of the Domain Integration Framework in chapter 4 as the result of the analyses. The relevant concepts within literature and their importance for the following chapters are summarized in Table 2.

Table 2 Revelations from the literature review for the further course of the thesis

– own representation

| Chapter in state of the art | Revelation for this thesis | Knowledge used within which part |
|-----------------------------|---|----------------------------------|
| 2.1.1 | Enabling an integrated view of both concepts, BP modeling and BP re-engineering | DIF |

| | | |
|-------|---|---------------------------|
| 2.1.1 | Concepts of GP BPM languages can be useful for the design and re-use of modeling languages | DIF, Case Study |
| 2.1.1 | Categorizing processes as management, core, and support processes | DIF |
| 2.1.1 | Crucial BP elements are activity, artifact, actor, and resource | Case Study |
| 2.1.2 | Use of an agile and domain-centric approach to engineer requirements, i.e. by user stories | Case Study |
| 2.1.2 | Distinguish between different categories of requirements, i.e. stakeholder requirements, goal requirements, boundary requirements | DIF, Case Study |
| 2.1.2 | Distinguish between requirements at design-time and changing requirements at usage-time | DIF, Case Study, Analysis |
| 2.2 | Differentiation between domain context (external view) and domain analysis (internal view) | DIF |
| 2.2.1 | Focus on the visual value of graphical models | Case Study, Analysis |
| 2.2.1 | Within different process levels, a different degree of domain-specificity can be considered | Case Study |
| 2.2.2 | For the domain context, industries, the level of detail (granularity), and the question whether the process is a core, management, or support process constitutes a good starting point | DIF, Case Study |
| 2.2.2 | For the domain analysis, requirements as well as quality criteria reveal insights into the processes themselves and their evaluation | DIF, Case Study |
| 2.3 | Focus on meta-models, because flexibility as well as domain-specificity is enabled on the meta-level | Analysis |

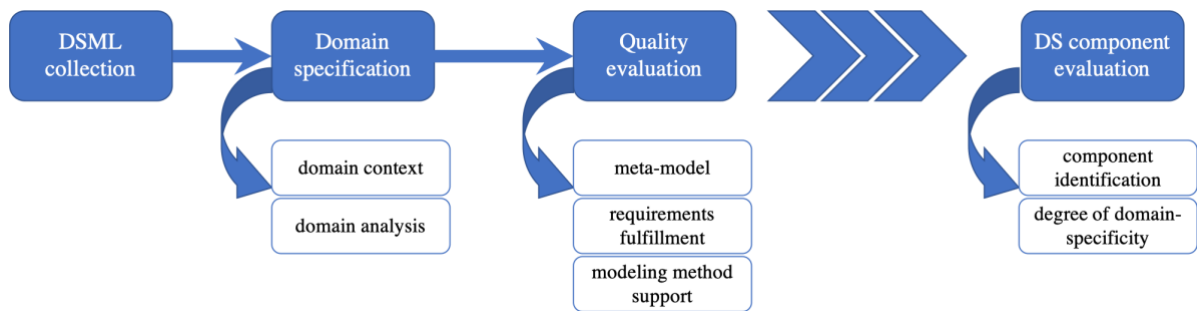
| | | |
|-----|--|---------------------------|
| 2.4 | In order to provide guidance and acceptance for usage, a whole modeling method is needed | DIF, Case Study, Analysis |
|-----|--|---------------------------|

Among the advantages of DSBPM languages, their functionalities for automatic code generation are often mentioned (see chapter 2.2). The emphasis of this thesis, however, lies in the value of the graphical representation of DSBPM rather than the production of code. Nevertheless, this could constitute a starting point for considerations for automation e.g. in the form of workflow engines or ERP systems. In the context of visual value of a model, the semantics of a modeling language contribute to domain-specificity. For example, the symbol of a steering wheel might be needed when defining business processes within the automotive industry.

The Domain Integration Framework (DIF) developed within this thesis should not be confused with the domain framework defined by [60]. The authors describe a domain framework as a layer of code situated between the general components including the meta-modeling platform, and the generated code. It serves as a means to avoid code repetition as it resembles all elements, which are common to all applications within the domain. In contrast to that, this thesis and the DIF developed within it shall provide a guided modeling approach to help closing the gap between the knowledge of a method engineer and a domain expert. Its contribution is the provision of a structured way to analyze the specific application domain, such that no important domain-specific information is forgotten during the design of the modeling method for business process modeling.

3 Analysis of DSML

The literature overview shown in chapter 2 revealed valuable insights for the categorization and evaluation of DSM languages (DSML). In order to identify, what domain-specific in the context of BPM means and to extract knowledge about domain-specific concepts, a literature-collection and -analysis procedure is used (see Figure 13). Here, DSML within literature are collected and analyzed in a structured manner in order to derive valuable insights for the creation of new DSML.



*Figure 13 Literature-collection and -analysis procedure
– own representation*

The first step resembles a literature review process, where different DSML are collected in the manner described in chapter 1.3. Secondly, the respective domain of each DSML is specified. Every DSML is evaluated according to its quality, which in this case is the availability of a meta-model of the ML, whether the requirements are fulfilled, and whether it is supported by a complete modeling method. The goal of the procedure is to consolidate the knowledge gained throughout the process within the last step - the evaluation of components - which make a DSML domain-specific. By comparing differences and similarities between the DSMLs, those components are identified and their respective contribution to domain-specificity is elaborated.

A reference framework for the evaluation of BPMLs is offered by [36]. The authors propose general requirements for modeling languages (formal, user-related, application-oriented), requirements for BPM (fundamental requirements for modeling languages, business and operational requirements, control structures, exceptions, integrity requirements, support for the development of information systems, support of individual adaptations, documentation, specification), and the embedding within a modeling method (project specific roles and resources, modeling procedure) as criteria for evaluating and

selecting an appropriate modeling language or method. A further extension and concretization of this evaluation method is proposed in [37]. Within this chapter, the evaluation method by [37] is used to analyze DSML in a first step and to extract DS concepts in a second step.

3.1 Domain Specification

To sum up the previous discussions, a domain within this thesis is defined as follows:

*The following definition of a domain is derived from the findings of the literature review (chapter 2) and shall provide a guideline for the analysis of existing DSML within this chapter. A domain can be seen as an area or focus of application. It can be analyzed on the one hand by its **context**, i.e. its boundaries and dependencies to other domains and the classification of its processes (see chapter 2.2.2). On the other hand, inward looking characterizations or **analysis** can be made based on the modeling needs of this respective domain (see chapters 2.1.2, 2.2.1).*

The definition above implies, that there exists an external and an internal view of a domain. With these two points of view, the analysis of several DSML was conducted and is therefore divided into two parts – an external domain context part and an internal domain analysis part.

Following Table 3 shows the results of the domain context analysis of the DSML.

Table 3 Domain context of DSML
– own representation

| DSML | Domain context | | |
|-------------|-------------------|--------------|-------------------------|
| | Industry | MCS-level | Pursued level of detail |
| E-MEMO [35] | Online, insurance | Core process | Abstraction |
| MPN BP [82] | Banking | Core process | Abstraction |

| | | | |
|---|---|--|-------------|
| eGPM [18] | Governmental, insurance, car rental | Core process | Abstraction |
| DSLs4BPM [45] | Information technology | NA | Abstraction |
| PICTURE [12] | Governmental, banking | Management process, core process, support process | Abstraction |
| 3PL [13] | Robotics | Core process | Detail |
| SIMchronization [94] | Supply chain management, production, logistics, maintenance | Core process | Abstraction |
| ComVantage [53] | Supply chain management, plant commissioning, production, maintenance | Core process | Abstraction |
| State Social Insurance Agency (SSIA) [14] | Social security | Core process | Abstraction |
| DSL for automation systems [70] | Automation systems | Management process, core process, support process | Detail |
| PISCAS [95] | Pisciculture automation system | Core process | Detail |
| Flight control domain- specific language (FCSL) [100] | Safety critical flight control software | Core process | Detail |
| GISMO [28] | Development of gestural interaction applications | NA | Abstraction |
| Tramway Control Framework (TCF) [43] | Railway control systems, tramway control systems | Core process | Detail |

| | | | |
|---|--|---|-------------|
| Integrating the Healthcare Enterprise (IHE) Framework [5] | Medical e-services | Core process | Abstraction |
| Project assessment diagrams (PAD) [9] | Highlighting review and assessment of business processes | NA | Abstraction |
| Industrial Business Process Management (IBPM) [15] | Industrial | Management process, core process, support process | Abstraction |

It is notable, that many of the DSML are claimed to be valid in more than just one domain. Only for very narrow domains, which also focus on system implementation like the robot navigation language 3PL [13], no further application domains were named by the respective authors. Moreover, most of the DSML focus on core processes and do not allow for the integration within an organizational architecture. As to the pursued level of detail within each language, it is differentiated between abstraction and detail. Languages that aim for abstraction are more focused on reducing complexity by making use of a higher abstraction-level and leaving out unnecessary details. Languages pursuing a high level of detail specifically aim to address complex or detail-intensive issues.

Table 4 depicts the results of the domain analysis, applied on the DSMLs.

Table 4 Domain analysis of DSML

– own representation

| DSML | Domain analysis | | |
|-------------|--|--|---|
| | Underlying modeling language (if applicable) | Model types | Elements |
| E-MEMO [35] | Domain-specific | Strategy network model, business process model | Process, decomposition hierarchy, event, control structure, exception, note, organizational unit, resource, |
| MPN BP [82] | Modified Petri Net | Organizational structure model, | Activity, resource, control, flow, organizational structure |

| | | | |
|----------------------|-----------------|---|---|
| | | business process model | |
| eGPM [18] | Domain-specific | Cooperation view, use-case diagram, process landscape, IT landscape, conceptual model, work environment model | Actor, group, meeting, object, business case, aggregation, note, informs with object, passes object, edits object, initiates |
| DSLs4BPM [45] | BPMN2.0 | NA | Process, organizational element, subprocess, variant, process building block, PBB occurrence, derived DSL |
| PICTURE [12] | Domain-specific | Process landscape, process model, procedure model | Process block, subprocess |
| 3PL [13] | Domain-specific | Flow chart | Configuration, flow, gateway, connection |
| SIMchronization [94] | Domain-specific | Supply chain network model, resource model, component model | resource, plan, information object, material flow object, item |
| ComVantage [53] | Domain-specific | Resource pool, business and organization structure, value structure, process model, mobile IT-support model, orchestration model, location structure, information space model, station structure, machine state model | Many elements, combined into the structure groups KPI, market, mobile support, value exchange flow, value structure, business model, orchestration, enterprise structure, business structure, location structure, permission pool, information space, evaluation process, requirements process, business process, interaction flow, notification, navigation, collaboration |

| | | | |
|--|----------------------------|---|---|
| State Social Insurance Agency (SSIA) [14] | ProMod | Business process model, information systems diagram, customer service diagram, organizational structure, regulations and local instructions diagram, information artifacts, customer services diagram | Activity, event, sequence and message flows, data objects |
| DSL for automation systems [70] | Domain-specific | System model | Automation domain object, name, voltage, output, input, in, out, automation domain wire connection |
| PISCAS [95] | DSL for automation systems | Graphical overview, wiring plan, list of parts, labels for wiring closet | Pond, switch, aerator, feeder, light, module, time switch, twilight switch, PH value, waterlevel, temperature sensor, muddiness sensor |
| Flight control domain-specific language (FCSL) [100] | Domain-specific | Fault model, behavior model, Data flow/ state flow | Processor, device, interconnection network, partition, process, task, function element, relationship |
| GISMO [28] | Domain-specific | Gestural interaction model | Element, object state, gesture type, interaction controller, movement direction, variable, state variable |
| Tramway Control Framework (TCF) [43] | Domain-specific | Tramway network model, deterministic sequential state machine, safety monitor state machine, signal setting table, point position table, route conflict table, route definition table | Abstract signals, abstract points, counters, abstract sensors, route requests, signal drivers, point drivers, sensor drivers, tram comm drivers, signal type, point type, sensor type, tram |

| | | | |
|---|-----------|--|---|
| Integrating the Healthcare Enterprise (IHE) Framework [5] | UML, BPEL | Sequence diagram, activity diagram, administrative process flow, patient registration transaction, modality worklist model | Actor, activity, transaction, flow, patient class, document, type, variable, message, correlation set, partner, process, port type, service link type |
| Project assessment diagrams (PAD) [9] | ProMod | NA | Elements of UML activity diagrams, elements for controlling execution duration (setTimer, checkTimer), link between system and editor |
| Industrial Business Process Management (IBPM) [15] | BPMN2.0 | Company map, business process diagram, document model, working environment model | Swimlane (vertical), swimlane (horizontal), process, performance, actor, external partner, aggregation, note, elements of BPMN2.0, document, organizational unit, performer, role |

From the table above can be seen, that DSMLs strongly vary in their size, recognizable by the number of model types and elements. Moreover, DSMLs addressing similar domains use similar model types and elements. This can be seen e.g. when comparing the ComVantage [53] and the SIMchronization [94] modeling methods (MM). Even though the SIMchronization MM focuses on a narrower application domain, both include models to depict resources and supply-chain processes. Also, both DSMLs addressing the public administration domain⁵ include a process landscape and elements enabling process aggregation. It is also noticeable, that many DSMLs are based on GPMLs or include model types, which enable the use of GPMLs (e.g. IBPM [15] uses BPMN2.0 in the business process diagram model type).

3.2 Quality Evaluation

An extensive catalogue of questions providing a useful tool to evaluate the quality of modeling methods is proposed by [37]. The evaluation questionnaire is systematically

⁵ i.e. eGPM [18] H. Breitling and S. Hofer, "Schwerpunkt-beispielhaft gut modelliert: Exemplarische Geschäftsprozessmodellierung in der Praxis," *Objekt Spektrum*, no. 6, p. 8, 2012. and PICTURE [11] J. Becker, D. Pfeiffer, and M. Räckers, "Domain specific process modelling in public administrations—the PICTURE-approach," in *International Conference on Electronic Government*, 2007, pp. 68-79: Springer.

structured into different categories and used as a base within this thesis to evaluate the collected DSML. Within this section, an adapted version of the question catalogue by [37] is used, extended by categories which are seen as relevant in the context of this thesis. For example, the section “BPM specific criteria” is changed to “Domain-specific criteria” for the evaluation of DSMLs.

The DSMLs are assessed regarding general criteria and domain-specific criteria. The general criteria encompass formal, user-oriented, and usage-oriented criteria. The domain-specific criteria contain questions regarding the functionality, model types, concepts, adaptability, documentation, meta-model support, and modeling method support. The original questionnaire consists of 119 sub-questions, which are used for the evaluation within this thesis as a guidance for the evaluation of the DSMLs. However, the results of the evaluation are shown in an aggregated way, presenting the main points of the questionnaire. The results of the evaluation are summarized in Table 5 and the complete evaluation sheet is attached in Appendix A.

Table 5 Quality evaluation of DSML

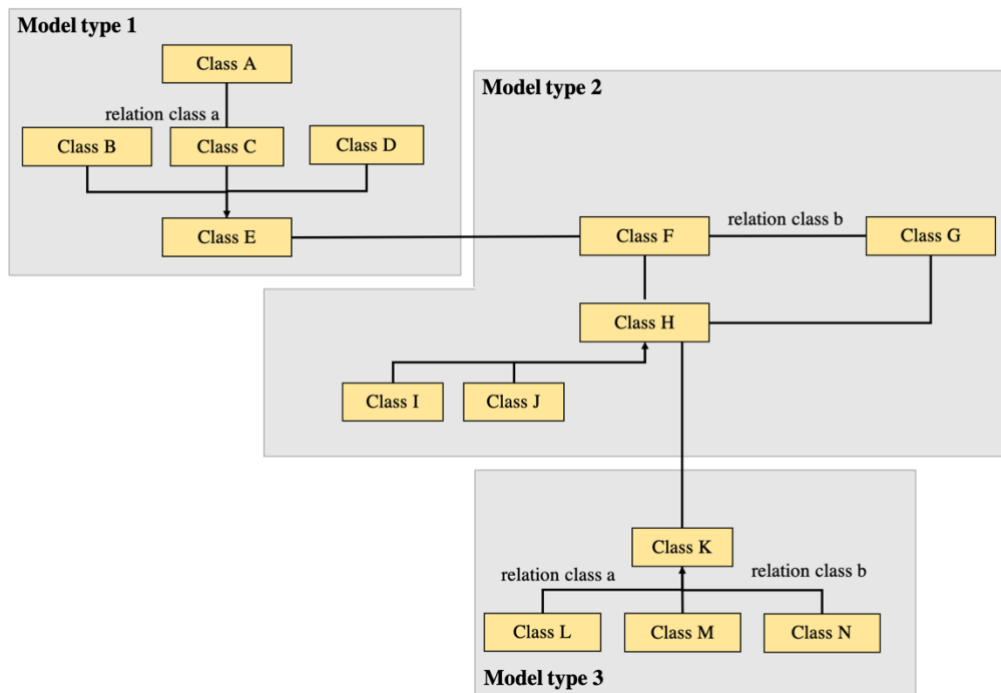
– own representation

| Evaluation of DSML | General criteria | | | Domain-specific criteria | | | | | | |
|--------------------|------------------|------------------------|-------------------------|--------------------------|----------------|-------------|-------------------------------|------------------------|--------------------|-----------------------|
| | Formal criteria | User-oriented criteria | Usage-oriented criteria | DS functionality | DS model types | DS concepts | Adjustments on platform level | Provided documentation | Meta-model support | ML embedded into a MM |
| E-MEMO [35] | ● | ● | ● | ● | ◐ | ● | ● | ◐ | ● | ◐ |
| MPN BP [82] | ◐ | ◐ | ◐ | ● | ◐ | ◐ | ◐ | ◐ | ◐ | ◐ |
| eGPM [18] | ● | ◐ | ● | ● | ◐ | ◐ | ◐ | ◐ | ○ | ● |
| DSLs4BPM [45] | ● | ◐ | ◐ | ● | ◐ | ● | ◐ | ◐ | ● | ◐ |
| PICTURE [12] | ● | ● | ◐ | ◐ | ◐ | ◐ | ◐ | ◐ | ◐ | ● |

| | | | | | | | | | | |
|---------------------------------|--|--|--|--|--|--|--|--|--|--|
| 3PL [13] | | | | | | | | | | |
| SIMchronization [94] | | | | | | | | | | |
| ComVantage [53] | | | | | | | | | | |
| SSIA [14] | | | | | | | | | | |
| DSL for automation systems [70] | | | | | | | | | | |
| PISCAS [95] | | | | | | | | | | |
| FCSL [100] | | | | | | | | | | |
| GISMO [28] | | | | | | | | | | |
| TCF [43] | | | | | | | | | | |
| IHE Framework [5] | | | | | | | | | | |
| PAD [9] | | | | | | | | | | |
| IBPM [114] | | | | | | | | | | |

Table 5 summarizes the main findings of the quality evaluation of the investigated DSMLs. It has to be stated that due to restrictive information, assumptions about categories were made to the best of knowledge and no claim to completeness and correctness is made. The table shows, to what extent different DSMLs fulfilled the criteria on a scale from one to four (shown as quartered circles). The meta-models of the DSMLs from the OMiLAB platform [87] all follow the modeling method approach described in chapter 2.4. This leads to exceptionally good results in the quality assessment. An explanation for the high-quality modeling methods from OMiLAB is the provision of a standardized scheme, which still allows for flexibility on the meta-level. On the one hand by the modeling method approach with its three components (see chapter 2.4), and on the other hand by making active use of meta-modeling to depict dependencies not only generally but structured according to model

type building blocks. An example of structuring a meta-model according to its model types can be seen in Figure 14.



*Figure 14 Example of a meta-model structured according to its model types
– own representation*

Figure 14 shows the representation of the meta-model of a modeling method structured according to the model types it consists of. The main elements are the different classes, which can consist of sub-classes. The classes are connected by relation classes, which might be specific for groups of classes or just two classes. Each class and relation class can furthermore inhere attributes, which are not shown here due to reasons of simplicity.

The previous analyses of domain-specific business process modeling languages regarding their domain context, domain analysis, and quality revealed valuable insights. By comparing their differences and similarities, patterns could be identified. Those patterns are listed in the next chapter and their contribution to domain-specific BPM is estimated.

3.3 DS Component Identification

Reaching the end of this chapter, the first major phase of this thesis is accomplished. Namely, to identify differences and similarities by analyzing DSML and to derive

components, which make the MLs domain-specific. Those components serve as the main focus points of the Domain Integration Framework developed in the next chapter.

During the analysis of DSMLs, the different degrees of domain-specificity found in the modeling languages were:

- General-purpose language applied within a specific domain
- General-purpose language extended by domain-specific concepts
- Completely domain-specific language

Table 6 shows the identified concepts, which contribute to domain-specificity of business process modeling languages and methods.

Table 6 Domain-specific component identification
– own representation

| Component | Contribution to domain-specificity | Explanation |
|--------------------------------|------------------------------------|---|
| Process framework | ★ | Process frameworks for specific domains can provide useful information on the vocabulary used within the process context. |
| Requirements assessment method | ★ | There exist different assessment methods (see chapter 2.2.1), which might be suitable within different domains. |
| Meta-model | ★ | The meta-models of the DSMLs differed the most throughout the analysis, including its elements and model types. This underlines the hypothesis, that domain-specificity is located on the meta-level of the language. |
| Model type | ★ | The model types of a DSML build a sub-part of the language meta-model. Still, the selection and combination of model types itself contributes to domain-specificity to a great extent. |

The process framework, which is part of the domain context analysis can provide useful information regarding process steps, hierarchies, and domain-specific concepts. Especially domain-specific process frameworks offer a commonly accepted vocabulary, which is essential for the later acceptance of the DSMM to be developed. However, for its contribution to domain-specificity it was rated with half a star, as several techniques for

extracting domain-specific knowledge exist, e.g. interviews with domain-experts or user stories.

The requirements assessment method also contributes to domain-specificity, as it helps to identify concepts and requirements specific to the respective domain. Formulating general requirements for the DSMLs might be valid, but not sufficient to fulfill all existing needs. The meta-model shows all the essential classes, relations, and attributes of a modeling language and can be seen as the key concept of domain-specificity. Not only does it offer the possibility to compare modeling languages with each other. It also provides the opportunity to use specific blocks of the meta-model and integrate them into meta-models of other languages, therefore enabling extension and re-usability. The domain analysis conducted within this chapter revealed the tendency, that the better the meta-model of a language, the better the quality of a language. The analysis has shown, that the model types used vary considerably between the DSMLs. This indicates, that the model type contributes to a great extent to domain-specificity and needs to build a central part of DSMM design. Considerations about the right amount and the right kinds of model types have to be made.

To sum up the previous chapter, the components contributing to domain-specificity were identified by a thorough and systematic analysis of seventeen business process modeling languages with varying degrees of domain-specificity. Their external domain environment as well as their internal domain-facets were analyzed, put into context, and compared to each other. The DS components identified build the foundation for the Domain Integration Framework, which is introduced in the following chapter.

4 Results – the Domain Integration Framework (DIF)

After the introduction of related concepts and literature in chapter 2 and the profound analysis of DSMLs in chapter 3, implications are derived and put together in the heart of this thesis – the Domain Integration Framework (DIF) for business process modeling. Whereas the focus of the previous chapter was to show and categorize a variety of existing DSMLs and identify DS components, the following chapter consolidates the gained information. The resulting DIF is described and in the later course proven by applying it on the automotive assembly line case study (see chapter 5).

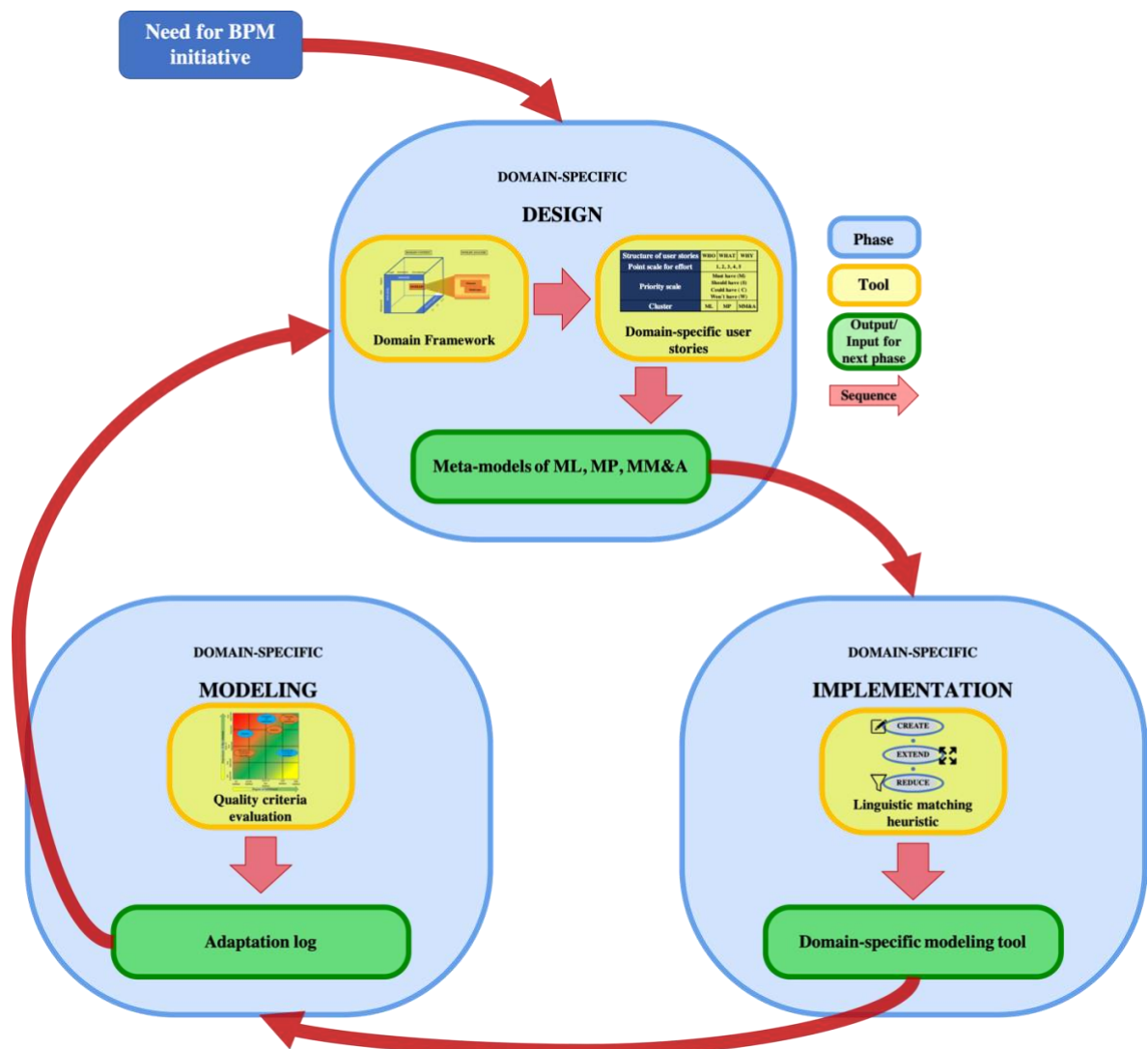


Figure 15 The Domain Integration Framework (DIF)

- own representation

The DIF as a lifecycle model is shown in Figure 15. As the three main phases of the framework, the *domain-specific design*, *domain-specific implementation*, and *domain-specific modeling* build the base. At the beginning of the DIF procedure model stands a need for a BPM initiative, which appears or is identified within a problem domain. Each phase of the DIF is accompanied by tools, which help to develop the output of the respective phase.

During domain-specific design, the problem domain is analyzed in a first step by the use of the DF described in chapter 4.1.1 in order to gain an understanding of the process boundaries and dependencies. A careful analysis and consideration of the requirements influences the usefulness, acceptance, and applicability of the modeling method to be developed. Therefore, the tool “user stories” is used to identify those requirements. Understanding the environment of the domain is vital to the success of the modeling method to be developed. The requirements reveal, what has to be included into the modeling method. The outputs of the design phase constitute the meta-models of the modeling language, modeling procedure, and modeling mechanisms and algorithms. Those serve as the input for the next phase.

The domain-specific implementation phase makes use of the “linguistic matching heuristic” as a tool. It requires the knowledge of the method engineer about existing BPMLs in order to build the DSBPML, as the word *heuristic* implies. Here, the previously designed meta-models are converted to a domain-specific modeling tool. The modeling tool as the output of this phase is ready-to-use for the next phase.

In the domain-specific modeling phase, the implemented tool is used, and models are created based on the previous design and implementation phases. Throughout the course of time, tools for quality evaluation need to be used in order to assess, whether the validity of the DS modeling tool is still given. This is done by a quality criteria assessment in the form of complexity of the model and usage of elements and model types. Needed changes or points for re-assessment are documented in the adaptation log.

In the following sections, the three phases of the DIF, the tools used within them, and the outputs of each phase are described in further detail.

4.1 Domain-specific Design

The goal of this chapter is to show, how requirements can be derived and a domain-specific modeling method (DSMM), consisting of the meta-models of the DS modeling language, DS modeling procedure, and DS mechanisms & algorithms, be designed. These constitute the output of the DS design phase. In the process of designing a domain-specific modeling method, a key intermediate deliverable is the modeling method specification [cf. 117]. Whether a domain-specific modeling language supports a certain application domain depends on the fulfillment of domain requirements, properties, constructs, and grammar. Keeping in mind the general structure of the ADOxx meta² model, all domain-specific concepts are declared as *class*, *relation class*, *attribute*, and *model type* [cf. 34]. Although the meta² model of ADOxx is seen as rigid, its high abstraction-level allows for enough flexibility to declare domain-specific concepts on the meta level.

4.1.1 Domain Framework (DF)

The first step of the design phase is to identify the external boundaries and the internal specifics of the application domain. In order to do so in a systematic way, the Domain Framework (DF) is introduced as a tool (see Figure 16).

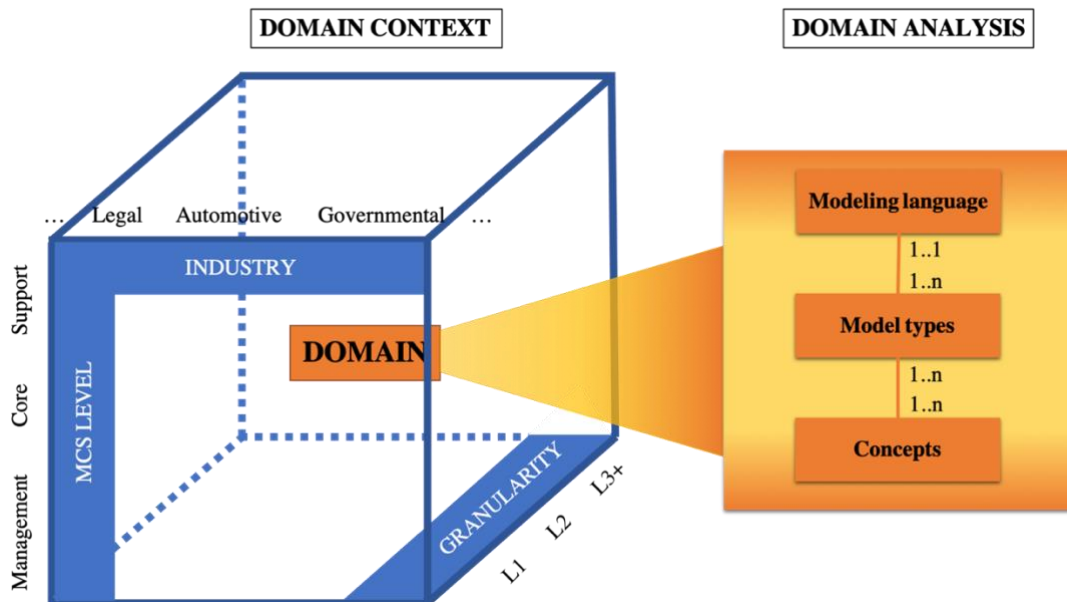


Figure 16 The Domain Framework (DF)

– own representation

The DF specifies two views of the domain, the *domain context* providing an external classification of a domain within a three-dimensional space and the *domain analysis* for the evaluation of internal domain-specifics. This integrated view allows for the identification of domain boundaries and is a useful step to gain an understanding of the external and internal particularities. The DF helps the modeling method engineer to systematically evaluate the area of application, where the developed method should be used. For the success of a modeling method in both, acceptance by the domain experts and the usefulness of the provided functionalities, this step of understanding the application domain and its connections to other domains is crucial.

4.1.1.1 Domain Context

The domain context's purpose is to reveal the external context of the domain to the modeling method engineer. The first dimension is the industry, in which the domain is located. The second dimension is the MCS-level, where the specific process-domain is classified into whether it is a **m**anagement, **c**ore, or a **s**upport process. The third dimension depicts the granularity-level (level of detail) of the domain-process to be designed. According to [32], the granularity is structured from level 0 to level 3+, where level 0 constitutes the process landscape. In the following, the three dimensions specifying the domain context are described in more detail and shall provide a foundation for the evaluation case study in chapter 5.

Figure 17 shows the domain context within the three interrelated dimensions *industry*, *MCS level*, and *granularity*.

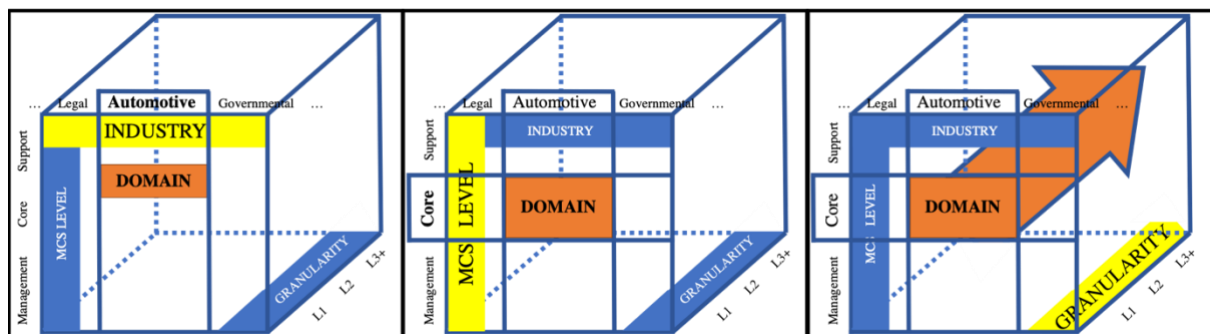


Figure 17 Correlations between the three dimensions of the DF

– own representation

The modeling method engineer should consider the respective application domain to have dependencies to each of those dimensions. The domain can be seen as a node within a network with links to other domains. This fact can be considered manifold throughout the modeling method design process: for collecting domain-specific requirements, for determining links to other organizations or departments, for modeling business processes and referencing them to related processes or models, to identify conceptual blocks for re-use, and more.

Industry

In chapter 2.2.2, several possibilities to structure industry domains were presented. For one, there exist process classification frameworks for general process representation, aiming to achieve a “one size fits all” approach. For the other, domain-specific categorization schemes have emerged and are constantly updated in order to integrate new process concepts⁶. Using such frameworks to identify the industry of the domain at hand helps to identify:

- **Domain-specific concepts**

Roles, necessary classes, and relations can be derived from the framework descriptions (e.g. [94] used the SCOR framework by [6] to develop the modeling method SIMchronization) to be included in the meta-model.

- **Hierarchical structure**

Most process frameworks are visualized in a hierarchical structure. This layout allows to identify possible sub-processes and related processes. Finding links to other domains or hierarchical levels helps to avoid errors in the course of modeling method design and business process modeling itself.

- **Re-usability opportunities**

Certain processes may appear throughout different industries. This provides a starting point for considerations, whether the modeling method or parts of it can be re-used in a different company. Examples of such re-usable processes are supply chain, maintenance, and finance, among others.

⁶ E.g. [6] APICS. (2018, 15.02.2019). *Frameworks*. Available: <http://www.apics.org/apics-for-business/frameworks> update their SCOR framework in order to integrate new global standards, qualification profiles, or best practices

In general, domain-specific process frameworks should be preferred over general process frameworks, as they reveal DS concepts in the most accurate way. However, if DS process frameworks are not available, general-purpose process frameworks can be used and combined with methods to extract DS concepts (one method is described in chapter 4.1.2 and later used in the case study).

MCS level

In order to design a target-oriented modeling method, the hierarchical integration of the focused business process has to be considered. This means, that depending on the specific hierarchical level, different aspects of the modeling method can be of importance. Different approaches towards structuring business processes in a hierarchical way were shown in chapter 2.1.1. The distinctions made in the DF hierarchy within this thesis are [cf. 32]:

- **Management process**

Every process contributing to the strategic direction of a company can be considered as a management process. This may include competitor analyses or budgeting activities.

- **Core process**

A core process accounts for direct value creation of a company. This may either be the production of certain goods or the offering of services, for which the customer is ready to pay. Most of the DSM languages analyzed within chapter 3 focus on core processes. However, core processes within one company might be a support processes of another, which creates opportunities for re-usability (e.g. accounting processes).

- **Support process**

Those processes facilitate the execution of core processes. Examples are HR management, IT, or technology development, which again are dependent on the specific industry and may be core processes in another.

In order to design a consistent modeling method, all hierarchical levels should be considered, and the processes structured accordingly. This helps to keep track of the process importance and may serve as a guideline in the modeling procedure, e.g. to start with the modeling of core processes first. As core processes build the foundation of the company's value creation, they should be modeled with priority.

Granularity level

The levels 0 to 3+ stand for an increasing granularity. Level 0 constitutes the process landscape and depicts all processes on a high abstraction level within one model. Level 1 is the value chain model of the domain-process. On level 2, the main processes are shown with their basic steps and decisions. Levels 3 and more⁷ depict the respective sub-processes, which reach until detailed task-level. As a general modeling guideline the modeling method engineer should assure to hide complexity and that the domain is well restricted [cf. 60]. Therefore, high-quality processes are structured in a hierarchical way to depict only relevant information of the specific level. The complexity is hidden by the use of sub-processes.

Between the different dimensions of the DF, dependencies do exist. Within certain industries, the core processes might be the same, whereas in other cases support processes (like e.g. IT) might be the core processes of other industries. The granularity dimension should be regarded in an integrated way. Moreover, it should be considered, which degree of detail is needed for the specific domain needs.

4.1.1.2 Domain Analysis

Whereas the domain context describes the external view of the domain, i.e. its location among other domains, the domain analysis looks at the intra-domain facets. The goal of this step is to gain insights into domain-specific concepts and to assess them in a structured way such that they can already be mapped to the respective meta-model concepts.

The goal of designing a modeling method is to allow for re-use and modularity, which contributes to the advantages of domain-specific modeling. The use of DSM rather than GPM is shown to improve the productivity, quality, the leverage of expertise, and economics within a company [cf. 60]. Therefore, domain analysis contributes to identifying domain-specific model types and concepts, which provide the opportunity for re-use. In the domain analysis, the modeling language consisting of classes and relation classes, and used model types is investigated.

Modeling language

As discussed in chapter 2.2.1 and analyzed in chapter 3, there exist different levels of domain-specificity. Some of the languages investigated are completely domain-specific,

⁷ Indicated by the + sign for each additional level

whereas others build on general-purpose modeling languages extended by domain-specific concepts. Considerations for creating a DSBPML can be made on syntactical, semantical, or notational level. For a certain domain, one domain-specific modeling method is created, which can consist of several domain-specific modeling languages.

Model types

During the investigation of DSMLs, the use of different model types resulting from requirements analyses was striking. DSMLs for similar domains often have similar model types and concepts. This fact shows a significant potential for the re-use of DSML fragments in the form of model types. Given a domain-specific modeling language, it can consist of several different model types.

Elements

To collect the elements used within the DSML, the meta-models of the respective language were investigated in chapter 3. The higher abstraction level of meta-models allows for an overview of the classes and relation classes used within the language and makes them comparable to other languages. An obstacle during the analysis was the absence or incompleteness of meta-models, or different approaches towards their representation. It is noteworthy to state, that DSMLs based on well-established meta-model standards were easier to understand and with less time-efforts. The meta-modeling approaches based on conceptual building blocks to represent different model types seem to be the most comprehensive. A model type consists of several concepts or elements. Certain concepts can be used by several model types at the same time.

4.1.2 Domain-specific User Stories

In chapter 3.3, requirements were identified as one of the influencing components of domain-specificity. The requirements for developing an applicable and relevant modeling method for the specific application domain are determined at design-time. As the analysis of the application domain (see chapter 4.1.1) answers the question *where*, the requirements help to specify the meta-models of the three building blocks of the modeling method (language, procedure, mechanisms & algorithms) in a next step. Understanding the requirements constitutes a necessity to build a modeling method, which will be accepted and is of value to the underlying need. Whereas in chapter 2.1.2 several approaches towards requirements

engineering were shown, the most suitable in the light of this thesis are *user stories*, which are complemented with the help of additional methods.

User stories are widely used in agile software development and offer several advantages [cf. 99]:

- **They deliver the highest value**

Formulated correctly, user stories help to deliver small and immediate deliverables. Compared to traditional approaches, where weeks or months are spent to develop one feature, user stories are focused on delivering working prototypes frequently.

- **They enhance collaboration**

Due to the minimalistic description of user needs on the story cards, those needs have to be grasped in the most exact way. This encourages teams to work closely with the customer and therefore avoids misunderstandings in the first place.

- **They allow for building blocks of the product**

The incremental and frequent delivery of new features guarantees a constant increase of the product value. A deliverable, which is built into the wrong direction can be corrected in a timely manner and non-conforming features easily omitted.

- **They increase transparency**

User stories, which are visible to everyone, enhance transparency within the team and towards the stakeholders. This transparency leads to higher trust in the product and depicts priorities and progress.

- **They enhance a shared understanding**

In contrast to traditional development approaches, where documents are handed around and not visible to all team members, user stories openly show what is expected and how the work is split between the team members. Therefore, the focus shifts from a detail orientation to an integrated view of the whole product.

- **They reduce risk**

The risk of delivering a non-working or not accepted product reduces considerably, as each sprint produces a working deliverable. The risk is further

reduced by the benefits named above, as e.g. collaboration with and transparency towards all stakeholders prevent misunderstandings.

Table 7 shows considerations for writing user stories based on [4]. The annotations on the right side of the table show, how these points are adapted to the use of domain-specific business process modelling in the light of this thesis.

Table 7 User stories for domain-specific design
– own representation

| Components of user stories based on [4] | Adaptation to DSBPM |
|---|---|
| Written by the stakeholders | This is an essential point when it comes to formulating requirements for a domain-specific business process modeling method, as acceptance by the users of the modeling method is essential. A focus on the customer needs by active collaboration helps to stay goal-oriented and design a valid solution. |
| “Keep it simple” is the policy when it comes to the tool-choice | Index cards are regarded as the best choice, as they have the right size to formulate a feature without adding too many details. Therefore, they guarantee to stay focused on the solution. |
| Different types of requirements can be addressed | Within this thesis, the requirements are first collected and in a second step clustered regarding their belonging. The clustering follows the three building blocks of a modeling method, namely modeling language, modeling procedure, and modeling mechanisms and algorithms. |
| Indication of the estimated size | The estimation of size within this thesis is done by the indication of working hours needed to implement the respective requirement. If for example a certain model type needs to be designed and the workload is estimated to be three hours, “3” is the indication of estimated size. |
| Indication of priority | Each requirement is rated with a priority. It is differentiated between a “must have”, “should have”, “could have”, and “won’t have” requirement. |

| | |
|--|--|
| Inclusion of a unique identifier (optional) | During the collection of user stories, a continuous number is added to each new requirement. |
|--|--|

Within the DIF, requirements collection by user stories is regarded as the best method, as it is user centric, deliverable-oriented, and based on agile methods. The requirements are first collected and then clustered according to the three building blocks of a modeling method. The results lead to the respective meta-model of each building block, including the domain-specific concepts. Whenever necessary, the use of supplementary methods like e.g. expert interviews can reveal additional concepts or identify missing requirements and should be considered in the requirements engineering process.

4.1.3 Meta-models of Modeling Method Building Blocks

The collection and clustering of requirements described in the previous chapter builds the base for designing the domain-specific modeling method. The clusters are tailored to the three building blocks of a modeling method and provide the domain-specific concepts for the meta-model of each block. Those meta-models are based on [67] and described in the following. The components, which make a modeling language domain-specific to a high extent (see chapter 3) are meta-model and model type. Within the meta-model, the elements represent the domain-specific concepts in the form of classes, relation classes, and attributes. The model types used were also identified to vary across different domains. Strictly speaking, they are part of the meta-model and can be integrated within it but are regarded separately due to their importance for domain-specificity.

For developing a DS modeling method, the modeling method engineer can consider three options. The first option is to build a new modeling method from scratch, which fulfills the requirements of the respective application domain as accurately as possible. The second option is to make use of already existing modeling methods with a fit as high as possible and to adapt them by adding or changing individual features. The third option constitutes the most time and resource saving one: already fitting modeling methods can be applied to the specific application domain without the need of change. This can be achieved as well by combining the meta-models of model types. As this option needs the least monetary and time resources, it constitutes the preferred one for companies aiming to select an appropriate BPM method. Figure 18 depicts the previously described options.

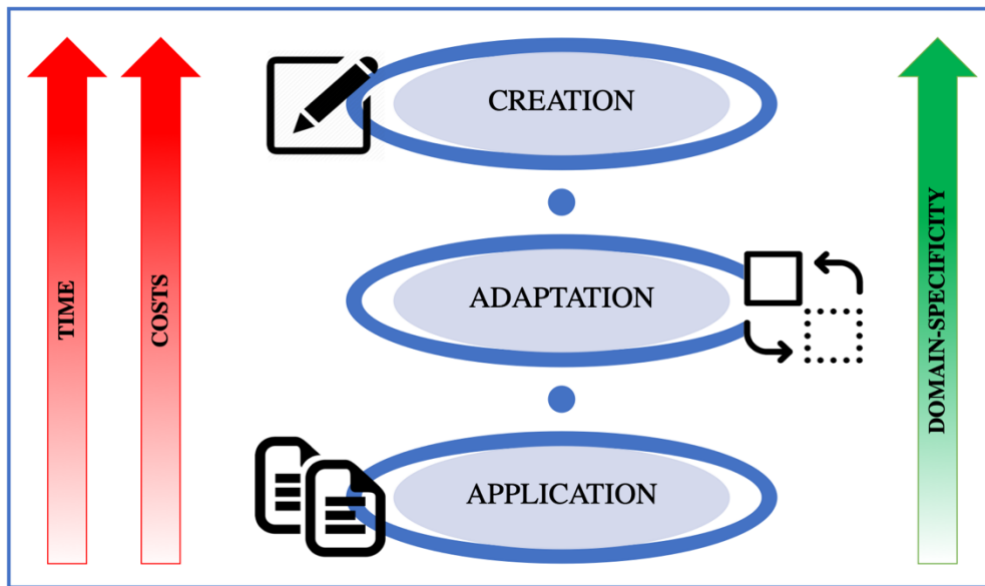


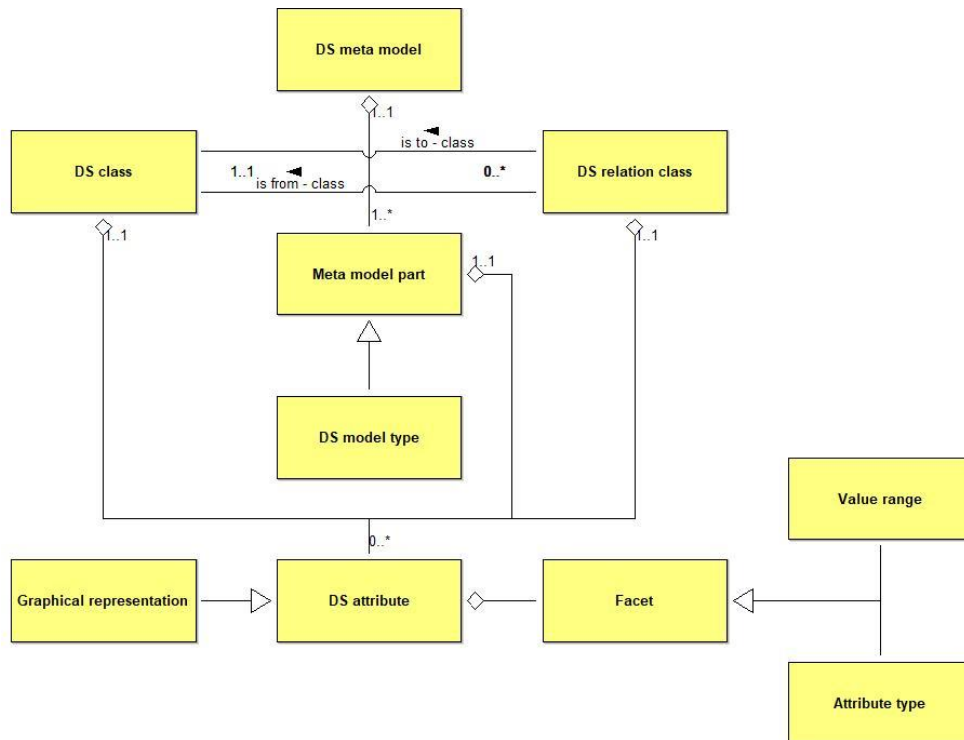
Figure 18 Options of modeling method design

– own representation, icons from [39] and [38]

There exists a trade-off between the three options described above, which the modeling method engineer has to balance in a suitable way. Ideally, as much reusable blocks of the meta-models shall be combined in order to achieve a high domain fit.

4.1.3.1 DIF Modeling Language

The general modeling language building block was described in chapter 2.4.1 and is shown here tailored to the needs of the DIF. The meta-model of the modeling language depicts all concepts used within the language and domain-specific concepts are integrated on the meta-level. Figure 19 shows the meta-model of the DIF modeling language including necessary components for domain-specificity.



*Figure 19 DIF modeling language
- own representation*

The domain-specific classes identified during the requirements engineering phase are integrated within the meta-model and their possible relations are described by relation classes. Ideally, the concepts are clustered within distinct model types in order to increase readability and comprehensibility. If not all domain-specific concepts are identified by the user stories, they can be identified from product specifications, employee knowledge, used vocabulary, architecture, existing products, patterns, expert knowledge, or used code, among others [cf. 60]. If none of these sources exist, the definition of sample applications is useful. The domain knowledge is located within the organization as well as individuals' memory.

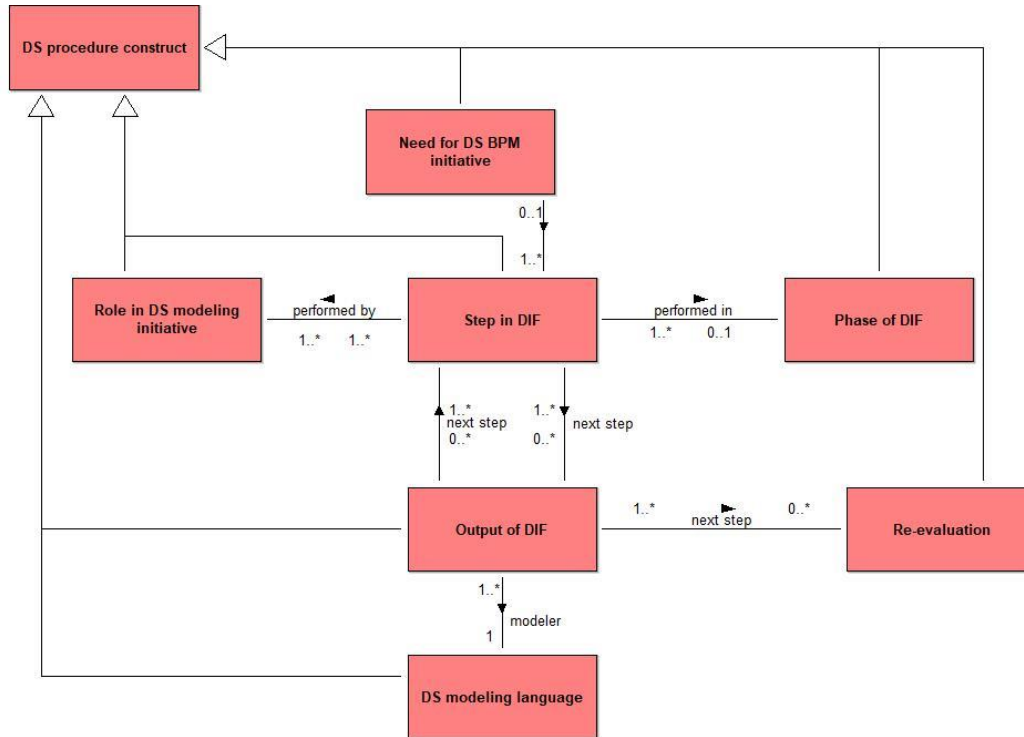
4.1.3.2 DIF Modeling Procedure

The modeling procedure defines the steps a process modeler should follow and the sequence of model types he or she should use. The modeling procedure can be seen as a systematic way to approach BPM and to avoid confusion or forget models. Several approaches to proceed with modeling can be utilized:

- From generic to specific by first identifying the high-level processes and diving into deeper detail with the help of sub-processes.
- From processes of a high hierarchical level to low-level processes.

- Starting from core processes, which limits the danger of forgetting crucial parts and makes sure, that all other processes are aligned with the most valuable BPs of the company.

The modeling procedure is also domain-specific, as different approaches towards modeling might be necessary within different environments. This is shown in below Figure 20, which constitutes the meta-model of the DIF procedure.



*Figure 20 DIF modeling procedure
- own representation*

4.1.3.3 DIF Modeling Mechanisms & Algorithms

The functionality needed within varying domains differs due to diverse key performance indicators (KPIs), which are of importance within certain industries and process types. Examples for model functionality are simulation, transformation, evaluation, and visualization [cf. 57].

Identifying the needed mechanisms and algorithms requires a good understanding of the BPM goal and the domain. A generic to specific functionality description of ADOxx is provided in [57]:

- SQL generation mechanism and token-based simulations
- Path analysis, workload assessment, reasoning mechanisms
- Machine interpretable semantics

Resulting from the requirements engineering phase of the DIF, functionality needs in the form of mechanisms and algorithms are extracted. Moreover, they should be described on a meta-level in order to depict the dependencies of inputs and outputs and be understandable to the model users. The domain-specific meta-model of the DIF is shown in Figure 21.

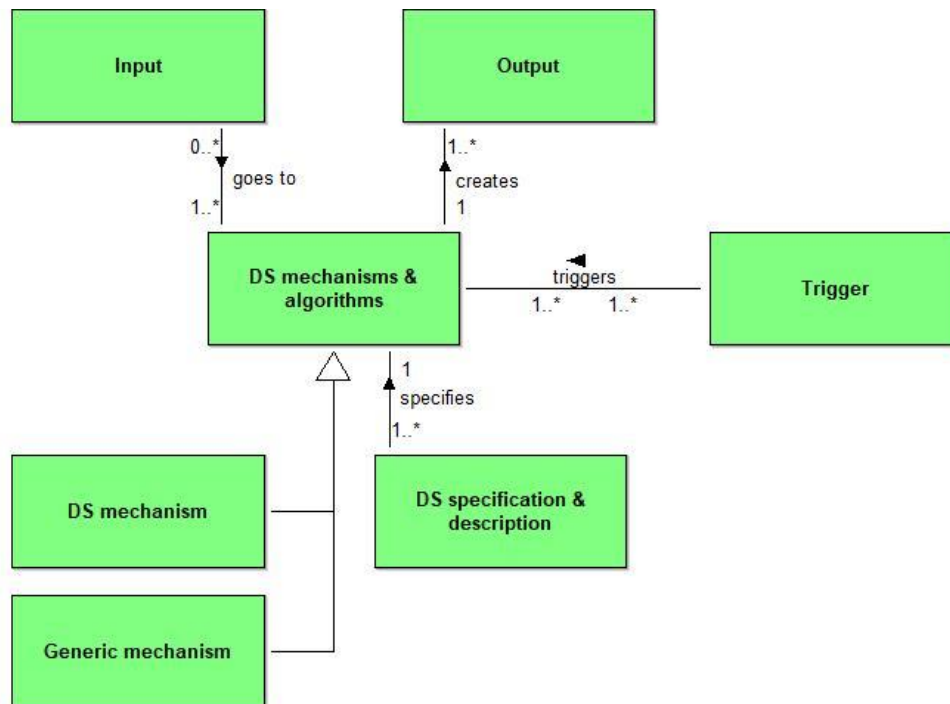


Figure 21 DIF modeling mechanisms & algorithms

- own representation

4.2 Domain-specific Implementation

In the previous section, the domain-specific design phase was shown including the domain context and analysis by the use of the DF and the creation of a domain-specific modeling method on the meta-layer by the use of DS user stories. The outputs of the previous phase, the domain-specific meta-models, constitute the input for the following domain-specific implementation phase.

4.2.1 Linguistic Matching Heuristic

The output of the previous chapter is a to-be domain-specific design including all needed information about concepts and functionalities. Those are captured within the meta-models and enable comparability to other meta-models due to the higher abstraction level. Within this thesis, a “linguistic matching heuristic” is used to develop a ready-to-use modeling tool, which constitutes the output of the implementation phase of the DIF. Expressed by the word “linguistic”, the focus lies on the language meta-model with its classes and relation classes. The goal is to identify synergies between the domain-specific modeling method and other, already existing modeling methods. The different matching possibilities are shown in Figure 22 and explained in the following.

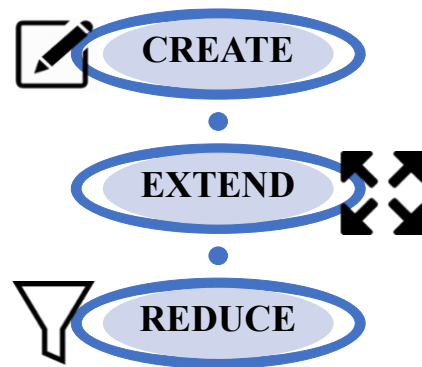


Figure 22 Matching possibilities

– own representation, icons from [39], [83], and [49]

The creation of a new modeling tool requires most time and money resources. The tool is developed from scratch without using existing concepts. This involves a high effort as no synergies are used. The second possibility is to extend an already existing modeling tool, which contains all or almost all concepts required. Those existing meta-model elements are then extended by the elements needed on top of the current solution in order to fulfill all requirements. A third option is the reduction of existing modeling tools by the elements not needed within the domain-specific solution. An already existing implementation could include all needed elements but refer to a much wider domain. If the domain under investigation is smaller, unnecessary elements have to be removed due to complexity and redundancy reasons. In practice, a mixture of all three matching possibilities is required. The process of domain-specific implementation can be described as a heuristic, because it follows the previously described thought-pattern as a combination of modeling tool creation, extension, and reduction. Another possibility is to use algorithmic pattern matching, which

exceeds the focus of this thesis. It has to be noted, that the heuristic approach requires prior knowledge of existing GP BPMLs and DSBPMLs in order to achieve a higher degree of re-usability.

4.2.2 Domain-specific Modeling Tool

After building on the knowledge base about GPMLs and DSMLs and matching model types, concepts, and functionalities, the domain-specific modeling tool has to be implemented. As described in the previous chapters, the DIF within this thesis relies on the ADOxx meta-modeling platform and its provided meta² model [cf. 34]. The domain-specific concepts identified within the design-phase are mapped to the elements of the ADOxx meta² model. The major elements are model type, class, relation class, and attribute. For the implementation, the extensive ADOxx documentation and tutorial resources provide the instructions. The output of the domain-specific implementation phase is the ready-to-use domain-specific modeling tool, which constitutes the input for the next step, the domain-specific modeling phase.

4.3 Domain-specific Modeling

During the domain-specific design phase of the DIF, a collection and clustering of requirements in the form of user stories led to the conceptualization of a domain-specific modeling method on the meta level. This step constitutes a valuable deliverable, as it assures visibility to the user groups and serves as a means of discussion and further refinement. Users of the DSBPM solution should receive a training and a template specification as a guideline, as they are usually non-programmers but domain experts. The modeling method meta-models can serve as such a training instrument.

After finishing the DS implementation, the fit between the modeling method and the respective target domain is the highest. This fit can get lost, as requirements might show to be implemented with an incorrect priority or change over time, or domain-specific concepts need to be adjusted in the meta-models. This might lead to a loss in acceptance of the modeling method and discontent among the users of the DSBPM tool. Therefore, regular checks need to be undertaken by a dedicated team in order to regenerate the fit between the modeling method and the domain. This can be done by the mechanisms described in the following chapters.

4.3.1 Quality Criteria Evaluation

After the initial design of the modeling method and implementation of the modeling tool, the fit to the application domain is high. However, the primarily established fit gets lost in the course of time and changing requirements. Therefore, regular re-evaluation is an essential part of the DIF. This re-evaluation is made systematically by the assessment of the criteria, stated in Table 8.

For the usage of a domain-specific modeling method it is important to regularly review its applicability. The domain-specific requirements as well as the domain itself may change over time, so mechanisms for regular quality checks have to be implemented into the framework. How the quality of a model is perceived relates to the degree of fulfillment of the underlying modeling requirements [cf. 54]. Following quality criteria build the foundation for regular re-evaluation checks within the DIF:

Table 8 Quality criteria collected
– own representation

| Quality criterion | Description |
|---------------------------|--|
| Requirement re-evaluation | In regular, predefined intervals, a re-evaluation of requirements needs to be done in order to assure the topicality of the modeling method. Therefore, a dedicated team ideally consisting of the modeling method engineer, the tool developer, and the business process modeler (user) should come together and take some time to re-assess existing requirements and identify new ones. |
| Usage of elements | Elements, which are not used regularly or are used ambiguously need to be revised or deleted. This assures the relevancy and simplicity of the meta-models and keeps acceptance of the model users high. |
| Usage of model types | Infrequent use of certain model types might indicate, that they are obsolete. A re-investigation based on the underlying requirements should be conducted to find an alternative integration of elements used within the model type. |
| Consistent use | If it becomes visible that the modeling method is used incorrectly over time, measures need to be taken in order to find out the reasons for this |

| | |
|--------------|--|
| | inconsistency. Adjustments of the modeling method might need to be undertaken in order to avoid misleading use of the modeling method. |
| Complexity | A measure for evaluating model complexity is provided by [71] and described below. Complexity can be a metric, which explains bad user acceptance or ambiguous method use. In general, complexity should be kept as low as possible and as high as necessary, to fulfill the specific domain requirements. |
| Service team | A dedicated service team responsible for the domain-specific modeling method is essential for long-term success. Possible issues with the DSBPM method can be sent to the service team, questions answered, and trainings coordinated. |

As indicated in Table 8, [71] propose metrics to assess the complexity of modeling method components. The authors distinguish between the complexity of interfaces, elements, and properties. Properties correspond to attributes in ADOxx and the modified formulas are shown below.

$$C_{interface} = n_{relations} + n_{constraints}$$

$$C_{element} = n_{element}$$

$$C_{attributes} = n_{attributes}$$

$$C_{overall} = C_{interface} + C_{element} + C_{attributes}$$

Figure 23 Metrics for complexity assessment

- based on [71]

Frank and Van Laak [37] propose a framework for quality-evaluation of modeling languages. Within their work, they distinguish regarding the degree of formality between informal, semi-formal, and formal modeling languages. The use of each characteristic depends on the domain and might differ. There exists a trade-off between the simplicity of a modeling language and its level of detail. In their approach, [37] suggest the evaluation of criteria in the light of the pursued modeling purpose. The evaluation is undertaken on the one hand by making qualitative statements about the importance of the respective criterion (very important, important, semi-important, less important, unimportant), and on the other

hand by assessing its degree of fulfillment (fully fulfilled, well fulfilled, semi-fulfilled, badly fulfilled, not fulfilled). In order to visualize the criteria, the authors propose the portfolio diagram. Figure 24 depicts an adaptation of this portfolio diagram for the use within this thesis.

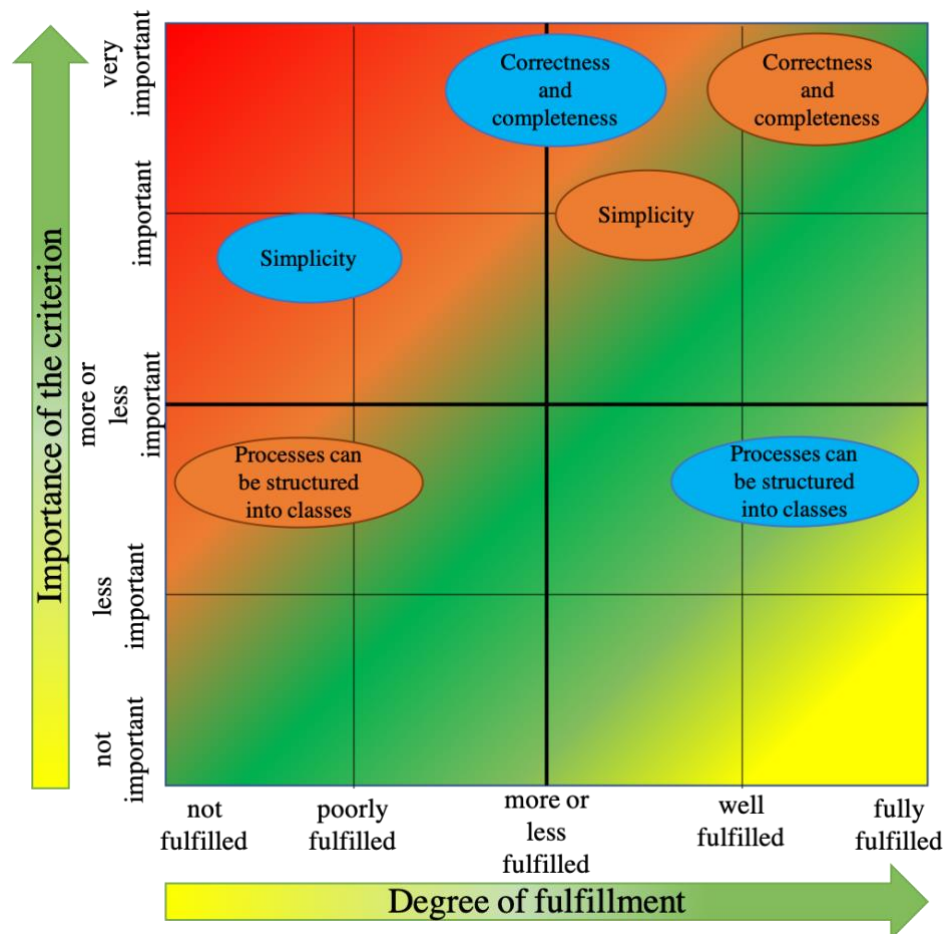


Figure 24 Portfolio-diagram for the evaluation of modeling language criteria
– own representation based on [37]

The ellipses in Figure 24 depict the range of criteria fulfillment, e.g. the blue ellipse on the right side of the figure shows that the criterion is fulfilled between *good* and *fully*. The more an ellipse is located in within the red area in the upper left corner, the more the criterion is a knock-out criterion, which makes the modeling language unusable. The different colors of the ellipses stand for different modeling languages.

4.3.2 Adaptation Log

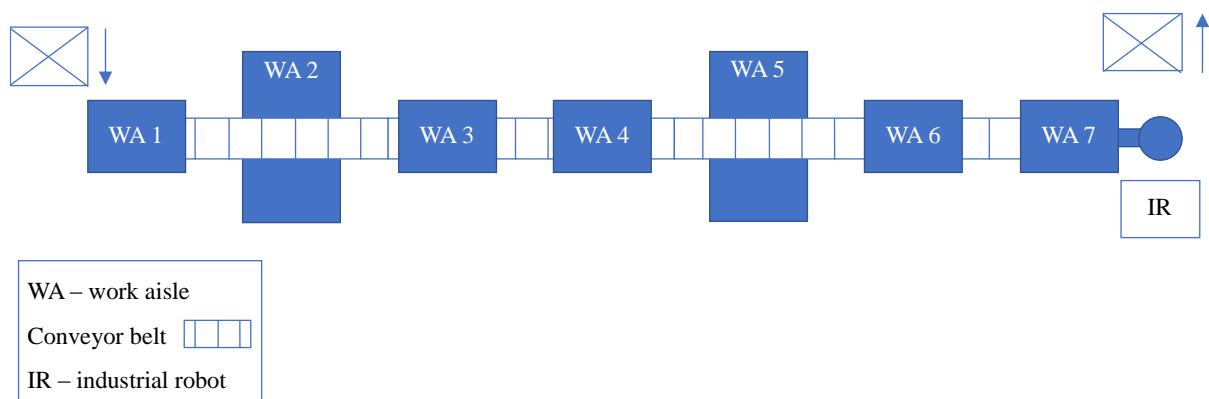
During the use of the DSMM, the requirements defined during design-time are likely to change, so can the problem domain itself. But not only changing or new requirements lead to a need for making adjustments in the DS design - also factors like an improved knowledge of the specific domain, improved understanding while implementing and using the solution, as well as feedback from other users of the DSMM can induce this need for change.

In order to confront the issue of changing requirements, the service team mentioned in the previous chapter plays an essential role. Such a team should be available for the users of the modeling method and be equipped with sufficient authority and expertise to carry out changes. The output of the regularly scheduled quality assessment is the adaptation log, which serves as an instrument for change request documentation and as a check list for conducting the changes on the domain-specific meta-models and the domain-specific modeling tool.

5 Proof of Concept – Automotive Assembly Line Case Study

Following the research approach described in chapter 1.3, the domain under investigation is the automotive assembly line. Due to missing information about the detailed business processes within a real automotive assembly line, several different sources about automobile manufacturing in general and automobile assembly specifically were assessed. For the assembly line and general production processes, [92] deliver the base for designing the domain-specific assembly process. The goal is to go through all phases of the DIF and design a modeling method, which is implemented in form of a modeling tool, the automotive assembly line domain-specific language – AAL-DSL.

The final assembly in the automotive industry is in most of the cases just-in-sequence (JIS), e.g. the car seats are provided in the correct order regarding the assembly steps for the respective car type. The JIS as well as the just-in-time (JIT) concepts therefore require a strong connection between the supplier and the manufacturer. The supplier plants are therefore often within a distance of 30 km to the manufacturer [cf. 73, p.343]. Figure 25 shows the concept of a line assembly, which is the typical form present in the automotive industry.



*Figure 25 Line assembly
– own representation based on [73]*

Typically, several work stations are aligned consecutively and are connected by a material flow system. The core unit enters the system and gets attachments at every station of the flow. Especially in the automotive industry, automation plays a central role. The work on modern assembly lines is augmented more and more by the use of robots.

Following the approach described within the previous chapters, the case study on the automotive assembly line is structured as follows. As described in chapter 4.1, the first step is to gain a profound understanding of the domain. Therefore, the domain context as well as the domain analysis are conducted to identify process dependencies and boundaries, and to gain an understanding of model types, concepts, and functionality, which might be needed. Within the domain-specific design phase, domain-specific requirements are collected, clustered, and analyzed, building the base for the domain-specific modeling method. The second part (see chapter 4.2) constitutes the domain-specific implementation, where the meta-models identified within the previous step, are implemented within the ADOxx Development Environment. Here, the dynamic as well as the static libraries build the foundation to provide needed model functionality. In the domain-specific modeling phase (see chapter 4.3), the implemented modeling tool as well as the defined functionality are used by the domain experts. Process models and other needed model types are modeled, provided with data, and evaluated and simulated on a regular basis.

5.1 Domain-specific Design

Within this chapter, the domain of the automotive assembly line is analyzed as described in chapter 4.1. This constitutes the first phase of the DIF and provides the necessary insights for the domain-specific implementation phase. The goal is to extract all needed model types, concepts, and functionality and specify them in meta-models of the modeling method building blocks.

5.1.1 Domain Specification

Taking the example of an automotive assembly line, a domain-specific scenario is described in the following. The final assembly line in the automotive industry still includes highly manual tasks. A typical production layout for the automotive domain is a flow production with just-in-sequence delivery (JIS) [cf. 92]. The assembly process constitutes a core-process of every car manufacturer as it adds significant value to its business. The integration of this application domain within the DF can be seen in Figure 26 and is described in the following.

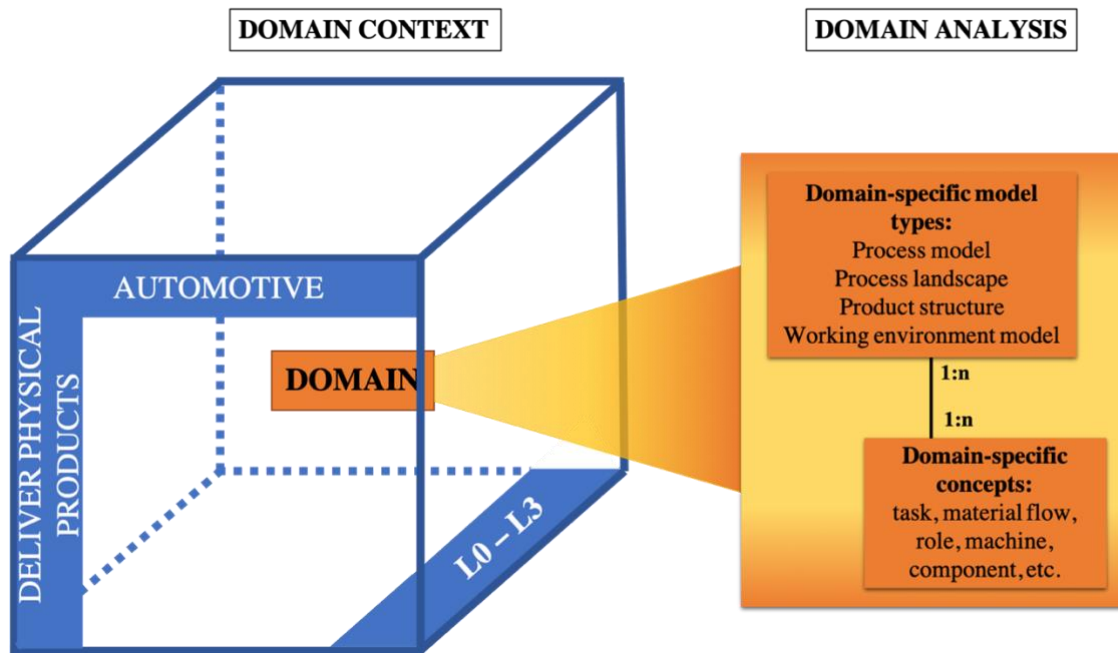


Figure 26 Domain Framework (DF) of the AAL-DSL
– own representation

The above figure summarizes the output of the domain specification after the external and internal analysis of the domain. The single steps are described in more detail within the following chapters.

5.1.1.1 Domain Context

For the domain-specification, the Automotive Process Classification Framework® by [8] is used. It uses domain-specific concepts of the automotive industry and provides information about management, core, and support processes specific to the automotive sector.

Industry

The industry can be classified as automotive, as the name of the application domain already suggests. Its main business goal is the production of different car models, which consist of varying combinations of components and raw materials. Figure 27 shows the industry specification within the DF.

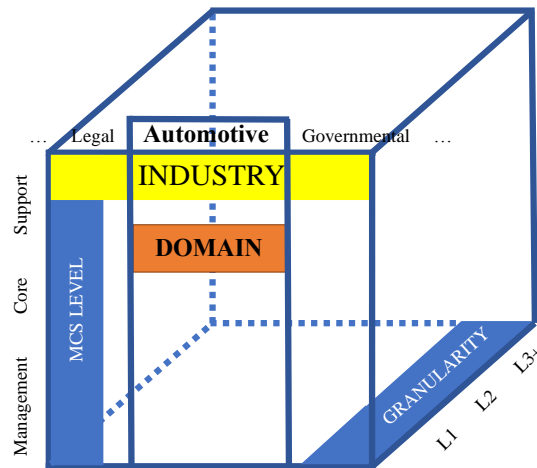


Figure 27 DF industry specification of the AAL-DSL
– own representation

MCS-level

The automobile assembly, which is the domain under investigation, constitutes a core process of automobile manufacturers. In Figure 28, this fact is depicted.

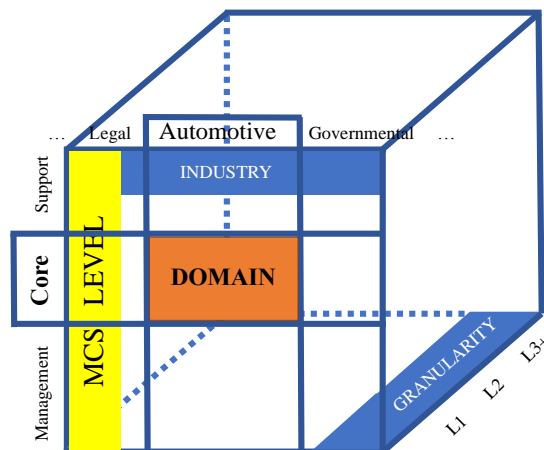
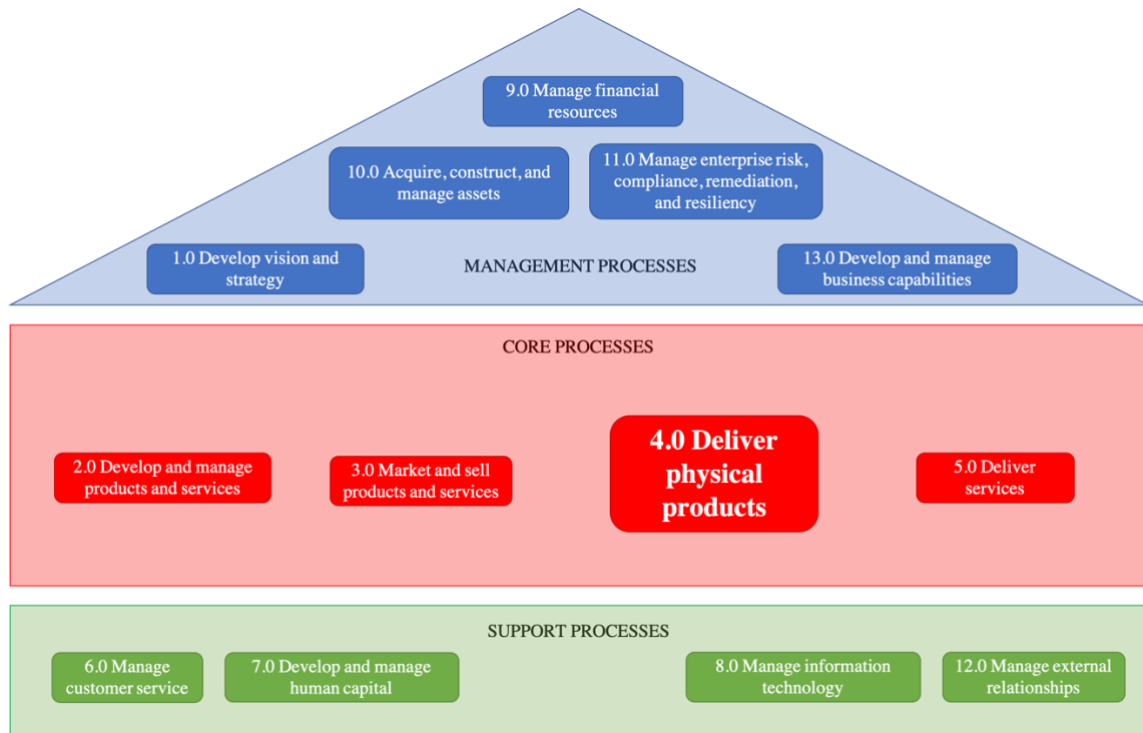


Figure 28 DF MCS level specification of the AAL-DSL
– own representation

Figure 29 resembles a collection of management, core, and support processes, which were identified as relevant throughout the investigation of the automotive domain. The processes are derived from the domain-specific PCF® for the automotive industry. Whereas the PCF® only categorizes core and support processes, a further distinction is made to also highlight the management processes.



*Figure 29 The process architecture of the AAL-DSL
– own representation*

The main process under investigation, the assembly process, is included within point 4.0 *Deliver physical products*, depicted in bold letters within the figure.

Granularity-level

Including the level 0 process landscape, 4 levels of granularity are needed in order to depict the assembly process. The arrow shows the integrated view of all granularity levels, which is achieved by the usage of referenced subprocesses. This is shown in Figure 30.

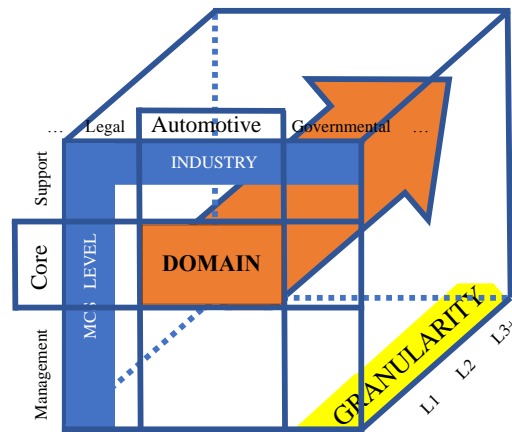


Figure 30 DF granularity specification of the AAL-DSL
- own representation

Figure 31 depicts the above described granularity of the assembly process in a more detailed view. This figure constitutes a first schematic overview of the assembly line process, which is investigated in greater detail within the further course of this chapter.

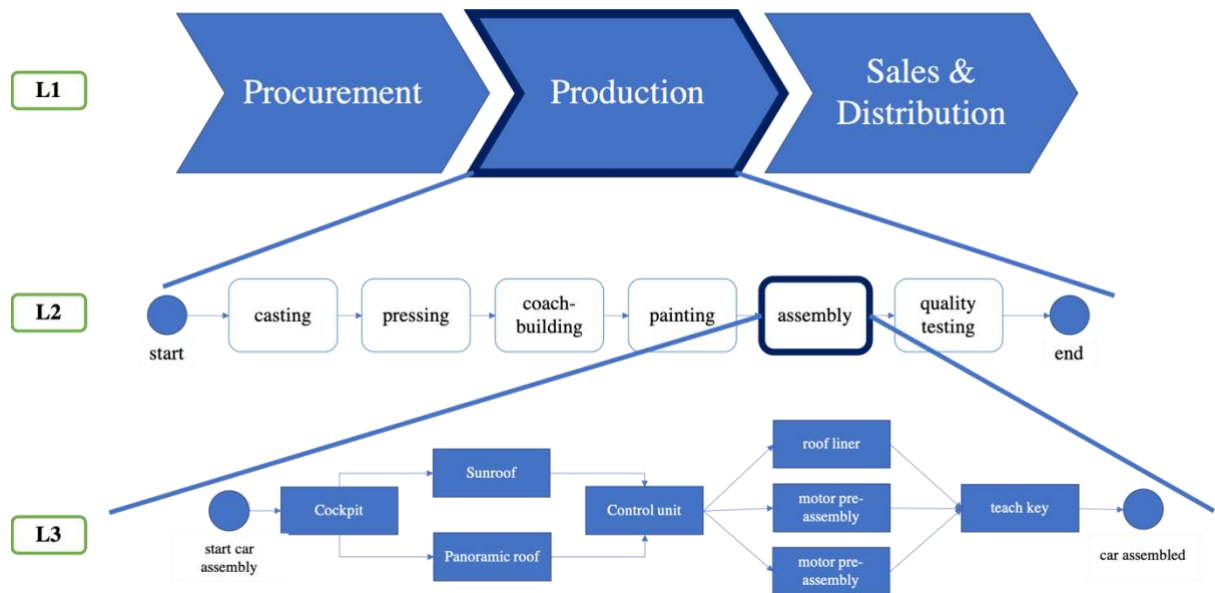


Figure 31 Integrated granularity-levels of the AAL-DSL
- own representation

The process landscape constitutes level 0. The referenced process of *4.0 Deliver physical products* is the process on level 1. Level 2 shows the major production steps, the automobile undergoes. The detailed assembly process itself is situated on level 3 within the process hierarchy as one of the sub-processes of the production of automobiles.

5.1.1.2 Domain Analysis

As described in chapter 4.1.1.2, the domain analysis aims at investigating the internal specificities of the domain. The main goal is to collect needed domain-specific constructs within the process, to identify appropriate model types and to find suitable model functionality. When it comes to the assembly line process of a car manufacturer, following process characteristics are identified in a first brainstorming session.

Model types

Following the description of the assembly line process and the structure defined by the PCF®, an idea about needed model types can be formed. In order to design a well-understandable modeling method for this respective domain, a process model is indispensable. The BPMN2.0 language could be considered a good starting point, but a mere sequence flow not sufficient, as elements representing material flow are needed as well. BPMN2.0 also offers the possibility to form sub-processes, which enables the reduction of complexity within the extensive processes of automobile production.

Another essential model type is the process landscape, in which management, core, and support processes are depicted as an overview. This should be enhanced further by a working environment model type in order to depict the departments as well as roles and performers within the company.

As automobile manufacturers usually assemble several different products, a product structure model is needed. Here, the different car models are structured by their peculiar components and differences and similarities are shown.

The previously described model types, which are needed to depict the automotive assembly line domain, are summarized in following Table 9.

Table 9 Summary of model types for the AAL-DSL
– own representation

| Need derived from domain analysis | Collected model types for the automotive assembly line domain | ADOxx applicable library |
|---|--|---------------------------------|
| Need for modeling the sequence of activities within | Process model | Dynamic library |

| | | |
|--|---------------------------|-----------------|
| the automotive assembly process | | |
| Need to have an overview of all processes and their relations | Process landscape | Dynamic library |
| Need to view related departments and roles involved in the processes | Working environment model | Static library |
| Need to see the components, the product consists of | Product structure model | Dynamic library |

Elements

Stakeholders are an integral part of the assembly line process as well as its interconnected processes. External partners constitute for example suppliers of the car components and the clients of the automobile manufacturer. Important internal roles are the assembly line workers, who manufacture the product, and the quality managers checking for correctness and quality criteria.

For handling, transportation, and carrying out difficult tasks, machines and tools are used within the process. For one, robots assist humans and carry out activities like painting or screwing. For the other, a handling unit is used to transport the heavy product across the factory.

Also, elements representing the evolving car and its components including the storage they are kept in are needed. A relation dedicated to material flows increases the understandability of the domain-specific modeling language.

Last but not least, different types of locations are helpful to describe the assembly line process, which is done in the form of swim lanes. Departments involved within the process can be depicted, e.g. the procurement, quality, and logistics department.

Throughout the first domain description, the elements summarized in Table 10 are identified. In the third column, the mapping to the concepts inherent in the ADOxx meta² model is depicted.

Table 10 Elements identified by the DF
– own representation

| Concept identified | Description | Mapping to ADOxx meta² model | Mapping to model type(s) |
|---------------------------|---|--|----------------------------------|
| Task | Tasks resemble the sequential steps taken within the automotive assembly line process, sub-class of “__Activity__” | Class | Process model |
| Subsequent | The order, in which the tasks are completed, already provided in the ADOxx meta-model | Relation class | Process model |
| Department | Part of the organization, which is responsible for a specific area and to which employees belong to, sub-class of “__S_group__” | Class | Working environment model |
| Performer | Is an employee fulfilling a specific type of task, sub-class of “__S_person__” | Class | Working environment model |
| Role | Is a collection of employees fulfilling a specific type of task, sub-class of “__S_group__” | Class | Working environment model |
| External partner | Is the generic term for business partners outside the organization | Abstract class | Process landscape |
| Robot | Different from a human role, but carries out certain tasks within the process | Class | Process model |
| Storage | A place where raw materials, intermediate, and final products are stored | Class | Process model, product structure |
| Machine | Needed to conduct tasks beyond human capabilities | Class | Process model |
| Tool | Used by a role to fulfill a certain task | Class | Process model |

| | | | |
|------------------|---|----------------|--|
| Material flow | Sequence in which components and raw materials are added to the process | Relation class | Process model |
| Information flow | Shows the flow of information exchange within the process | Relation class | Process model |
| Swim lane | Helps to differentiate between different areas or departments, where tasks take place, also for general structure | Class | Process model, process landscape |
| Product | Is the final product, i.e. the finished car at the end of the process | Class | Product structure model |
| Component | E.g. car body, headlights, motor | Class | Product structure model, process model |
| Raw material | E.g. paint or screws | Class | Product structure model, process model |

5.1.2 Domain-specific Requirements

In chapter 5.1.1, the foundation for the following section was built by specifying the domain context and conducting the domain analysis. As the collection of elements and model types within the previous chapter was done by the description of the assembly line process in a brainstorming manner, the next step is a profound analysis of requirements. From those requirements, the design of the modeling method is derived by specifying the meta-models of the modeling language, procedure, and mechanisms and algorithms.

In the way described in chapter 4.1.2, requirements are collected by formulating user stories. The advantages comprise their stakeholder focus, shortness, and the possibility to quantify and prioritize them. The requirements collected for the purpose of this case study can be seen in Table 12. Table 11 gives an overview of the concepts used to structure, quantify, prioritize, and cluster the user stories.

Table 11 Key for requirements assessment
– own representation

| Structure of user stories | WHO | WHAT | WHY |
|-----------------------------------|---|----------------------------|--|
| Point scale for effort (in hours) | 1, 2, 3, 4, 5 | | |
| Priority scale | Must have (M), Should have (S), Could have (C), Won't have (W) | | |
| Cluster | Modeling language (ML) | Modeling procedure (MP) | Modeling mechanisms & algorithms (MM&A) |

The basic structure for the user-story formulation is the specification of requirements by the words “who”, “what”, and “why”. User stories are explained more profoundly in chapter 4.1.2. The effort of implementing the respective requirement is quantified by numbers from 1 to 5, corresponding to the estimated duration in hours. The priority scale helps to rank the requirements according to their importance for the project. The requirements are clustered regarding their belonging to the modeling language (ML), modeling procedure (MP), and modeling mechanisms and algorithms (MM&A) building block. Requirements, which do not specifically belong to one block and are of a general nature, are described by MM for “Modeling Method”.

Table 12 Collected, assessed, and clustered requirements
– own representation

| ID | Requirement / Story | Effort | Priority | Cluster |
|----|---|--------|----------|---------|
| 1 | As a business process modeler, I want to use BPMN2.0, because I am used to the language and it is easy | 2 | S | ML |
| 2 | As a business process modeler, I want to have an adaptable modeling tool, because requirements are likely to change over time | 1 | M | MM |
| 3 | As the production manager, I want to have an overview of the processes in an understandable way, because I need to react fast | 3 | C | MP, MA |

| | | | | |
|----|---|---|---|------|
| 4 | As the production manager, I want to have relevant KPIs integrated, because I need a means for fast decision-making | 4 | M | MM&A |
| 5 | As a domain expert, I want the modeling method to have different abstraction-levels, so that complexity is reduced | 2 | M | ML |
| 6 | As a domain expert, I want to see the process flow of the assembly line, so that the sequence of steps is clear | 3 | M | ML |
| 7 | As a domain expert, I want to see the material flow of the assembly line, so that it is clear, which components are needed in which part of the process | 4 | M | ML |
| 8 | As a business process modeler, I want to have an instruction for implementing the modeling method, so that nothing is forgotten within the procedure | 3 | M | MP |
| 9 | As the production manager, I want to have a process simulation mechanism, so that I can make better decisions | 5 | S | MM&A |
| 10 | As a quality engineer, I want to have a product and component overview, so that I can assess product quality more easily | 3 | C | ML |
| 11 | As a business process modeler, I want to work in phases with milestones, so that the progress is visible to everyone | 1 | S | MP |
| 12 | As a domain expert, I want to have a language representing the concepts I know, so that I do not use it ambiguously | 2 | S | ML |
| 13 | As a domain expert, I want to specify quantities and times, so that I can run simulations based on that information | 3 | M | MM&A |
| 14 | As a method engineer, I want an intuitive platform, so that I can manage the modeling method | 1 | M | MM |
| 15 | As a business process modeler, I want to know who is responsible for modeling which parts of the process, so that the workload is separated equally | 1 | W | MP |

| | | | | |
|----|---|---|---|------|
| 16 | As a method engineer, I want a flexible platform, so that I can adapt the modeling method to changes | 1 | M | MM |
| 17 | As a business process modeler, I do not want unnecessary elements, because they are out of scope and confusing | 2 | S | MM |
| 18 | As a modeling method engineer, I do not want to design a modeling method from scratch but build on a fitting solution, so that implementation efforts stay as low as possible | 3 | S | MM |
| 19 | As a modeling method engineer, I want to be able to adjust attributes of different classes easily, so that I do not need to re-implement the whole class | 1 | S | MM |
| 20 | As a business process modeler, I want to select between different modeling views, so that I have a predefined set of symbols for different purposes | 2 | S | ML |
| 21 | As a domain expert, I want to have a accessible customer service, so that I can get my questions answered fast | 4 | M | MP |
| 22 | As a business process modeler, I want to get regular trainings, so that I use the modeling method correctly | 3 | S | MP |
| 23 | As a production manager, I want to assign roles to tasks, so that I can oversee responsibilities and employee capacities | 4 | M | ML |
| 24 | As a domain expert, I want an appropriate graphical representation, so that the domain-specific concepts I know are used | 5 | S | ML |
| 25 | As a method engineer, I want the possibility to hierarchically structure classes, so that several sub-classes can belong to one class | 4 | M | ML |
| 26 | As a production manager, I want to specify pre- and post-conditions, so that special mechanisms and algorithms are triggered in certain situations | 4 | C | MM&A |

| | | | | |
|----|--|---|---|------|
| 27 | As a business process modeler, I want to be able to include text annotations, so that I can explain parts of the models in natural language to the model users | 2 | M | MM&A |
| 28 | As a business process modeler, I want the modeling method to be intuitive and relevant, so that it is accepted by the domain experts | 2 | M | MM |
| 29 | As a process analyst, I want to be able to conduct a path analysis | 5 | M | MM&A |
| 30 | As a process analyst, I want to be able to conduct a capacity analysis | 5 | M | MM&A |

The above table gives an overview of the aspects, which need to be implemented or considered in the modeling method design phase. They serve as a foundation for the modeling method design. The following table gives an overview of the concepts, which were identified by the user stories in addition to the previous elements from Table 10.

Table 13 Elements identified by the user stories
– own representation

| Concept identified | Description | Mapping to ADOxx meta ² model | Mapping to model type(s) |
|--------------------|--|--|--------------------------|
| Value block | Depicts the value chain in the process landscape | Class | Process landscape |
| Supplier | A sub-class of “external partner” | Class | Process landscape |
| Client | A sub-class of “external partner” | Class | Process landscape |
| Parallel gateway | A sub-class of “__Parallelity__” to show parallel sequences of tasks | Class | Process model |
| Exclusive gateway | A sub-class of “__Decision__” to show exclusive sequences of tasks | Class | Process model |

| | | | |
|------------------|---|----------------|---------------------------|
| Start | A sub-class of “__Start__”, showing the beginning of the process | Class | Process model |
| End | A sub-class of “__D_End__”, showing the end of the process | Class | Process model |
| Merging gateway | A sub-class of “__Merging__”, showing the merging after an exclusive or parallel process flow | Class | Process model |
| Random generator | A sub-class of “__D_random_generator__”, to describe the distribution probability of exclusive gateways | Class | Process model |
| Variable | A sub-class of “__D_variable__”, to describe the type and scope of the distribution | Class | Process model |
| Tool flow | Shows, which tools and machines contribute to which tasks | Relation class | Process model |
| Association | Shows connections between process steps and external partners | Relation class | Process landscape |
| Sets variable | Needed for variable setting | Relation class | Process landscape |
| Sets | Needed for setting the random generator | Relation class | Process landscape |
| Is component of | To show, which components and raw materials a final product consists of | Relation class | Product structure |
| Has role | To show the connection between roles and performers | Relation class | Working environment model |
| Belongs to | To show the connection between performers and departments | Relation class | Working environment model |

5.1.3 Domain-specific Modeling Method

The modeling method for domain-specific business process modeling of the automotive assembly line of a car manufacturer is shown and explained within this chapter. It consists

of the meta-models of the modeling method building blocks *modeling language*, *modeling procedure*, and *modeling mechanisms & algorithms*. It is based on the analysis techniques from above, which helped to derive insights on how the method needs to be built and which components are essential for its acceptance by the domain experts as its users. As outlined in chapter 2.4, a highly qualitative modeling method does not only consist of a modeling language. It is complemented with functionality in the form of mechanisms and algorithms on the one hand and its implementation steps are specified in the modeling procedure on the other hand. The meta-models for each building block are shown in the following.

5.1.3.1 Modeling Language

The meta-model elements for the modeling language building block were derived from the Domain Framework as well as the requirements collection and analysis phase. The task of clustering the requirements revealed implications for needed classes and relation classes within the meta-model. Moreover, the model types required by different stakeholders of the modeling method initiative were identified. The meta-models are shown for each model type separately and in the end consolidated within one overall meta-model of the modeling language building block. Within the single model type meta-models, the elements identified within chapter 5.1.1, the domain specification by the DF, are shown in orange. The additional elements identified throughout the requirements engineering phase in chapter 5.1.2 are depicted in green. This stresses the integrative and evolving nature of the domain-specific design phase.

Figure 32 depicts the meta-model of the model type *process landscape*.

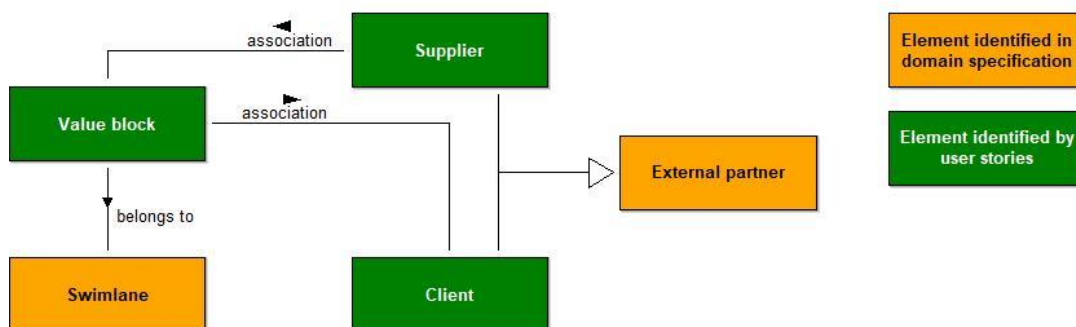


Figure 32 Process landscape meta-model of the AAL-DSL
- own representation

The goal of the process landscape model in Figure 32 is to provide an overview of the high-level processes, to which the automotive assembly line process belongs to. Moreover, the process hierarchy shows the external partners contributing to the respective processes. Management, core, and support processes are shown by the concept *swim lane*, which contains the *value blocks*, which represent the high-level processes. The abstract concept *external partner* is comprised of the *supplier* and *client* concepts. Those can be connected to the value blocks by the relation class *association* in order to depict deliveries from the supplier to the value block or from the value block to the client.

Figure 33 shows the *process model* meta-model of the AAL-DSL. The process model shows all necessary *tasks* between the *process-start* until the *process-end* in order to assemble the automobile. For each granularity level, there exists an own process model, which can be referenced from a *subprocess*. The tasks can be placed in *swim lanes* in order to show their belonging to certain departments or to provide structure. The process sequence is shown by the relation class *subsequent*, which connects the tasks and other process concepts like the *exclusive* and *parallel gateways*. The concepts *random generator* and *variable* enable the use of probabilities in case of exclusive paths. This is also valuable for the use of the simulation functionalities of the language.

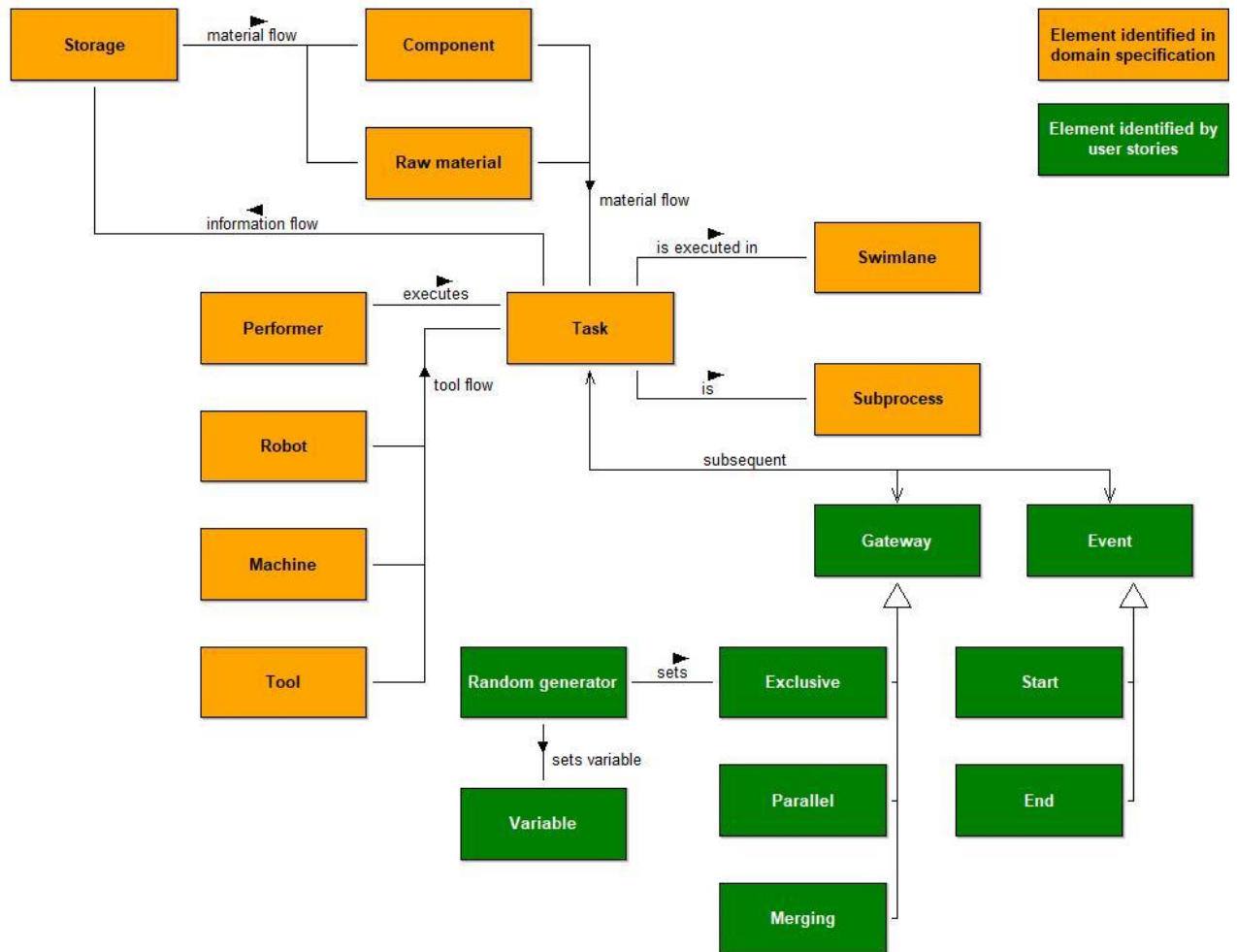


Figure 33 Process model meta-model of the AAL-DSL
- own representation

The process model (Figure 33) also enables the visualization of machinery and equipment, which is used within the process. Here, the concepts of *robot*, *machine*, and *tool* can be connected by the relation class *tool flow* to the process tasks. Each task can send messages to the *storage*, where *components* and *raw materials* are stored. This is depicted by the relation class *information flow*, whereas the sending of the materials from the storage to the respective task is depicted by a *material flow* relation. *Performers* are the human resources responsible for executing the tasks.

The *product structure* meta-model is shown in Figure 34. As several car types can be assembled by the same manufacturer on the same assembly line, a product structure model is needed. It shows the *raw materials* and *components*, of which the respective car types consist of. This is denoted by the relation class *is component of*. Furthermore, the *storage*

location of the materials can be stated, which is connected by a *material flow* to the components and raw materials.

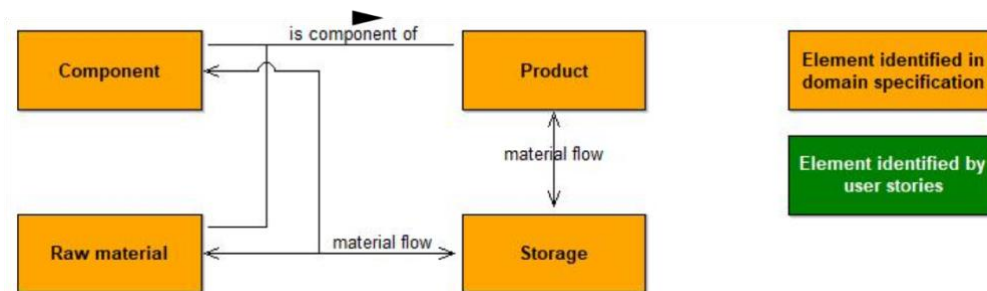


Figure 34 Product structure meta-model of the AAL-DSL
– own representation

In order to be able to assign tasks to certain *performers* in the process model, a *working environment model* is needed (see Figure 35). It shows the distinct *departments* and which performers are associated to them by the relation class *belongs to*. Moreover, several performers can share *roles*, which is denoted by the relation class *has role* within the working environment model.

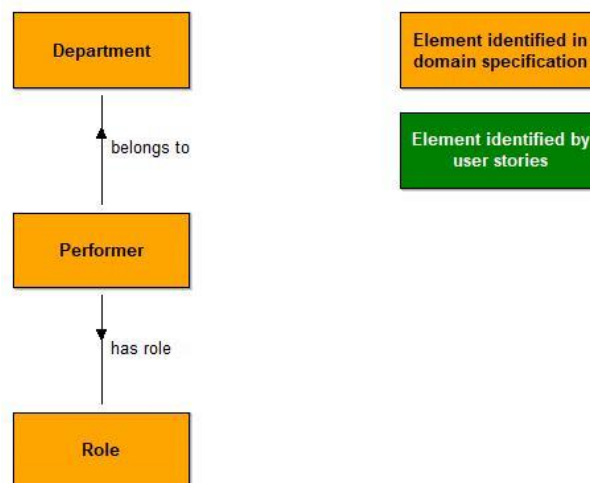


Figure 35 Working environment meta-model of the AAL-DSL
- own representation

Figure 36 depicts the complete domain-specific meta-model of the modeling language building block. It includes all identified concepts, which are needed in order to model the automotive assembly line and its surrounding processes in a suitable way.

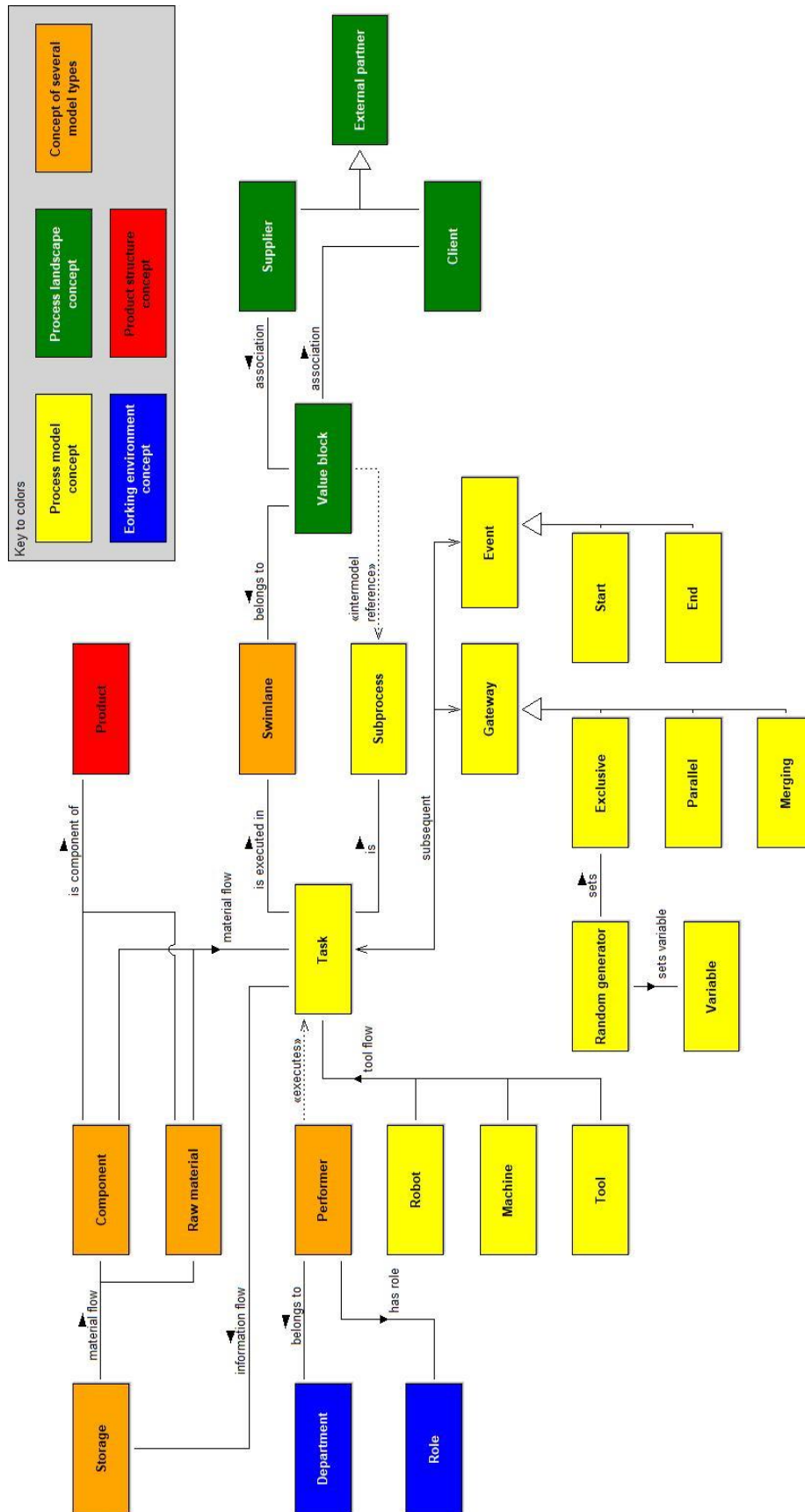


Figure 36 Complete meta-model of the AAL-DSL
- own representation

The color-scheme of Figure 36 helps to show the distinct model types. Concepts, which exclusively belong to the process model are depicted in yellow, those belonging to the working environment model in blue, the ones of the process landscape model in green, and concepts of the product structure model in red. The concepts, which appear in more than just one model type, are colored in orange. Dependencies and connections between the model types are shown by dotted lines, e.g. the *inter-model reference* between the concept value block and task and the association *executes* between the performer and the task.

5.1.3.2 Modeling Procedure

The modeling procedure specifies the phases and steps, which need to be undertaken, from the starting need for a BPM initiative to the DS modeling tool. In this case, it shows the views and responsibilities of the distinct roles within the DIF process and the respective tasks, they fulfill.

The modeling procedure meta-model is depicted in Figure 37. The *modeling method engineer* designs the DS modeling method, consisting of the language, procedure, and mechanisms & algorithms. The design of the DS modeling method by the modeling method engineer constitutes the first phase of the procedure. The modeling method itself consists of the three modeling method building blocks, which need to be designed consecutively. The language consists of the DS model types, which further consist of two concepts, namely *classes* and *relation classes*. For the mechanisms & algorithms part, *simulation* as well as *analysis* functionalities have to be evaluated, which fulfill the domain needs. The tool developer makes use of the developed modeling method and *creates the DS modeling tool* by implementing all concepts and functionalities of the modeling method. For the implementation, the ADOxx Development Toolkit is used by the tool developer. After successful implementation, the business process modeler, which is the language user of the tool, can work with the models and depict the steps and dependencies within the automotive assembly line process. Moreover, time, human, and material resources are added to the processes in order to enable working mechanisms and algorithms.

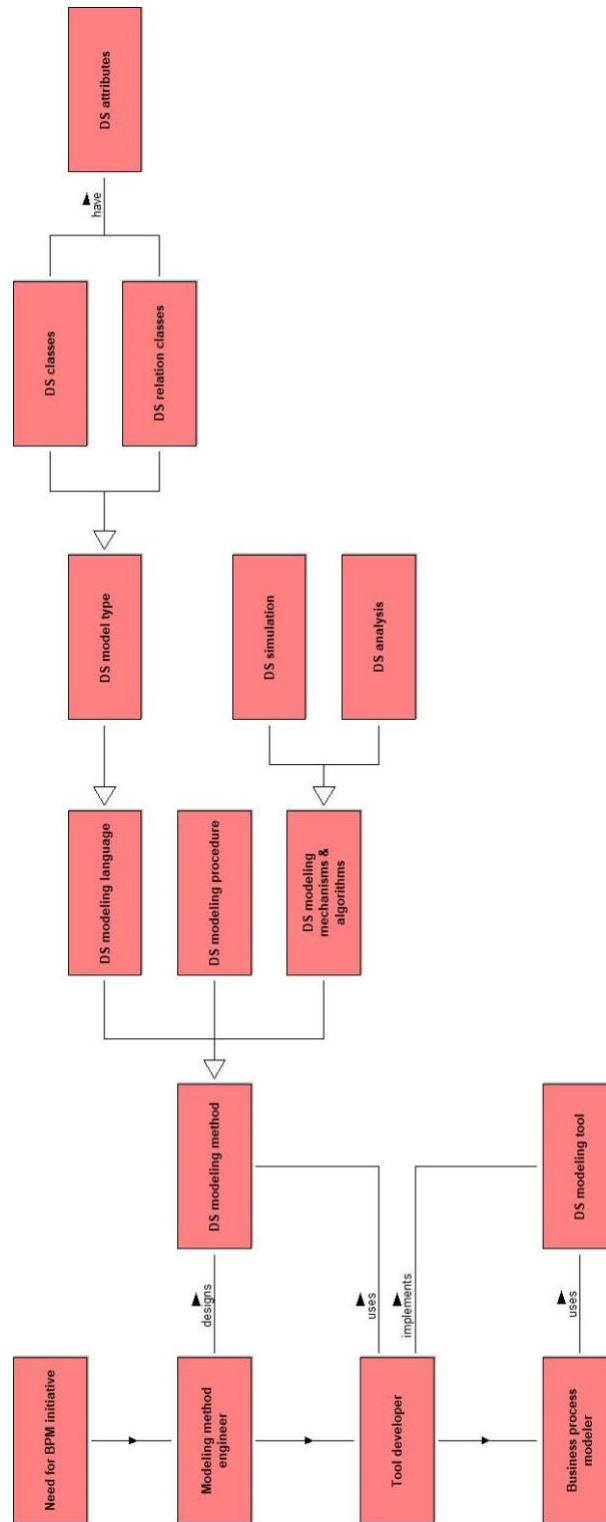


Figure 37 Modeling procedure meta-model of the AAL-DSL
– own representation

5.1.3.3 Modeling Mechanisms & Algorithms

The meta-model of the *modeling mechanisms and algorithms* building block of the method demonstrates the relationships between *inputs*, the *analysis and simulation* functionalities, and the *outputs*, as depicted in Figure 38.

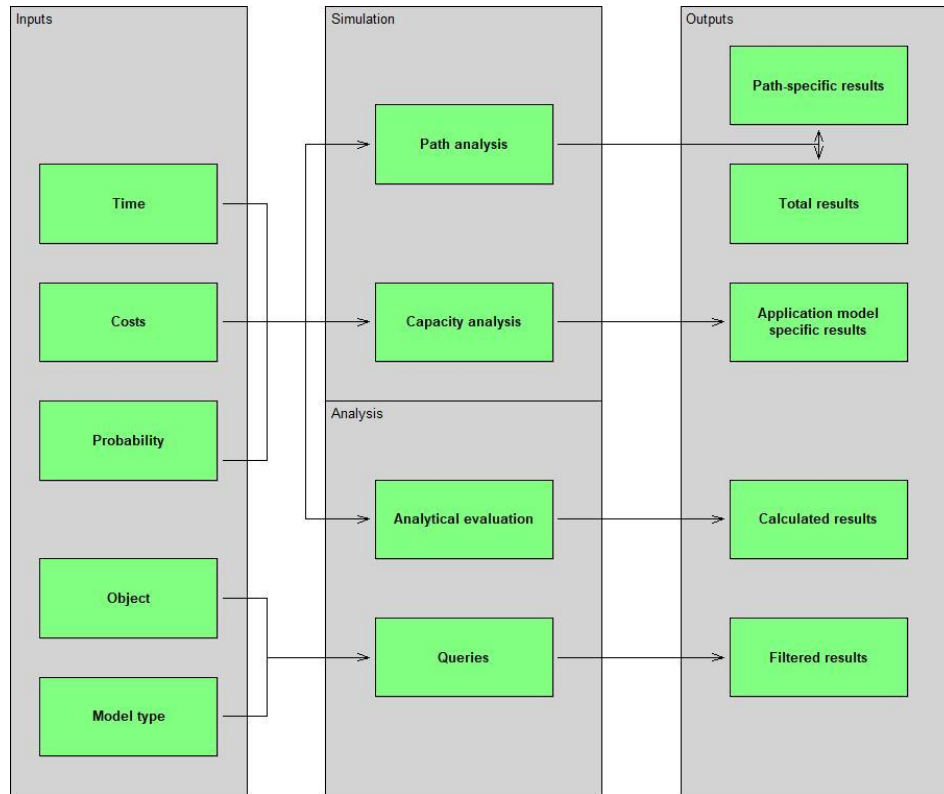


Figure 38 Modeling mechanisms and algorithms meta-model of the AAL-DSL
– own representation

The functionalities needed within the domain-specific BPM method for automotive assembly line modeling comprise analysis and simulation capabilities. Necessary inputs, like time, costs, and probability are defined within the simulatable process. Those inputs are used for the path analysis, capacity analysis, analytical evaluation, and queries. After running through the algorithmic functionalities, valuable outputs for management and operations are generated. For example, the total cycle time for different time units (week, month, year) can be assessed and costs for single workers, roles, or departments evaluated.

5.2 Domain-specific Implementation

This chapter showcases, how the meta-models of the modeling method from the previous chapter are implemented. The result of the implementation phase is a ready-to use modeling tool, namely the automotive assembly line domain-specific language (AAL-DSL).

5.2.1 Linguistic Matching Heuristic

The linguistic matching heuristic described in chapter 4.2.1 is used on the automotive assembly line domain within this chapter. The output of the domain-specific design phase, the modeling language meta-models of the different model types, is translated into the domain-specific modeling tool by using the linguistic matching heuristic. Aiming at identifying synergies and reusable parts, already existing meta-model concepts of general-purpose as well as domain-specific modeling languages are mapped to the concepts of the automotive assembly line language.

*Table 14 Matching heuristic applied
– own representation*

| Model type | Concept | Matching BPML | Matching concept |
|----------------------------------|------------------|-----------------|------------------|
| Process landscape | Value block | IBPM | Process |
| Process landscape | Supplier | - | - |
| Process landscape | Client | - | - |
| Process landscape | Association | UML | Association |
| Process landscape | External partner | IBPM | External partner |
| Process landscape, process model | Swim lane | BPMN2.0 | Swim lane |
| Process model | Task | BPMN2.0 | Task |
| Process model | Tool | SIMchronization | Equipment |

| | | | |
|--|-------------------|-----------------|-------------------|
| Process model | Robot | - | - |
| Process model | Machine | IBPM | Machines & tools |
| Process model | Parallel gateway | BPMN2.0 | Parallel gateway |
| Process model | Merging gateway | - | - |
| Process model | Exclusive gateway | BPMN2.0 | Exclusive gateway |
| Process model | Start event | BPMN2.0 | Start event |
| Process model | End event | BPMN2.0 | End event |
| Process model | Variable | IBPM | Variable |
| Process model | Random generator | IBPM | Random generator |
| Process model | Subprocess | BPMN2.0 | Subprocess |
| Process model | Subsequent | BPMN2.0 | Sequence flow |
| Process model | Sets variable | IBPM | Sets variable |
| Process model | Sets | IBPM | Sets |
| Process model | Information flow | BPMN2.0 | Message flow |
| Process model | Tool flow | - | - |
| Process model | Material flow | IBPM | Parts flow |
| Process model, product structure model | Component | SIMchronization | Component |
| Process model, product structure model | Raw material | - | - |
| Process model, product structure model | Storage | IBPM | Buffer |
| Product structure model | Product | SIMchronization | Item |

| Product structure model | Is component of | UML | Generalization |
|---------------------------|-----------------|------|---------------------|
| Working environment model | Performer | IBPM | Performer |
| Working environment model | Department | IBPM | Organizational unit |
| Working environment model | Role | IBPM | Role |
| Working environment model | Has role | IBPM | Has role |
| Working environment model | Belongs to | IBPM | Belongs to |

An example for the use of each linguistic matching heuristic is given in the following.

Concepts, which could not be identified in other existing modeling languages, need to be created. An example is the class “Robot” and the heuristic matching depicted in Figure 39.



*Figure 39 Linguistic matching heuristic – create
- own representation, icon from [39]*

An example for the extend matching heuristic is the need for a material flow in addition to the standard BPMN2.0 process model. This is depicted in Figure 40.

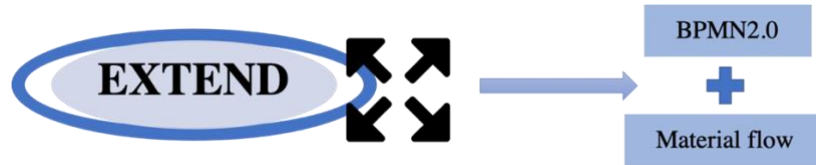


Figure 40 Linguistic matching heuristic – extend
 - own representation, icon from [83]

In above Table 14, quite often similarities to concepts from the modeling language IBPM [15] were identified. However, the IBPM language is far too extensive for the purpose of modeling the automotive assembly line. Therefore, reduction is applied as can be seen in Figure 41.



Figure 41 Linguistic matching heuristic – reduce
 - own representation, icon from [49]




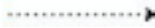




Because this process is a heuristic and requires prior knowledge of the problem domain, some elements might not be identified within existing languages. The goal is to find as many fitting concepts as possible.










5.2.2 Domain-specific Modeling Tool – Automotive Assembly Line DSL (AAL-DSL)













The automotive assembly line domain-specific language (AAL-DSL) was implemented in the ADOxx Development Environment following the analysis of the previous chapters. Table 15 shows the graphical representation and the associated attributes of each concept. The detailed tree structure exported from ADOxx is not shown here due to the extensive text length. The tree structure of the dynamic library is listed in Appendix B. Appendix C shows the tree structure of the static library.


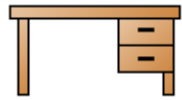


Table 15 Concepts of the AAL-DSL

– own representation

| Model type | Concept | Graphical representation | Attributes |
|----------------------------------|------------------|--|--|
| Process landscape | Value block |  | Name, referenced process, display name and reference, order |
| Process landscape | Supplier |  | Name, description |
| Process landscape | Client |  | Name, description |
| Process landscape | Association |  | Name |
| Process landscape | External partner | Abstract class | None |
| Process landscape, process model | Swim lane |  | Name, color, alignment |
| Process model | Task |  | Name, execution time, waiting time, resting time, transport time, costs, performer |
| Process model | Tool |  | Name |
| Process model | Robot |  | Name |

| | | | |
|---------------|-------------------|---|---|
| Process model | Machine |  | Name |
| Process model | Parallel gateway |  | Name |
| Process model | Merging gateway |  | Name |
| Process model | Exclusive gateway |  | Name |
| Process model | Start event |  | Name, quantity, time period, process calendar, tolerance waiting time, abandon after tolerance waiting time |
| Process model | End event |  | Name |
| Process model | Variable |  | Name, variable time, variable scope |
| Process model | Random generator |  | Name, value |
| Process model | Subprocess |  | Name, referenced subprocess, order, description, aggregated execution time, aggregated waiting time, aggregated resting time, aggregated transport time, aggregated costs |

| | | | |
|--|------------------|--|--|
| Process model | Subsequent |  | Transition condition, transition probability, visualized values, comment |
| Process model | Sets variable |  | None |
| Process model | Sets |  | None |
| Process model | Information flow |  | Name |
| Process model | Tool flow |  | Name |
| Process model | Material flow |  | Name |
| Process model, product structure model | Component |  | Name |
| Process model, product structure model | Raw material |  | Name |
| Process model, product structure model | Storage |  | Name |
| Product structure model | Product |  | Name |
| Product structure model | Is component of |  | None |
| Working environment model | Performer |  | Name, hourly wages, personnel costs, availability, calendar, capacity, workload, info on results |

| | | | |
|---------------------------|------------|---|------|
| Working environment model | Department |  | Name |
| Working environment model | Role |  | Name |
| Working environment model | Has role |  | None |
| Working environment model | Belongs to |  | None |

5.3 Domain-specific Modeling

After successful completion of the implementation phase, this chapter deals with the modeling of all needed processes and the usage of the developed AAL-DSL. This phase is usually done by the domain experts – the users of the DS modeling tool. Here, the previous work enters the test phase and it gets clear, whether all needed concepts and functionalities were considered.

5.3.1 Models of the AAL-DSL

In the following, the models of the automotive assembly line domain-specific language are shown and described. Moreover, results of the DS analysis and simulation are shown. Figure 42 depicts an overview of all models, which were implemented by using the AAL-DSL. The green circles show the process landscape on level 0 and the process models on the granularity levels from 1 to 3. These process models are linked to each other by process references within each model, which lead to the next, more detailed process level. The orange circles show the product structure model (PS) as well as the working environment models (WE). In the further course of this chapter, all model types are depicted and described separately and in greater detail.

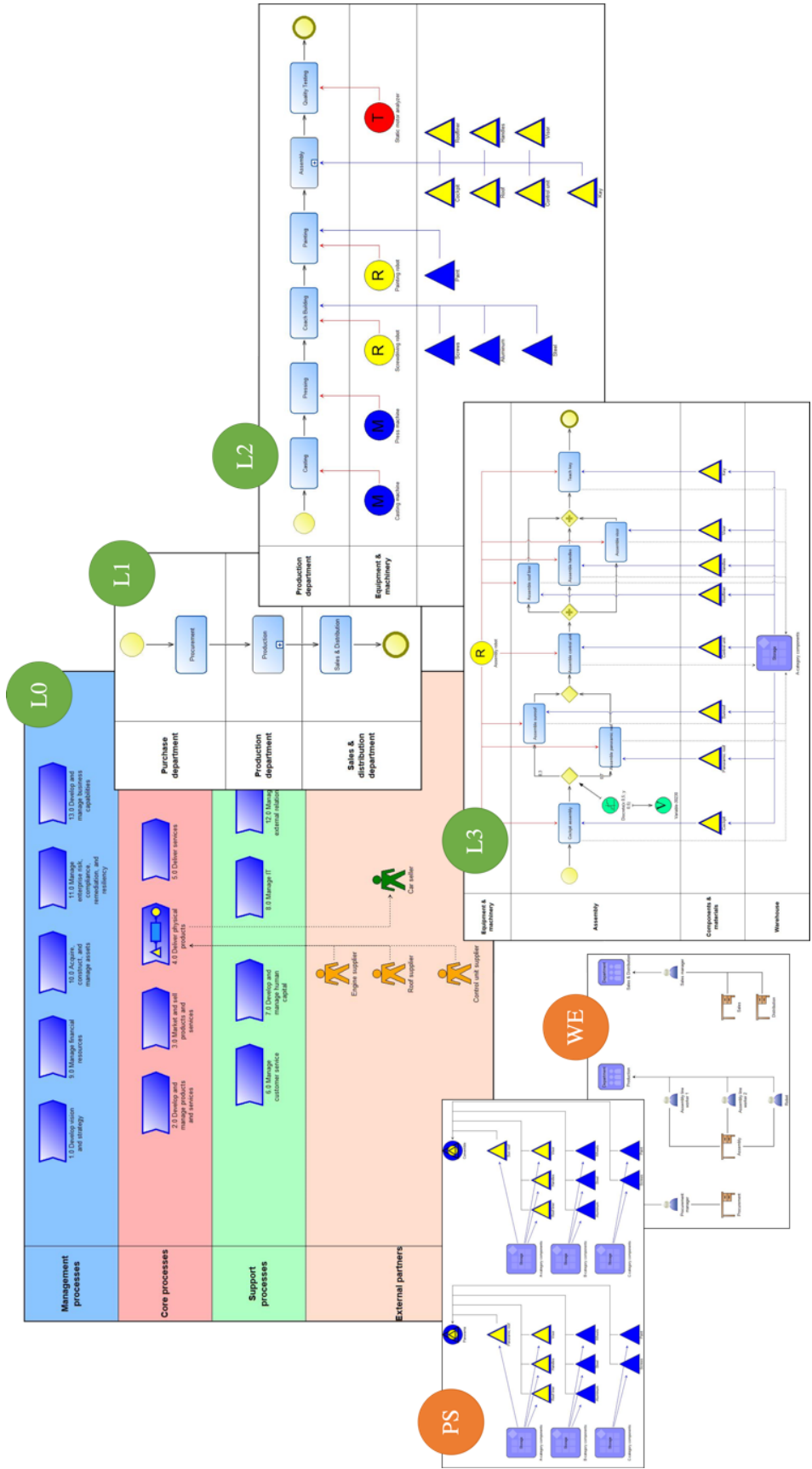


Figure 42 Model overview
- own representation

The process landscape model of the AAL-DSL is shown in Figure 43. The structure is derived from the industry-specific PCF[®] for the automotive industry [8]. The numbers in front of the process steps signify the sequence of the value blocks as proposed within the PCF[®].

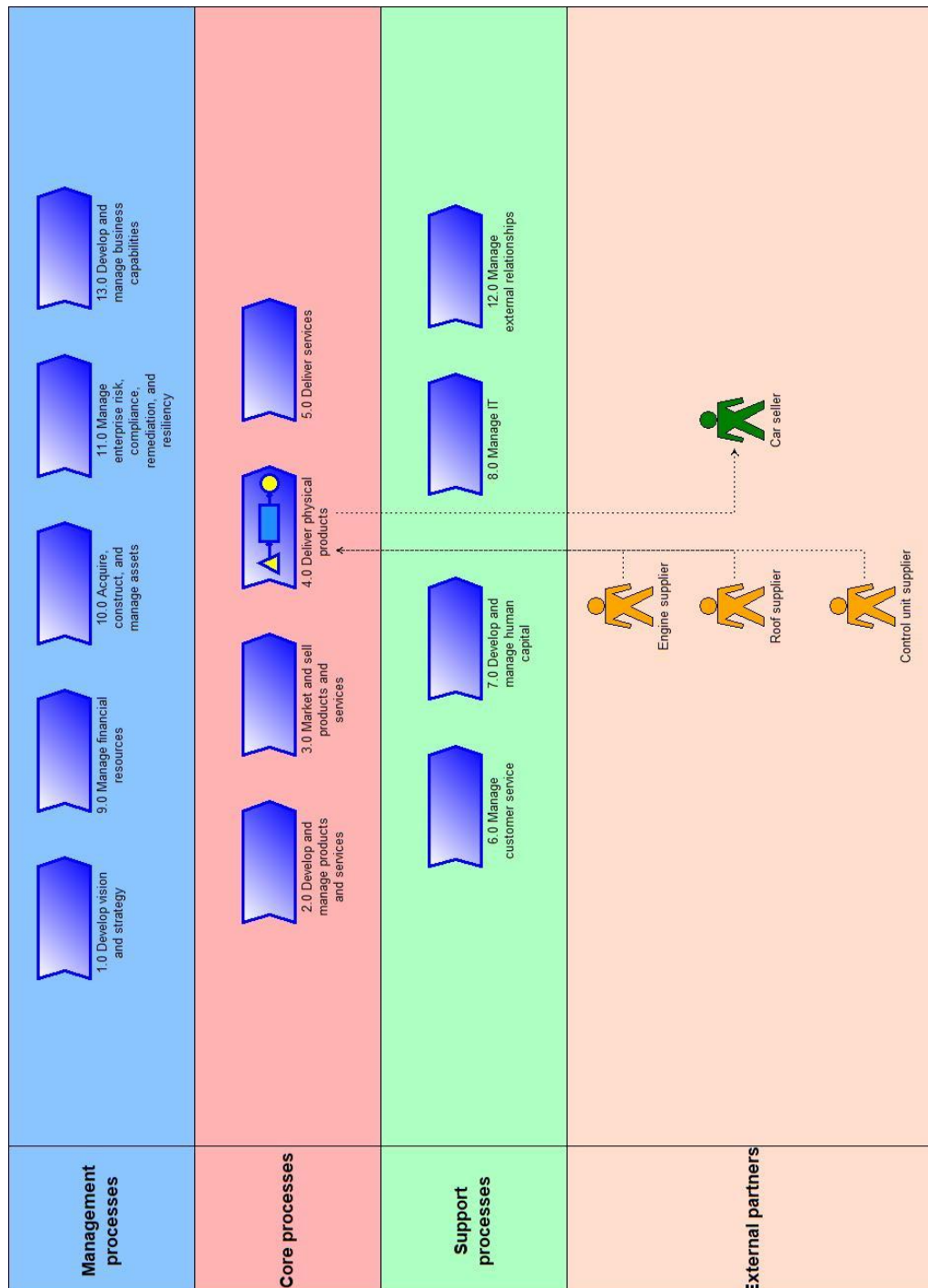


Figure 43 Process landscape L0

- own representation, process based on PCF[®] [8]

The swim lanes within Figure 43 structure the process value blocks into management, core, and support processes. Management processes within the automotive industry are strategy development, asset and risk management, and the management of financial resources and business capabilities. The core processes resemble all activities, which directly create value for the company and are essential for its financial success. This includes the development of new products, their marketing, production, and delivery to the clients. Also, services offered in this context are considered as core processes. Support processes of a car manufacturer are customer service, human resource management, IT, and partner management. As the focus within this proof of concept lies in the core process of modeling the automotive assembly line, the inter-model reference can be seen from the value block 4.0 Deliver physical products. Here, also associations to external partners of this process step are shown. On the left hand-side, three suppliers of the car manufacturer are depicted, which deliver components for the process. On the right hand-side the car seller as the client, to which the final product is delivered, is shown.

Figure 44 shows the level 1 process model, which is referenced by the value block 4.0 Deliver physical products in the process landscape. It shows the three major tasks of parts and materials procurement, production, and sales & distribution. The departments, in which these tasks are undertaken are depicted by swim lanes. A referenced sub-process is graphically symbolized by the plus sign of the step *production*.

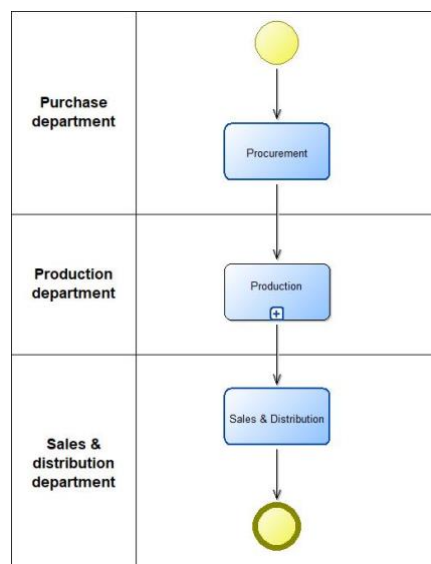


Figure 44 Process Model L1

- own representation, process based on [92]

Referenced in the level 1 process *Production*, the process model on level 2 is shown in Figure 45.

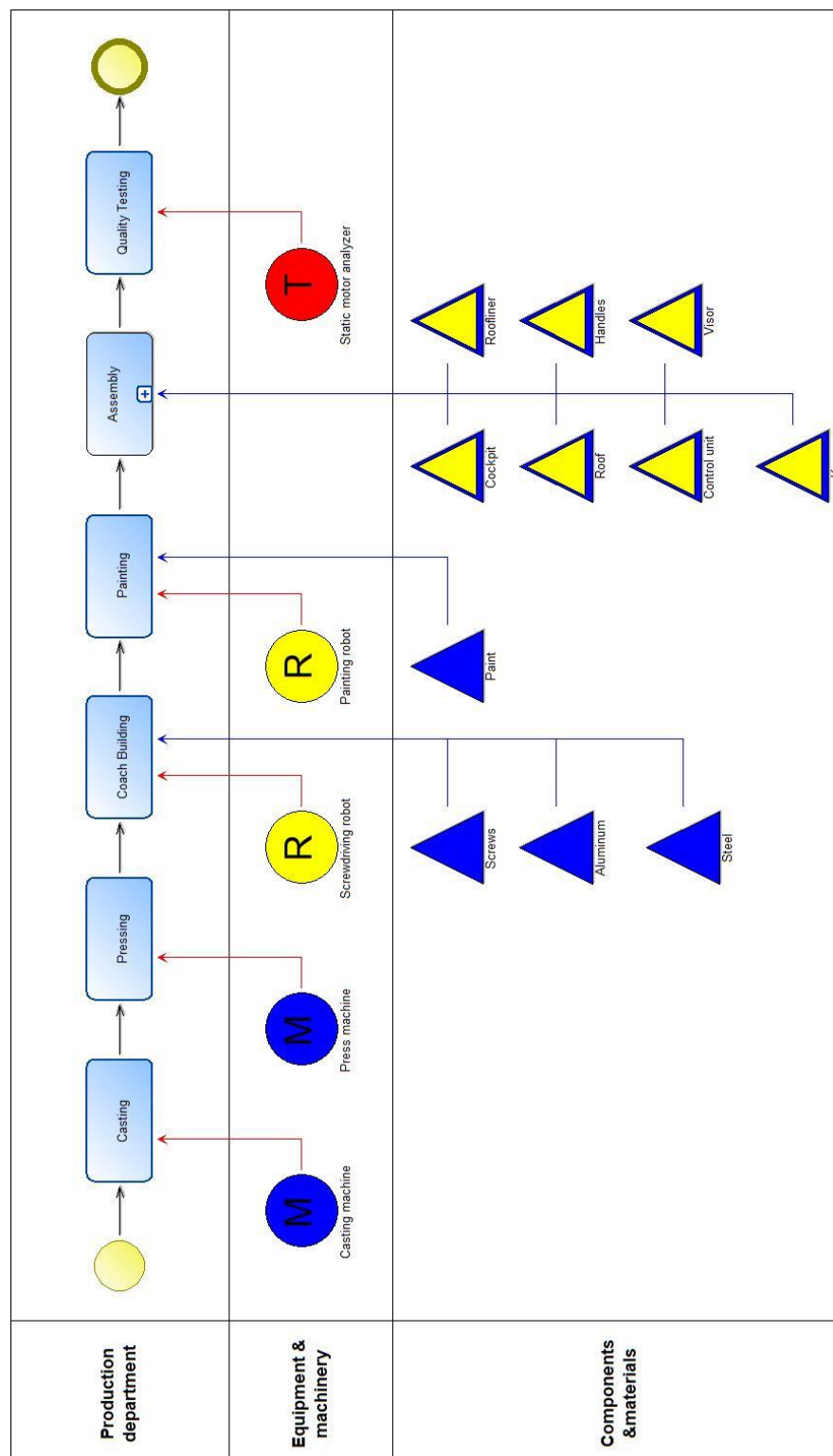


Figure 45 Process Model L2

- own representation, process based on [92]

Figure 45 shows the process of producing an automobile, as described in [92]. The swim lanes in the model help to structure the components of the process. The swim lane, in which the process steps take place, is the *production department* of the car manufacturer. Here, the subsequent tasks from start until the end are shown. The steady process flow indicates, that the car manufacturer uses a production line, in which the components are delivered JIS. Component and raw material delivery for the respective process tasks are shown by the swim lane *components & materials* in combination with *material flow* relations. A third swim lane is used to show the *equipment & machinery*, which is needed within the different production steps of the automobile. For the purpose of a good process-overview, the *tool flow* is illustrated, too. It shows, in which process tasks the respective *machines*, *robots*, and *tools* of the car manufacturer come to use. The assembly process is referenced in the next level, as indicated by the plus sign in the graphical representation if the task *assembly*.

Following the process reference from the previous level 2 process, the assembly process of the automobile is presented in greater detail in Figure 46. The focus is set on this process, as this constitutes the domain of interest within this thesis. Due to two different car models being produced within the company, an exclusive gateway resembles the discrete distribution of cars with a sunroof versus cars with a panoramic roof. Around 70% of the cars are built with a panoramic roof, whereas around 30% are built with a sun roof. As in the previous model, a swim lane is used to depict the *assembly* process. By using the attribute definition in the notebook of each task, information about the execution, waiting, resting, and transport time as well as the associated costs of the task are fed into the process. Moreover, the performer of the respective task is referenced from the working environment model. By providing this information, the analysis and simulation functionality of the AAL-DSL is enabled. The performers referenced within the tasks of the automotive assembly line process are assisted by an *assembly robot*. It is placed in the *machinery & equipment* swim lane and connected to the respective tasks by a *tool flow* relation. The components and raw materials are added to the tasks JIT. This fact is indicated by the blue *material flow*, which connects the *components & materials* swim lane to the process steps. Furthermore, the *warehouse* swim lane shows the storage, from which the respective materials are delivered in-house. As within this process, all materials are semi-finished, they come from the storage for A-category components.

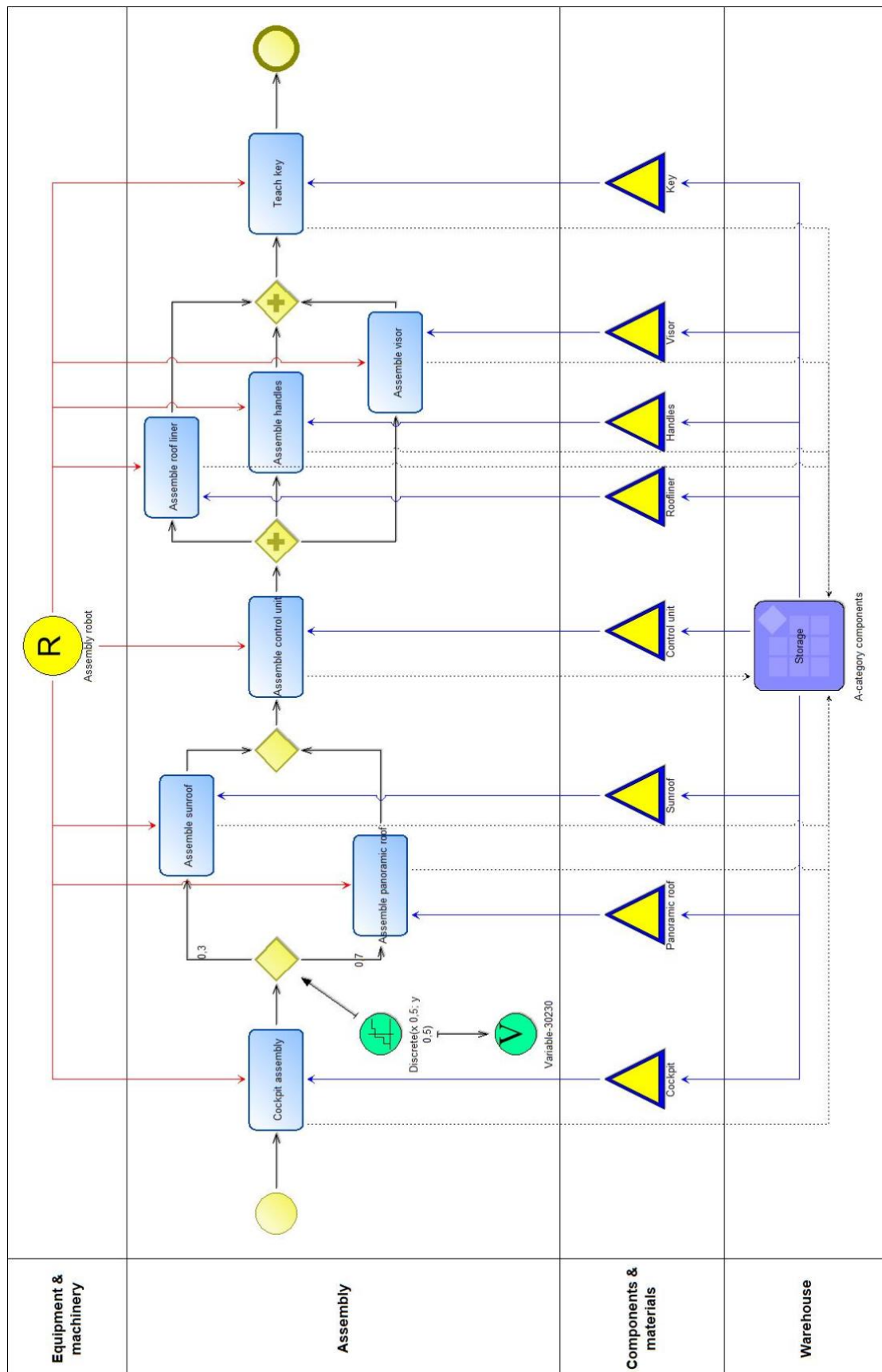


Figure 46 Process Model L3

- own representation, process based on [92]

Figure 47 shows the working environment associated with the process 4.0 Deliver physical products. It shows the departments involved, which were shown as swim lanes in Figure 44. The performers, which are referenced in the process tasks of Figure 46, are connected to the respective department by a *belongs to* relation. What is more, the notebooks of the performers allow to add working times, wages, and availability by the attribute definition within the implementation phase of the AAL-DSL. The provision of this information is also needed in order to permit the simulation and analysis functionalities of the language. *Roles*, which can be shared by several performers, are connected by the relation *has role*.

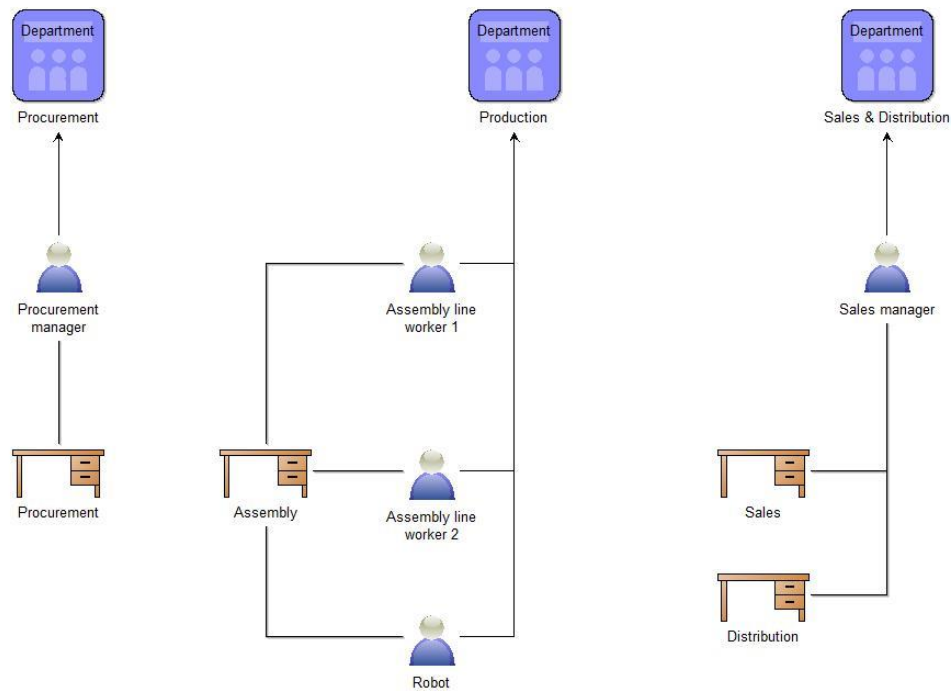


Figure 47 Working Environment Model
- own representation

Within the automotive assembly process, two distinct products are produced, one being a convertible and the other one a car with a panoramic roof. These are shown in the *product structure model* in Figure 48. As the car manufacturer uses as many synergies as possible, most of the parts are the same for both product types. The only difference is the roof, as the convertible needs a sun roof, whereas the panoramic car needs a panoramic roof. There exist three types of storages at the car manufacturer, which are divided into A-category components, B-category components, and C-category components.

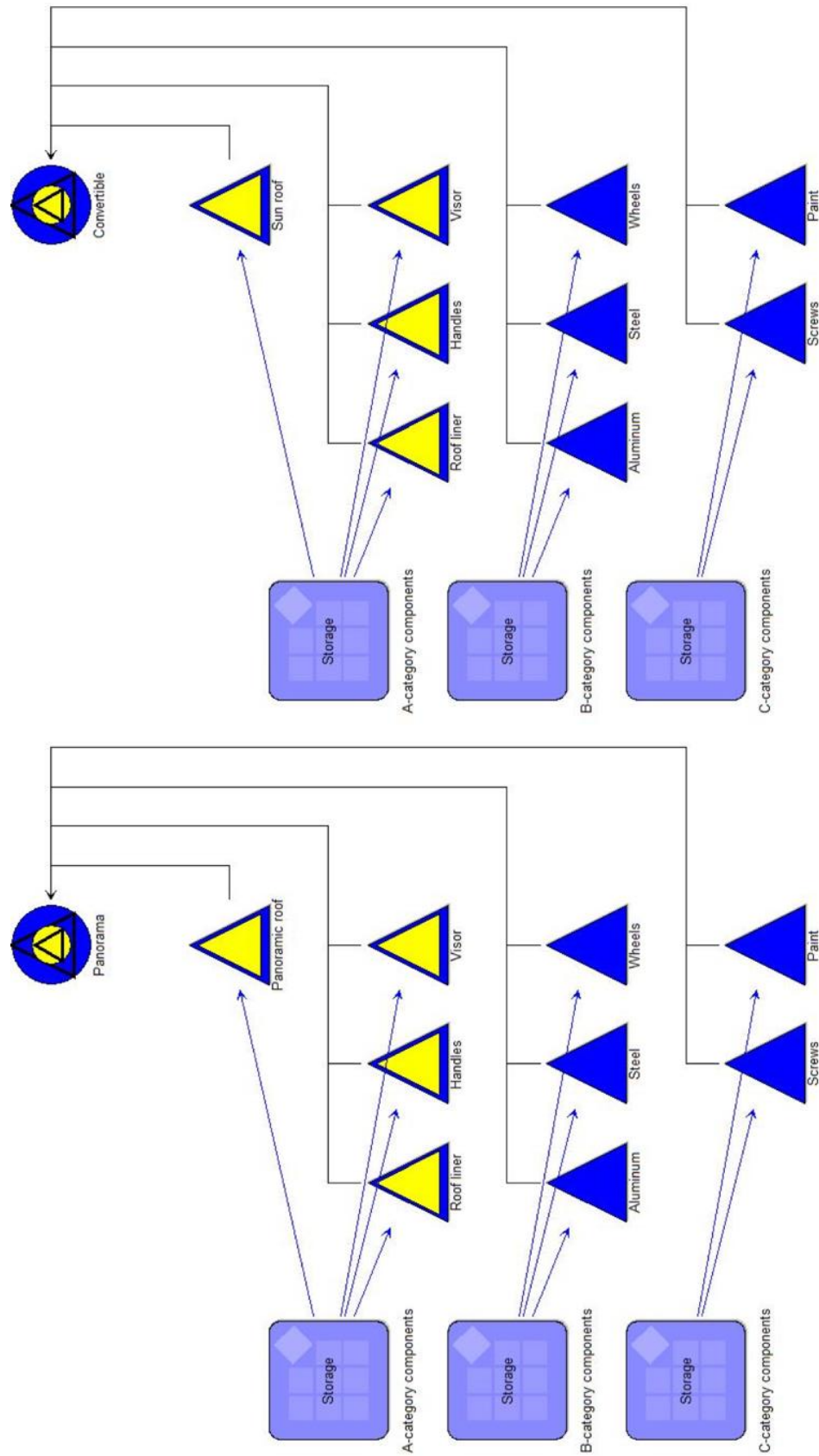


Figure 48 Product Structure Model
- own representation

In order to enable the simulation and analysis of the processes, the activity times and costs, as well as the performing role in the working environment have to be defined within the notebook of each process step. Figure 49 shows the task procurement as an example. The mere execution time for all procurement activities is assumed to be approximately 10 days. The waiting time, which results from awaiting replies and document handover tasks is stated with around 20 days. Resting time specifies the time, the task is ready to be proceeded but delayed in the responsibility of the own performers. The transport time is assumed to be 8 days. The costs for the whole procurement process are assumed to be € 12.000. It has to be stated, that due to restricted domain-knowledge and information sources, the times and costs are of an indicative nature and are not exhaustive.

The screenshot shows a software window titled "Procurement (Task)". It contains two main sections: "Activity times" and "Activity costs".

Activity times:

- Execution time: 00:010:00:00:00
- Waiting time: 00:020:00:00:00
- Resting time: 00:005:00:00:00
- Transport time: 00:008:00:00:00

Activity costs:

- Costs: 12000.000000

On the right side of the window, there is a sidebar with tabs: "Description", "Times/Costs" (which is selected), and "Working environment".

*Figure 49 Definition of task parameters
– own representation*

As part of the analysis, the query enables the collection and overview of selected objects and processes. The result of a query collecting all objects of class “Task” of all models can be seen in Figure 50. As the query was executed on the level 1 process 4.0 Deliver physical products, all elements of class task are shown in a hierarchical manner.

Query results - (<"Task">)

| |
|----------------------------------|
| 1. 4.0 Deliver physical products |
| Procurement |
| Sales & Distribution |
| 2. Assembly |
| Assemble control unit |
| Assemble handles |
| Assemble panoramic roof |
| Assemble roof liner |
| Assemble sunroof |
| Assemble visor |
| Cockpit assembly |
| Teach key |
| 3. Production |
| Casting |
| Coach Building |
| Painting |
| Pressing |
| Quality Testing |

Figure 50 Query results
- own representation

The analytical evaluation results of the automotive assembly line process are shown in Figure 51. With 170 workdays per year and 8 working hours per day, the execution time in working days and the cycle time is calculated. For the overall process 4.0 Deliver physical products, the total cycle time amounts to around 84 days and costs of approximately € 22.100 per produced unit.

Analytic evaluation results - 4.0 Deliver physical products (Days per year: 170.00 / Working hours per day: 8.00 / Volume: 1)

| | Process Model | Task | frequency | Execution time | Waiting time | Resting time | Transport time | Costs | Execution time in working days | Cycle time |
|------|-------------------------------|-------------------------|-----------|-----------------|-----------------|-----------------|-----------------|----------|--------------------------------|-----------------|
| 1. | 4.0 Deliver physical products | | - | 00:037:00:30:00 | 00:024:04:00:00 | 00:008:01:40:00 | 00:016:03:45:00 | 22100.00 | 37.0625 | 00:084:06:05:00 |
| 1.1. | | Procurement | 1.00 | 00:010:00:00:00 | 00:020:00:00:00 | 00:005:00:00:00 | 00:008:00:00:00 | 12000.00 | 10.00 | - |
| 1.2. | | Sales & Distribution | 1.00 | 00:020:00:00:00 | 00:003:00:00:00 | 00:002:00:00:00 | 00:008:00:00:00 | 4000.00 | 20.00 | - |
| 2. | Assembly | | - | 00:004:03:00:00 | 00:000:03:30:00 | 00:000:02:20:00 | 00:000:00:35:00 | 2800.00 | 4.375 | 00:003:05:35:00 |
| 2.1. | | Cockpit assembly | 1.00 | 00:000:05:00:00 | 00:000:00:30:00 | 00:000:00:20:00 | 00:000:00:05:00 | 400.00 | 0.625 | - |
| 2.2. | | Assemble panoramic roof | 0.70 | 00:000:03:30:00 | 00:000:00:21:00 | 00:000:00:14:00 | 00:000:00:03:30 | 280.00 | 0.4375 | - |
| 2.3. | | Assemble sunroof | 0.30 | 00:000:01:30:00 | 00:000:00:09:00 | 00:000:00:06:00 | 00:000:00:01:30 | 120.00 | 0.1875 | - |
| 2.4. | | Assemble control unit | 1.00 | 00:000:05:00:00 | 00:000:00:30:00 | 00:000:00:20:00 | 00:000:00:05:00 | 400.00 | 0.625 | - |
| 2.5. | | Assemble roof liner | 1.00 | 00:000:05:00:00 | 00:000:00:30:00 | 00:000:00:20:00 | 00:000:00:05:00 | 400.00 | 0.625 | - |
| 2.6. | | Assemble handles | 1.00 | 00:000:05:00:00 | 00:000:00:30:00 | 00:000:00:20:00 | 00:000:00:05:00 | 400.00 | 0.625 | - |
| 2.7. | | Assemble visor | 1.00 | 00:000:05:00:00 | 00:000:00:30:00 | 00:000:00:20:00 | 00:000:00:05:00 | 400.00 | 0.625 | - |
| 2.8. | | Teach key | 1.00 | 00:000:05:00:00 | 00:000:00:30:00 | 00:000:00:20:00 | 00:000:00:05:00 | 400.00 | 0.625 | - |
| 3. | Production | | - | 00:002:05:30:00 | 00:001:00:30:00 | 00:000:07:20:00 | 00:000:03:10:00 | 3300.00 | 2.6875 | 00:008:06:05:00 |
| 3.1. | | Casting | 1.00 | 00:000:05:00:00 | 00:000:04:00:00 | 00:000:03:00:00 | 00:000:02:00:00 | 400.00 | 0.625 | - |
| 3.2. | | Pressing | 1.00 | 00:000:02:00:00 | 00:000:01:00:00 | 00:000:00:30:00 | 00:000:00:10:00 | 700.00 | 0.25 | - |
| 3.3. | | Coach Building | 1.00 | 00:001:02:00:00 | 00:000:01:00:00 | 00:000:01:50:00 | 00:000:00:20:00 | 1400.00 | 1.25 | - |
| 3.4. | | Painting | 1.00 | 00:000:01:30:00 | 00:000:00:30:00 | 00:000:01:30:00 | 00:000:00:10:00 | 500.00 | 0.1875 | - |
| 3.5. | | Quality Testing | 1.00 | 00:000:03:00:00 | 00:000:02:00:00 | 00:000:00:30:00 | 00:000:00:30:00 | 300.00 | 0.375 | - |

Figure 51 Analytic evaluation results
- own representation

Figure 52 depicts the path analysis results for path number 1 (sun roof car variant) and path number 2 (panoramic roof car variant). The path probability resembles the ratio of the product *convertible* and the product *panoramic* to total car production, respectively. 30% of

the cars are of type convertible and need to be built with a sun roof, whereas 70% are panoramic cars, which need a panoramic roof. The path analysis shows the resulting probability with 1.000 simulations as well as the expected times and costs for each path.

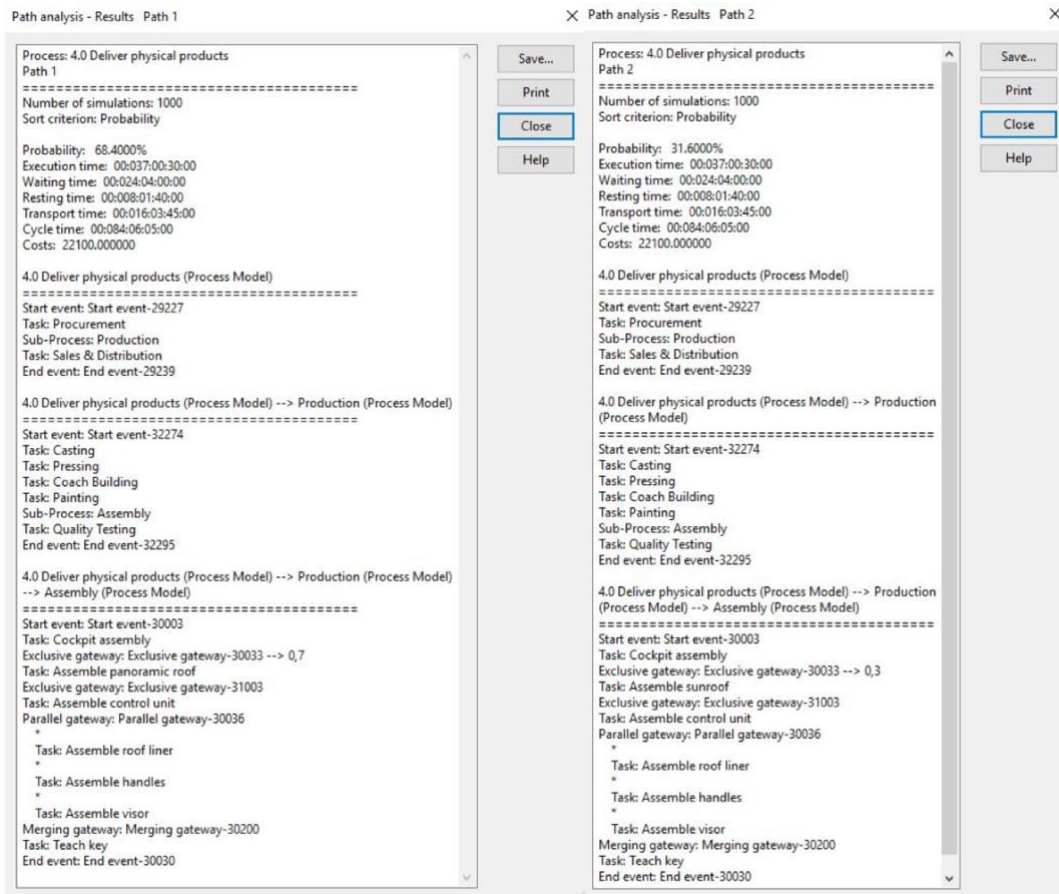


Figure 52 Path analysis results

- own representation

Figure 53 shows the overall expected value of the execution time, waiting time, resting time, transport time, cycle time, and costs after running 1.000 simulations.

| Simulation results - Path analysis - Dynamic model: 4.0 Deliver physical products | | |
|---|-----------------|--|
| | Expected value | |
| Execution time | 00:037:00:30:00 | |
| Waiting time | 00:024:04:00:00 | |
| Resting time | 00:008:01:40:00 | |
| Transport time | 00:016:03:45:00 | |
| Cycle time | 00:084:06:05:00 | |
| Costs | 22100.000000 | |

Figure 53 Simulation results path analysis

- own representation

The capacity analysis results are shown in Figure 54. It shows the work-capacity per assigned worker per task and gives an overview of their respective workload.

Simulation results - Capacity analysis (Working environment/Per month) - Application model: Assembly line capacity analysis

| | Department | Process Model | Task | Performer | Number | Execution time (sum) | Personnel cost (sum) | Personnel cost (average) | Costs (sum) |
|----------|------------------------------------|----------------------|--|--|-----------|----------------------|----------------------|--------------------------|---------------|
| 1. | Production (5 Working Environment) | | | | | 01:048:06:00:00 | 42500.000000 | | 140000.000000 |
| 1.1. | | 4 Assembly Process L | | | | 01:048:06:00:00 | 42500.000000 | | 140000.000000 |
| 1.1.1. | | | Cockpit assembly (4 Assembly Process) | | 50.000000 | 00:031:02:00:00 | 7500.000000 | 150.000000 | 20000.000000 |
| 1.1.1.1. | | | | Assembly line worker 1 (5 Working Env) | 50.000000 | 00:031:02:00:00 | 7500.000000 | 150.000000 | 20000.000000 |
| 1.1.2. | | | Assemble panoramic roof (4 Assembly Process) | | 24.900000 | 00:015:04:30:00 | 3735.000000 | 150.000000 | 9960.000000 |
| 1.1.2.1. | | | | Assembly line worker 1 (5 Working Env) | 24.900000 | 00:015:04:30:00 | 3735.000000 | 150.000000 | 9960.000000 |
| 1.1.3. | | | Assemble sunroof (4 Assembly Process) | | 25.100000 | 00:015:05:30:00 | 3765.000000 | 150.000000 | 10040.000000 |
| 1.1.3.1. | | | | Assembly line worker 1 (5 Working Env) | 25.100000 | 00:015:05:30:00 | 3765.000000 | 150.000000 | 10040.000000 |
| 1.1.4. | | | Assemble control unit (4 Assembly Process) | | 50.000000 | 00:031:02:00:00 | 5000.000000 | 100.000000 | 20000.000000 |
| 1.1.4.1. | | | | Assembly line worker 2 (5 Working Env) | 50.000000 | 00:031:02:00:00 | 5000.000000 | 100.000000 | 20000.000000 |
| 1.1.5. | | | Assemble roof liner (4 Assembly Process) | | 50.000000 | 00:031:02:00:00 | 5000.000000 | 100.000000 | 20000.000000 |
| 1.1.5.1. | | | | Assembly line worker 2 (5 Working Env) | 50.000000 | 00:031:02:00:00 | 5000.000000 | 100.000000 | 20000.000000 |
| 1.1.6. | | | Assemble handles (4 Assembly Process) | | 50.000000 | 00:031:02:00:00 | 5000.000000 | 100.000000 | 20000.000000 |
| 1.1.6.1. | | | | Assembly line worker 2 (5 Working Env) | 50.000000 | 00:031:02:00:00 | 5000.000000 | 100.000000 | 20000.000000 |
| 1.1.7. | | | Assemble visor (4 Assembly Process L3) | | 50.000000 | 00:031:02:00:00 | 7500.000000 | 150.000000 | 20000.000000 |
| 1.1.7.1. | | | | Assembly line worker 1 (5 Working Env) | 50.000000 | 00:031:02:00:00 | 7500.000000 | 150.000000 | 20000.000000 |
| 1.1.8. | | | Teach key (4 Assembly Process L3 1) | | 50.000000 | 00:031:02:00:00 | 5000.000000 | 100.000000 | 20000.000000 |
| 1.1.8.1. | | | | Assembly line worker 2 (5 Working Env) | 50.000000 | 00:031:02:00:00 | 5000.000000 | 100.000000 | 20000.000000 |

Figure 54 Capacity analysis results

- own representation

The above shown and described model types and functionalities aim at giving an overview of the AAL-DSL to the reader. The domain-specific modeling tool fulfills the requirements derived within the domain-specific design part of this proof-of-concept. It can be used to further define the processes within a car manufacturing company and depict and simulate times and costs in order to identify process bottlenecks and introduce optimization initiatives.

5.3.2 Quality Criteria and New Requirements

The fit between the modeling method and the domain of automotive assembly process modeling is the highest at completion of the DS implementation phase. At this point, a regular re-evaluation needs to be scheduled in order to keep the modeling method and tool up-to-date and to retain its usability and acceptance by the domain experts as its users. The quality criteria defined in chapter 4.3.1 are a useful tool for guiding the re-evaluation.

When it comes to complexity assessment of the AAL-DSL, Table 16 summarizes the key findings for the respective model types. For the assessment, the formulas by [71] were modified as explained in chapter 4.3.1. The complexity within the table is denoted by the letter *C*. *Interface* represents the number of relations and constraints, but as there are no constraints defined within the AAL-DSL, merely the relations are counted. *Element* complexity is derived by the number of modeled elements within the respective model and

attribute complexity by the sum of attributes, which specify the elements and relations. The overall complexity is the sum of the three previous metrics.

Table 16 Complexity assessment of the AAL-DSL

– own representation

| Model type | $C_{interface}$ | $C_{element}$ | $C_{attributes}$ | $C_{overall}$ |
|----------------------------------|-----------------|---------------|------------------|---------------|
| Process landscape | 4 | 21 | 68 | 93 |
| L1 process model | 4 | 8 | 37 | 49 |
| L2 process model | 23 | 27 | 74 | 124 |
| L3 process model | 50 | 30 | 92 | 172 |
| Working environment model | 11 | 12 | 32 | 55 |
| Product structure model | 36 | 26 | 26 | 88 |

For counting the relations, elements, and attributes of the respective model types, the query functionality of the AAL-DSL was of use, as shown in Figure 55.

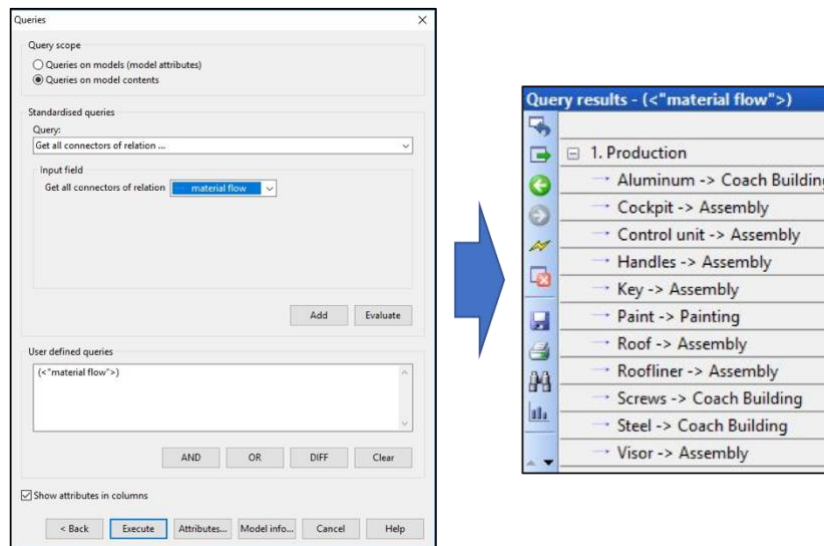
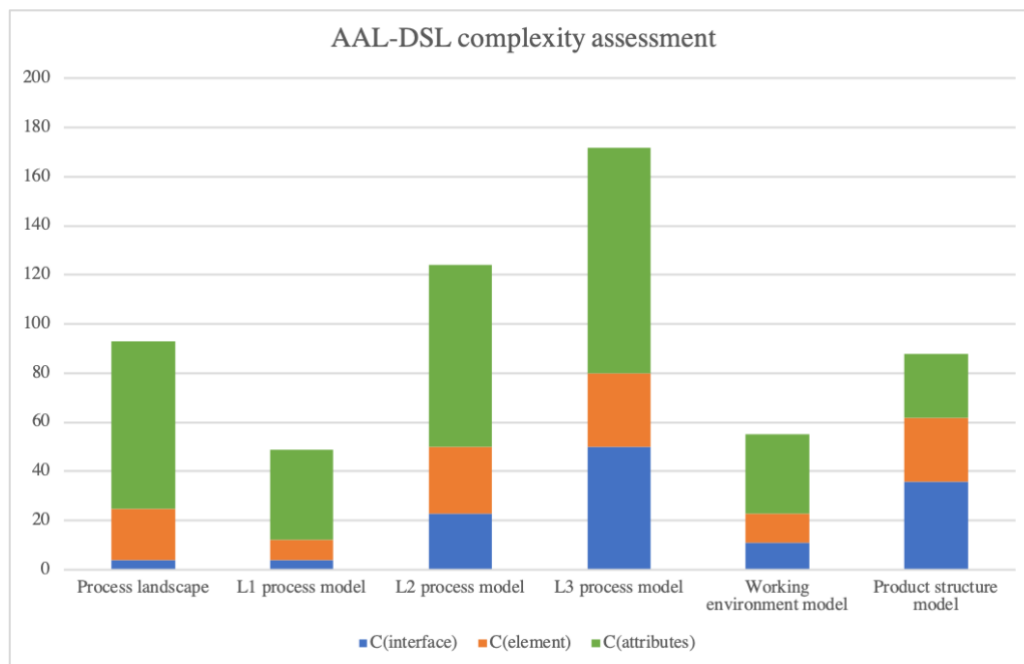


Figure 55 Queries for complexity assessment

- own representation

In the query dialogue, information on relations, elements, and attributes can be selected, either in combination or alone. The selection is then displayed in the results window, as on

the right side of Figure 55. The data collected in Table 16 can be seen in Figure 56 as a graphical representation.

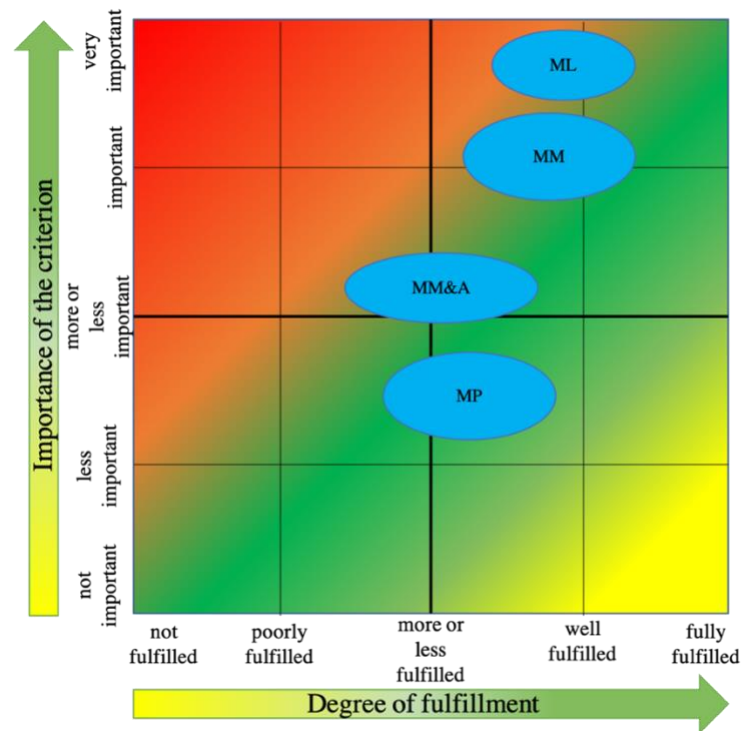


*Figure 56 Complexity assessment of the AAL-DSL
- own representation*

The figure reveals the overall complexity of each model type by the absolute height of the bar. C(interface) consists only of the number of relations within the model, as no constraints are defined. It can be seen, that the attribute complexity has the highest share within each model.

These analyses taken in isolation do not reveal insights into a rating of the AAL-DSL. An assessment gets possible in the presence of another DSBPM language, when complexity can be compared. Or, if changes are applied to the AAL-DSL in the course of time, the change in complexity can be evaluated.

Based on the requirements assessment by user stories (see chapter 5.1.2) and the resulting clusters *modeling language*, *modeling procedure*, *modeling mechanisms & algorithms*, and *modeling method*, Figure 57 shows a portfolio-diagram for quality assessment of the AAL-DSL.



*Figure 57 Quality assessment portfolio-diagram of the AAL-DSL
– own representation based on [37]*

The portfolio-diagram represents a qualitative grading of the four clusters regarding their importance and degree of fulfillment. The concepts of the modeling language including classes, relation classes, and attributes are seen to be very important and well fulfilled in the light of the requirements. The modeling method in general, including non-functional language requirements and requirements concerning the meta-modeling platform is also seen to be located in the green area of the diagram. Modeling mechanisms & algorithms are capable of expansion, as the main functionality at the moment comprises the analysis and simulation of times, costs, and capacities. As an extension, the addition of material usage and availability could constitute a valuable functionality. The modeling procedure is not seen as important as the other clusters due to the existence of only four model types. In case of a language extension, the modeling procedure might gain importance.

It is emphasized, that the establishment of a service center dedicated to managing and adjusting the modeling method is a crucial quality criterion. What is more, is the specifically expressed user-need for a service center, as communicated within the user stories of the requirements elicitation part (see chapter 5.1.2). This service center plays a central role within the re-evaluation of the modeling method. Of importance is, that regular points of re-

evaluation are scheduled on the one hand, and demand-based changes within the modeling method are enabled on the other hand.

5.4 Limitations of the Proof of Concept

In the previous chapters of the proof of concept section of this thesis, a domain-specific business process modeling method for automotive assembly line modeling was designed, implemented, and used. Following the Design Science Research approach introduced in chapter 1.3, the case study constitutes the proof of the DIF artifact. By completing this approach, the practical validity of the DIF is proven within the context of this thesis. The goal to derive a suitable modeling tool by applying the DIF was achieved. However, some final annotations regarding possible limitations of the proof of concept are stated in the following:

- **Limited domain knowledge**

Due to the restricted scope of this thesis regarding time and effort, it is not possible to grasp the exhaustive complexity of an automotive assembly line. In a complete project, several human resources would be required, and the involvement of domain-experts would be indisputable. The goal within this thesis was, to gain sufficient domain knowledge throughout the analysis phase in order to be able to demonstrate the DIF in a meaningful way.

- **Limited data**

The domain-specific data for the knowledge base within this thesis was collected from freely accessible resources. Among those were books, papers, websites of car manufacturers, and common knowledge of the author. However, especially the automotive industry is a highly protected one when it comes to best practices and process organization. Therefore, company-specific information was not readily accessible and could not be included.

- **Limited time**

Another limitation of the proof of concept is the timeframe within this thesis, which was possible for the implementation. As the proof of concept constitutes only one part within this master's thesis, the time-resources allowed a fast and sufficiently precise solution. This fact builds on the principles of agile software engineering, to build a working solution with every increment. The current state of the modeling tool can hence be seen as an interim result of the full solution.

6 Conclusion and Future Outlook

The work at hand contributes to the research field of domain-specific business process modeling. Based on a profound literature review, analysis techniques, and the Design Science Research methodology, the Domain Integration Framework (DIF) was developed. The DIF serves as a life-cycle model for systematic development of domain-specific business process modeling methods. In contrast to general-purpose business process modeling languages, domain-specific approaches require a profound understanding of the respective application domain in order to integrate domain-specific concepts into the BPM method. Even though this implies more work in the design phase, the resulting BPM tool is more likely to express the concepts specific to that domain and therefore increases acceptance among the language users.

The following SWOT analysis summarizes the internal strengths and weaknesses of the DIF as well as its external opportunities and threats. The analysis reflects the key learnings of deploying the DIF throughout the case study part of this thesis. The focus is set on the research objectives of this thesis (see chapter 1.2).

Strengths

Meta-modeling platforms like ADOxx enable flexible BPM language definition and maintenance and therefore constitute a powerful tool for domain-specific BPM. The DIF constitutes an artifact developed by following the Design Science Research methodology. Its applicability was tested on a case study on an automotive assembly line. A focus was set specifically on the use of agile methods in the field of IS. Examples of agile methods used within the DIF are the AMME approach and the formulation of user stories for requirements elicitation, as they are used in agile software development. The DIF can be seen as a life-cycle model for the creation of DSBPM languages as it shows a possible way to approach a modeling method design initiative for a specific application domain. Moreover, it includes the dynamic aspects of changing requirements and proposes quality criteria for testing the actuality of the modeling method. It serves as an orientation for the modeling method engineer on how to approach a DSBPM initiative. Throughout its different phases, it proposes methods and highlights focus points.

Weaknesses

Although the DIF can serve as a valuable guideline for domain-specific modeling method design and maintenance, it does not claim to have universal validity. This is due to its restricted proof of concept, which was done only for one domain application. Further application-tests are necessary to find out under which circumstances the DIF is valid and to examine its boundaries. Even though the DIF contributed to design a valid DSBPM method, a lack of real data and consultation possibilities with domain experts resemble a weakness. In order to improve the significance of the DIF, the cooperation with a real company would help to test its applicability within a real project.

Opportunities

The DIF can contribute to the business process modeling community as a life-cycle model, which provides guidance and a set of tools in order to include domain-specific concepts into the modeling language. Moreover, it can serve as a starting point for further investigation. Due to its modular structure, the tools within the phases could be modified or replaced by other methods, for instance the user-stories could be replaced with another method for requirements engineering. Moreover, the linguistic matching heuristic offers a potential for a more automated approach. A collection of existing language concepts could be provided for different domain applications, which would simplify the matching process.

Threats

Upon completion of this thesis, there also exist threats relating to the DIF. For one, the framework itself and the proof-of-concept are platform dependent, as all design-considerations and implementations were done on the ADOxx meta-modeling platform. The re-use potential of the DIF on other platforms is therefore questionable. Also, several literature sources regarding domain-specific language design exist, which are not tailored to BPM but provide their own design and implementation schemes. The DIF has to be positioned clearly as an instrument for DSBPM. What is more, the analysis of DSM languages as well as the linguistic matching heuristic are of a qualitative nature and highly rely on the knowledge and assumptions of the author.

To sum up the previous points, the DIF made a contribution towards a common understanding of domain-specific business process modeling and discloses a possibility to

approach design, implementation, modeling, and maintenance of DSBPM in a systematic way.

Future research in the scientific field of DSBPM is promising, as domain-specific approaches enable the inclusion of concepts, rules, and functionalities, which are valuable for a more adequate representation of the respective domain. This is enabled by meta-modeling, which lifts language definition to a higher level of abstraction. Here, future research can further investigate the opportunities of defining and integrating domain-specificity on the meta-layer. The work at hand has provided a first step towards considerations on that topic. Another focus of this research field is an analysis of further re-usability potential. Within this thesis, re-usability potential is seen in the model types as well as concepts of existing DSM languages. Further research can be done on the boundaries of this re-usability and a concept of a systematic approach towards it.

Literature

- [1] Agile Business Consortium. (2019, 06.02.2019). *DSDM*. Available: <https://www.agilebusiness.org/what-is-dsdm>
- [2] C. Aitken, C. Stephenson, and R. Brinkworth, "Process Classification Frameworks," in *Handbook on Business Process Management 2: Strategic Alignment, Governance, People and Culture*, J. vom Brocke and M. Rosemann, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 73-92.
- [3] J. Al-Marzougi, "Change Management: Process Classification Framework and Best Practices," Available at SSRN 2328978, 2013.
- [4] S. W. Ambler. (2018, 24.02.2019). *Agile Modeling*. Available: <http://www.agilemodeling.com/artifacts/userStory.htm>
- [5] R. Anzböck and S. Dustdar, "Modeling Medical E-services," in *Business Process Management*, Berlin, Heidelberg, 2004, pp. 49-65: Springer Berlin Heidelberg.
- [6] APICS. (2018, 15.02.2019). *Frameworks*. Available: <http://www.apics.org/apics-for-business/frameworks>
- [7] APQC, "APQC Process Classification Framework (PCF) - Cross Industry - PDF Version 7.2.0," p. 33, 2018.
- [8] APQC, "Automotive Process Classification Framework," vol. PCF® v7.0.5, 2018.
- [9] J. Barzdins *et al.*, "Domain specific languages for business process management: a case study," in *Proceedings of DSM*, 2009, vol. 9, pp. 34-40.
- [10] J. Becker, *Prozessmanagement*, 7., 7. korr. und erw. Aufl. 2013 Aufl. 2012 ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. XXVI, 687 S.
- [11] J. Becker, D. Pfeiffer, and M. Räckers, "Domain specific process modelling in public administrations—the PICTURE-approach," in *International Conference on Electronic Government*, 2007, pp. 68-79: Springer.
- [12] J. Becker, B. Weiss, and A. Winkelmann, "Developing a business process modeling language for the banking sector-a design science approach," *AMCIS 2009 Proceedings*, p. 709, 2009.
- [13] A. Berg *et al.*, *PG 582-Industrial Programming by Example*. Universitätsbibliothek Dortmund, 2015.
- [14] J. Bicevskis, J. Cerina-Berzina, G. Karnitis, L. Lace, I. Medvedis, and S. Nesterovs, "Domain Specific Business Process Modeling in Practice," in *Proceedings of the 9th International Baltic Conference on Databases and Information Systems (Baltic B&IS '2010)*, Riga, Latvia, 2010, pp. 5-7.
- [15] BOC Asset Management GmbH. (17.04.2019). *Industrial Business Process Management*. Available: <https://www.adoxx.org/live/web/disrupt/industrial-business-process-management>
- [16] BOC Asset Management GmbH. (2018, 13.05.2019). *ADOxx—official website*. Available: <https://www.adoxx.org/live/home>

- [17] J. Bortz and N. Döring, *Forschungsmethoden und Evaluation*, Zweite, vollständig überarbeitete und aktualisierte Auflage ed. (Springer-Lehrbuch). Berlin Heidelberg New York [NY]: Springer, 1995, pp. XV, 768 Seiten.
- [18] H. Breitling and S. Hofer, "Schwerpunkt-beispielhaft gut modelliert: Exemplarische Geschäftsprozessmodellierung in der Praxis," *Objekt Spektrum*, no. 6, p. 8, 2012.
- [19] T. Bucher and R. Winter, "Taxonomy of business process management approaches," in *Handbook on Business Process Management 2*: Springer, 2010, pp. 93-114.
- [20] R. A. Buchmann and D. Karagiannis, "Agile Modelling Method Engineering: Lessons Learned in the ComVantage Research Project," in *The Practice of Enterprise Modeling: 8th IFIP WG 8.1. Working Conference, PoEM 2015, Valencia, Spain, November 10-12, 2015, Proceedings*, 2015, vol. 235, p. 356: Springer.
- [21] P. P.-S. Chen, "The entity-relationship model—toward a unified view of data," *ACM Transactions on Database Systems (TODS)*, vol. 1, no. 1, pp. 9-36, 1976.
- [22] M. Cohn, *User stories applied: For agile software development*. Addison-Wesley Professional, 2004.
- [23] Collins English Dictionary. (2019, 13.04.2019). *Definition of 'domain'*. Available: <https://www.collinsdictionary.com/dictionary/english/domain>
- [24] H. M. Cooper, "Organizing knowledge syntheses: A taxonomy of literature reviews," *Knowledge in society*, vol. 1, no. 1, p. 104, 1988.
- [25] B. Curtis, M. I. Kellner, and J. Over, "Process modeling," *Communications of the ACM*, vol. 35, no. 9, pp. 75-90, 1992.
- [26] T. H. Davenport and J. E. Short, "The new industrial engineering: information technology and business process redesign," 1990.
- [27] T. De Bruin and M. Rosemann, "Towards a business process management maturity model," 2005.
- [28] R. Deshayes, B. Meyers, T. Mens, and H. Vangheluwe, "ProMoBox in Practice: A Case Study on the GISMO Domain-Specific Modelling Language," in *MPM@ MoDELS*, 2014, pp. 21-30: Citeseer.
- [29] I. DeToro and T. McCabe, "How to stay flexible and elude fads," *Quality Progress*, vol. 30, no. 3, p. 55, 1997.
- [30] A. v. Deursen, P. Klint, and J. Visser, "Domain-specific languages: an annotated bibliography," *SIGPLAN Not.*, vol. 35, no. 6, pp. 26-36, 2000.
- [31] Dreamstime. (2019, 12.05.2019). *Developer, application, programming icon vector image*. Available: <https://www.dreamstime.com/stock-illustration-software-developer-application-programming-icon-vector-image-can-also-be-used-development-suitable-use-web-apps-mobile-image78730067>
- [32] M. Dumas, M. La Rosa, J. Mendling, and H. A. Reijers, *Fundamentals of business process management*. Heidelberg: Springer, 2013.

- [33] G. Engels, J. H. Hausmann, R. Heckel, and S. Sauer, "Dynamic meta modeling: A graphical approach to the operational semantics of behavioral diagrams in UML," in *International Conference on the Unified Modeling Language*, 2000, pp. 323-337: Springer.
- [34] H.-G. Fill and D. Karagiannis, "On the conceptualisation of modelling methods using the ADOxx meta modelling platform," *Enterprise Modelling and Information Systems Architectures–International Journal of Conceptual Modeling*, vol. 8, no. 1, pp. 4-25, 2013.
- [35] U. Frank and C. Lange, "E-MEMO: a method to support the development of customized electronic commerce systems," *Information Systems and E-Business Management*, vol. 5, no. 2, pp. 93-116, 2007.
- [36] U. Frank and B. van Laak, "Anforderungen an Sprachen zur Modellierung von Geschäftsprozessen," *Arbeitsberichte des Instituts für Wirtschaftsinformatik*, 2003.
- [37] U. Frank and B. Van Laak, "Ein Bezugsrahmen zur Evaluation von Sprachen zur Modellierung von Geschäftsprozessen," *Arbeitsberichte des Instituts für Wirtschaftsinformatik Universität Koblenz-Landau (2003), H*, vol. 36, 2003.
- [38] Free Icons. (2019, 12.05.2019). *Copy - Free Web Icon*. Available: <https://icons8.com/icon/29/copy>
- [39] Free Icons Library. (2019, 12.05.2019). *Create Icon #228973*. Available: <http://chittagongit.com/icon/create-icon-14.html>
- [40] D. Götzinger, E.-T. Miron, and F. Staffel, "OMiLAB: an open collaborative environment for modeling method engineering," in *Domain-Specific Conceptual Modeling*: Springer, 2016, pp. 55-76.
- [41] M. Hammer, "The process audit," *Harvard business review*, vol. 85, no. 4, pp. 111-9, 122-3, 142, 2007.
- [42] M. Hammer and S. Stanton, "The reengineering revolution," HARPER BUSINESS 1995.
- [43] A. E. Haxthausen and J. Peleska, "A domain specific language for railway control systems," in *Proceedings of the sixth biennial world conference on integrated design and process technology, (IDPT2002), Pasadena, California*, 2002, pp. 23-28.
- [44] X. He, Z. Ma, W. Shao, and G. Li, "A metamodel for the notation of graphical modeling languages," in *31st Annual International Computer Software and Applications Conference (COMPSAC 2007)*, 2007, vol. 1, pp. 219-224: IEEE.
- [45] H. Heitkötter, "A Framework for Creating Domain-specific Process Modeling Languages," in *ICSOF*, 2012, pp. 127-136.
- [46] A. R. Hevner, S. T. March, J. Park, and S. Ram, "Design science in information systems research," *Management Information Systems Quarterly*, vol. 28, no. 1, pp. 57-106, 2004.
- [47] A. R. Hevner, S. T. March, J. Park, and S. Ram, "Design science in information systems research," *Management Information Systems Quarterly*, vol. 28, no. 1, p. 6, 2008.
- [48] IBM Knowledge Center. (2011, 13.02.2019). *eTOM process classification framework*. Available: https://www.ibm.com/support/knowledgecenter/en/SSFTDH_7.5.0/com.ibm.ws.icp.telopr.doc/tel/opr/opdev/concept/ci/indstd/c_capmdl_etom.html

- [49] Icon Shop. (2019, 12.05.2019). *Filter Icon Outline*. Available: <http://freeiconshop.com/icon/filter-icon-outline/>
- [50] International Standards Association. (1994, 14.04.2019). *Industrial automation systems and integration - Product data representation and exchange - Part 11: Description methods: The EXPRESS language reference manual (ISO 10303-11:2004 ed.)*. Available: <https://www.iso.org/standard/38047.html>
- [51] S. Junginger, "Modellierung von Geschäftsprozessen - State-of-the-Art, neuere Entwicklungen und Forschungspotenziale," in "BPMS-Bericht," University of Vienna2000.
- [52] D. Karagiannis, "Agile modeling method engineering," presented at the Proceedings of the 19th Panhellenic Conference on Informatics, 2015.
- [53] D. Karagiannis, R. Buchmann, P. Burzynski, and J. Brakmic, "D3. 1.2—specification of modelling method including conceptualisation outline, comvantage public deliverables," ed.
- [54] D. Karagiannis, R. A. Buchmann, P. Burzynski, U. Reimer, and M. Walch, "Fundamental conceptual modeling languages in OMiLAB," in *Domain-Specific Conceptual Modeling: Concepts, Methods and Tools*, 2016, pp. 3-30.
- [55] D. Karagiannis and P. Höfferer, "Metamodeling as an integration concept," in *International Conference on Software and Data Technologies*, 2006, pp. 37-50: Springer.
- [56] D. Karagiannis and H. Kühn, "Metamodelling platforms," in *EC-Web*, 2002, vol. 2455, p. 182.
- [57] D. Karagiannis, H. C. Mayr, and J. P. Mylopoulos, *Domain-specific conceptual modeling concepts, methods and tools*. Cham: Springer International Publishing AG, 2016, pp. xii, 594 Seiten.
- [58] D. Karagiannis and R. Woitsch, "Knowledge engineering in business process management," in *Handbook on Business Process Management 2*: Springer, 2010, pp. 463-485.
- [59] G. Keller, A.-W. Scheer, and M. Nüttgens, *Semantische Prozeßmodellierung auf der Grundlage" Ereignisgesteuerter Prozeßketten (EPK)"*. Inst. für Wirtschaftsinformatik, 1992.
- [60] S. Kelly and J.-P. Tolvanen, *Domain-specific modeling: enabling full code generation*. John Wiley & Sons, 2008.
- [61] H. Kern, A. Hummel, S. K, #252, and hne, "Towards a comparative analysis of meta-metamodels," presented at the Proceedings of the compilation of the co-located workshops on DSM'11, TMC'11, AGERE! 2011, AOOPEs'11, NEAT'11, & VMIL'11, Portland, Oregon, USA, 2011.
- [62] E. Kindler, B. Axenath, and V. Rubin, "AMFIBIA: a meta-model for the integration of business process modelling aspects," in *Dagstuhl Seminar Proceedings*, 2006: Schloss Dagstuhl-Leibniz-Zentrum für Informatik.
- [63] kisspng. (2019, 12.05.2019). *Mechanical Engineering, Engineering, Computer Icons, Black, Black And White PNG*. Available: <https://www.kisspng.com/png-mechanical-engineering-clip-art-computer-icons-col-5998537/>
- [64] A. Kleppe, *Software language engineering: creating domain-specific languages using metamodels*. Pearson Education, 2008.
- [65] C. R. Kothari, *Research methodology: Methods and techniques*. New Age International, 2004.

- [66] D. Kühn and J. Neubauer, "Guided domain-specific tailoring of jABC4," in *International Symposium on Leveraging Applications of Formal Methods*, 2016, pp. 113-127: Springer.
- [67] H. Kühn, *Methodenintegration im Business Engineering*. 2004.
- [68] S. Laghrabli, L. Benabbou, and A. Berrado, "Strategic decision processes classification framework using UTADIS," in *2016 11th International Conference on Intelligent Systems: Theories and Applications (SITA)*, 2016, pp. 1-6.
- [69] A. Lapouchnian, "Goal-oriented requirements engineering: An overview of the current research," *University of Toronto*, vol. 32, 2005.
- [70] A. Leitner, C. Preschern, and C. Kreiner, "Effective development of automation systems through domain-specific modeling in a small enterprise context," *Software & Systems Modeling*, 2014, Vol.13(1), pp.35-54, p. 35.
- [71] A. Leitner, R. Weiß, and C. Kreiner, "Analyzing the complexity of domain model representations," in *Engineering of Computer Based Systems (ECBS), 2012 IEEE 19th International Conference and Workshops on*, 2012, pp. 242-248: IEEE.
- [72] B. List and B. Korherr, "An evaluation of conceptual business process modelling languages," in *Proceedings of the 2006 ACM symposium on Applied computing*, 2006, pp. 1532-1539: ACM.
- [73] B. Lotter and H.-P. Wiendahl, "Montage in der industriellen Produktion. Ein Handbuch für die Praxis. 2., Aufl," ed: Berlin: Springer Berlin (VDI-Buch), 2012.
- [74] R. Lu and S. Sadiq, "A survey of comparative business process modeling approaches," in *International Conference on Business Information Systems*, 2007, pp. 82-94: Springer.
- [75] Markets and Markets. (2016, 13.05.2019). *Business Process Management Market*. Available: <https://www.marketsandmarkets.com/Market-Reports/business-process-management-market-157890056.html>
- [76] S. Mary, "Risk factors in enterprise-wide/ERP projects," *Journal of Information Technology*, vol. 15, no. 4, p. 317, 2000.
- [77] L. McIver and D. Conway, "Seven deadly sins of introductory programming language design," in *seep*, 1996, p. 309: IEEE.
- [78] N. Melão and M. Pidd, "A conceptual framework for understanding business processes and business process modelling," *Information systems journal*, vol. 10, no. 2, pp. 105-129, 2000.
- [79] M. Mernik, J. Heering, and A. M. Sloane, "When and how to develop domain-specific languages," *ACM computing surveys (CSUR)*, vol. 37, no. 4, pp. 316-344, 2005.
- [80] MetaCase. (2019, 07.02.2019). *MetaEdit+*. Available: <https://www.metacase.com/products.html>
- [81] L. H. Nastansky, W.; Ott, M.; Riempp, G., "Die Produktivität von Groupware-basierten Anwendungen: Geschäftsprozeßorientierte Modellierung & Workflow Management," *Information Management, Workgroup Computing, Office Systems*1995.
- [82] M. Nikolaidou, D. Anagnostopoulos, and A. Tsalgaidou, *Business processes modelling and automation in the banking sector: A case study*. 2008.

- [83] Noun Project Inc. (2019, 12.05.2019). *Extend icon*. Available: <https://thenounproject.com/term/extend/424331/>
- [84] B. Nuseibeh and S. Easterbrook, "Requirements engineering: a roadmap," in *Proceedings of the Conference on the Future of Software Engineering*, 2000, pp. 35-46: ACM.
- [85] OMG. (2014, 09.04.2019). *About the Object Constraint Language Specification Version 2.4*. Available: <https://www.omg.org/spec/OCL/About-OCL/>
- [86] OMG. (2019, 09.04.2019). *Business Process Model and Notation*. Available: <http://www.bpmn.org>
- [87] OMiLAB Europe. (2018, 20.10.2018). *Ideas and objective*. Available: <http://austria.omilab.org/psm/about>
- [88] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee, "A design science research methodology for information systems research," *Journal of management information systems*, vol. 24, no. 3, pp. 45-77, 2007.
- [89] S. Pissierssens, "Revealing the scientific basis of graphical representation design."
- [90] K. Pohl, *Requirements Engineering: Fundamentals, Principles, and Techniques*. Springer Publishing Company, Incorporated, 2010, p. 813.
- [91] K. Pohl and C. Rupp, *Basiswissen Requirements Engineering: Aus-und Weiterbildung nach IREB-Standard zum Certified Professional for Requirements Engineering Foundation Level*. dpunkt.verlag, 2015.
- [92] Porsche Leipzig GmbH. (2018, 28.10.2018). *Die Montage*. Available: <https://www.porsche-leipzig.com/produktion/montage/>
- [93] M. E. Porter, "Competitive advantage: creating and sustaining superior performance. 1985," *New York: FreePress*, vol. 43, p. 214, 1985.
- [94] C. Prackwieser, "SIMchronization: a method supporting the synchronisation of information and material flows," in *Proceedings of the Winter Simulation Conference*, 2012, p. 355: Winter Simulation Conference.
- [95] C. Preschern, "PISCAS-A Pisciculture Automation System Product Line," 2011.
- [96] R. Prieto-Díaz, "Domain analysis: An introduction," *ACM SIGSOFT Software Engineering Notes*, vol. 15, no. 2, pp. 47-54, 1990.
- [97] R. L. Raschke, "Process-based view of agility: The value contribution of IT and the effects on process outcomes," *International Journal of Accounting Information Systems*, vol. 11, no. 4, pp. 297-313, 2010.
- [98] P. Ravesteyn and S. Jansen, "A situational implementation method for business process management systems," *AMCIS 2009 Proceedings*, p. 632, 2009.
- [99] K. Ravlani. (2017, 24.02.2019). *Agile for Growth*. Available: <http://agileforgrowth.com/blog/userstory-benefits/>
- [100] S. Ray, G. Karsai, and K. M. McNeill, "Model-based adaptation of flight-critical systems," in *2009 IEEE/AIAA 28th Digital Avionics Systems Conference*, 2009, pp. 6.B.6-1-6.B.6-16.

- [101] A. Roques. (2012, 09.04.2019). *PlantUML in a nutshell*. Available: <http://plantuml.com/index>
- [102] M. Rosemann, "The service portfolio of a BPM center of excellence," in *Handbook on Business Process Management 2*: Springer, 2010, pp. 267-284.
- [103] J. Rowley and F. Slack, "Conducting a literature review," *Management research news*, vol. 27, no. 6, pp. 31-39, 2004.
- [104] A. Ruffner, "Wissenschaftstheoretische Überlegungen zur Betriebswirtschaftlichen Organisationslehre," *Dlugos, G. ua, Hrsg., Wissenschaftstheorie und Betriebswirtschaftslehre, Düsseldorf*, pp. 185-207, 1972.
- [105] J. Rumbaugh, I. Jacobson, and G. Booch, *Unified modeling language reference manual, the*. Pearson Higher Education, 2004.
- [106] W. Sadiq and M. E. Orlowska, "On capturing process requirements of workflow based business information systems," in *BIS'99*: Springer, 1999, pp. 281-294.
- [107] O. Saidani and S. Nurcan, "Towards Situational Business Process Meta-Modelling," in *CAiSE Forum*, 2008, pp. 93-96.
- [108] H. J. Schmelzer and W. Sesselmann, "Geschäftsprozessmanagement in der Praxis," *Kunden zufrieden stellen-Produktivität steigern-Wert erhöhen*, vol. 6, pp. 1-2, 2008.
- [109] A. Schoknecht, A. Vetter, H.-G. Fill, and A. Oberweis, "Using the Horus Method for Succeeding in Business Process Engineering Projects," in *Domain-Specific Conceptual Modeling*: Springer, 2016, pp. 127-147.
- [110] B. Scholz-Reiter, A. Nethe, and H. Stahlmann, "Process Modelling," 1999.
- [111] Scrum.org. (2019, 06.02.2019). *The home of Scrum*. Available: <https://www.scrum.org/>
- [112] P. M. Shields and N. Rangarajan, *A playbook for research methods: Integrating conceptual frameworks and project management*. New Forums Press, 2013.
- [113] H. Stachowiak, "Allgemeine modelltheorie," 1973.
- [114] S. Strahinger, "Metamodellierung als Instrument des Methodenvergleichs: Eine Evaluierung am Beispiel objektorientierter Analysenmethoden," Darmstadt Technical University, Department of Business Administration, Economics and Law, Institute for Business Studies (BWL)1996.
- [115] C. Thiemich and F. Puhlmann, "An agile BPM project methodology," in *Business Process Management*: Springer, 2013, pp. 291-306.
- [116] P. Uhnak. (2018, 07.02.2019). *OpenPonk (meta)modeling platform*. Available: <https://modeling-languages.com/openponk-metamodeling-platform/>
- [117] N. Visic, H.-G. Fill, R. A. Buchmann, and D. Karagiannis, "A domain-specific language for modeling method definition: From requirements to grammar," in *Research Challenges in Information Science (RCIS), 2015 IEEE 9th International Conference on*, 2015, pp. 286-297: IEEE.
- [118] J. Vom Brocke, A. Simons, B. Niehaves, K. Riemer, R. Plattfaut, and A. Cleven, "Reconstructing the giant: On the importance of rigour in documenting the literature search process," in *Ecis*, 2009, vol. 9, pp. 2206-2217.

- [119] M. von Rosing, N. Kemp, M. Hove, and J. W. Ross, "Process Tagging-A Process Classification and Categorization Concept," ed, 2015.
- [120] B. Webster, "Buy vs. build software applications: The eternal dilemma," 2008.
- [121] B. Webster. (2008, 13.05.2019). *Buy vs. build software applications: The eternal dilemma*. Available: <http://www.baselinemag.com/c/a/Application-Development/Buy-vs-Build-Software-Applications-The-Eternal-Dilemma>
- [122] R. Winter, "Design solution analysis for the construction of situational design methods," in *Engineering Methods in the Service-Oriented Context*: Springer, 2011, pp. 19-33.
- [123] G. Wolters, "Jürgen Mittelstrass (Hrsg.): Enzyklopädie Philosophie und Wissenschaftstheorie. 1984."
- [124] M. Zacarias, P. V. Martins, and A. Gonçalves, "An Agile Business Process and Practice Meta-model," *Procedia Computer Science*, vol. 121, pp. 170-177, 2017.
- [125] M. Zairi and D. Sinclair, "Business process re-engineering and process management: a survey of current practice and future trends in integrated management," *Business Process Re-engineering & Management Journal*, vol. 1, no. 1, pp. 8-30, 1995.
- [126] M. D. Zisman, "Representation, Specification and automation of office procedures," 1977.
- [127] M. zur Muehlen and R. Shapiro, "Business process analytics," in *Handbook on Business Process Management 2*: Springer, 2010, pp. 137-157.

Appendix A

The quality criteria suggested by [37] are the following:

| Criterion | Manifestation |
|---|--|
| General criteria | |
| Formal criteria | |
| Correctness and completeness | Syntactically unambiguous identification of incorrect models |
| | Semantically unambiguous identification of incorrect models |
| | It is possible to model all needed models with the existing language resources |
| Uniformity and non-redundancy | Similar representation of similar concepts within the language |
| | Information does not need to be filed redundantly within the model |
| Re-usability and maintainability | Processes can be combined into classes |
| | Ability to hide information |
| | Generalization and specialization is possible |
| User-oriented criteria | |
| Simplicity | The model is not overloaded, i.e. is uses exactly the needed number of symbols |
| | Little rules are sufficient for language-use |
| Comprehensibility and clarity | The terminology used within the ML corresponds to the domain-specific concepts of the application domain |
| Usage-oriented criteria | |
| Powerfulness and adequacy | All relevant aspects are depictable with the ML in a sufficiently detailed way |
| | The user is not forced to model or read unnecessary information |
| Operationalization | The ML uses concepts relevant to the software context, e.g. object-orientation |
| | Ability to generate workflow-schemes from the model |
| | Possibility to annotate data relevant to the business context |
| | The language provides concepts for the creation of simulation models |
| BPM-specific criteria | |
| General criteria for BPML | |
| Abstraction-levels | Single process- and resource- instances can be distinguished |
| | Process-types can be modeled (intentional class-concept) |
| | Different process-instances can be summarized into sets (extensional class-concept) |
| Flexibility and adaptability | The language can be extended by stereotypes |
| | An amount of process-types is already provided |
| | New language elements can be added via meta-language |
| Support of different views | Processes can be decomposed and the decomposition can be depicted graphically |
| | Relations between process-types can be depicted |

| | |
|--|---|
| | Processes are depictable in different degrees of detail |
| General concepts for BPM | |
| Processes | Criteria affecting the process-start can be modeled |
| | Results existing after the process-end can be modeled |
| | The run-time duration of the process can be defined |
| | Costs can be mapped to the process |
| | If a process is not further decomposable, text annotations for sufficient description can be added |
| | Formal specification of a non-decomposable process function is possible |
| | Critical success-factors can be mapped to the process |
| | Processes can be aggregated and the cardinality specified in this context |
| | Through a “used”-relation it can be shown, which processes are needed for a correct processing of a certain process |
| | Similarity-relations are used to identify similar processes |
| | There exists a concept for identification of process-instances |
| Events | The concept of “event” is supported by the ML |
| | There is a predefined number of event-types, e.g. different temporal events |
| | Different events can be connected logically to each other |
| | There can be specialized relationships between event- types |
| | Like with processes, similarity-relations can be depicted |
| | Additional annotations to event-types are allowed (formal, semi-formal, non-formal). For formal annotations, an appropriate language is available |
| Modeling of business concepts | |
| Goal-modeling | Goals can be mapped to processes in a natural way |
| | Process-goals are specifiable also semi-formally |
| | Different goal-typed are predefined and accessible via goal-categorization |
| | Goal-types can be user-defined |
| | Relations between goals can be specified, e.g. independency, concurrency, complementarity, contradiction, and “is upper-level goal” |
| | Relations between goals can be defined according to the context |
| | Goals can have a state (degree of goal attainment), for which a calculated function can be defined |
| Resource modeling | From the process-model associations are possible to the resource-model |
| | Different categories of resources are already defined |
| | Resource-types can be user-defined |
| | Costs resulting from resource-usage can be mapped to the resource in different ways, e.g. per time-unit, per unit of material |
| | Resources can have correlating relations, e.g. substitutional, usage, specialization, aggregation |
| Modeling of the static organization | The static organization can be modeled in the ML |

| | |
|--------------------------------------|---|
| | There exist different relationship-types between organizational entities, e.g. “is entitled to issue instructions”, “is part of” |
| | Relations between entities cannot only be depicted 1:1 and 1:n, but also n:m |
| | Single posts can be described via profiles |
| | There exist pre-defined types of organizational entities, e.g. main department, department, group, post |
| | The graphical representation of the organization corresponds to the official organigrams |
| Modeling of roles | Complementary to organizational-units, executing roles can be mapped to processes |
| | Roles can be associated to organizational entities of the static organizational model |
| Interorganizational processes | Sub-processes, which are undertaken by external entities, can be labeled as such |
| | The PML enables the modeling of interfaces and protocols |
| | Standard interfaces and protocols already exist |
| | For logistical processes, the means of transportation can be defined |
| | Specific exceptions can be modeled and processes for exception-handling be defined, e.g. breakdown of a communication link |
| Control structures | |
| Sequence | Any number of process parts can be arranged in a linear order. Their succession is represented by appropriate graphical representation, e.g. arrows |
| Conditions and rules | The language allows to depict process procedures by the definition of “if, then” rules |
| | For the definition of conditions, a formal language can be used |
| Alternative sequences | A process can be split into any number of alternative paths |
| | To each alternative path, probabilities can be added in order to specify the probability, with which the alternative is taken within the process |
| Parallelism | Real-parallel sequences can be defined within the process |
| Concurrency | Concurrent processes can be defined within the process |
| | There is an explicit distinction between parallelism and concurrency |
| Abstraction of sequences | A sequence of sub-processes can but does not have to be defined. Instead, it is allowed to name several sub-processes, of which one or any number can run in any sequence |
| Repetitions | Process-parts can be run repetitively (iteration) |
| | The number of iterations can be set by a number |
| | The number of iterations can be set by rules |
| Synchronization of processes | It can be defined that a process can only start once one or several other processes are completed |
| | It can be defined whether the still running processes in the above case terminate or not |
| Transactions | A process can be labelled as a transaction |

| Exceptions | |
|--|--|
| Processes can be associated with exceptions | |
| To every exception-type, a counter-measure can be associated, e.g. in the form of another process | |
| Integrity conditions | |
| Cardinalities | For all relations within the model, cardinalities can be defined |
| | For depicting cardinalities, a min-max-notation is used |
| Pre-conditions | The language allows a specification of pre-conditions, which have to be fulfilled in order to start the process |
| | An exception-type can be added |
| Post-conditions | Post-conditions state, which conditions have to be fulfilled after the completion of the process |
| | An exception-type can be added |
| Process-type invariants | The language allows a definition of process-type invariants |
| | An exception-type can be added |
| Support of the development of information systems | |
| Integration with IT abstractions | Entities of an associated data model can be referenced from the process model |
| | Objects, classes, attributes, and methods from an associated object-model can be referenced from the process model |
| Support of the usage of WfMS | Software applications and editable data are allocated to the individual processes as needed |
| | The process-specification should be exportable in a format, which can be read by WfMS |
| Support of individual adjustments | |
| The symbols used within the language are exchangeable | |
| Libraries of domain-specific symbols exist | |
| Criteria for the evaluation of learnability of the BPML | |
| Documentation | |
| General criteria | The ML is prepared in a didactically appropriate way |
| | The documentation contains all language-symbols as well as all possible syntactical constructions |
| | The use of the language is demonstrated by examples |
| Differentiation between different user-groups | There exist different pieces of documentation for different user-groups, e.g. software engineers, organizers |
| | For each user-group, goals are specified, which can be reached by the use of the language |
| | Also disadvantages and exemplary problem-cases are demonstrated |
| | The style of the specific user-group documentation corresponds to the linguistic style of this user-group |
| Specification | |

| | |
|---|--|
| Specification of semantics and abstract syntax | The abstract syntax of a language is formally described either by specifying a grammar or a meta-model |
| Specification of notation | The used symbols precisely match the concepts, which the language uses |
| | A number of conventions for the naming of identifiers and the usage of additional textual elements is formally specified |
| Embedding of a ML into a MM | |
| General | The ML is embedded in a MM |
| Project-specific roles and resources | The method provides a commented list of roles and requirement-profiles |
| | The method includes propositions for communication-relations within the modeling project |
| | The method provides a catalogue of quality assurance measures |
| | The method includes a component for the management of project-resources |
| Procedure model | A procedure model structures the project into manageable sub-tasks |
| | For each sub-task, critical success-factors, roles involved, communication relations, and expected outcomes are defined |
| | Depending on the modeling-purpose, the procedure model contains several variants |
| | The usage of the procedure model is demonstrated by examples |

Appendix B

__D_construct__ (Metamodel)
__D_event__ (Metamodel)
__D_variable_assignment_object__ (Metamodel)
__Neutral_element__ (Metamodel)
AnimRep (Metamodel) STRING (Short string)
AttrRep (Metamodel) LONGSTRING (Long string)
Class cardinality (Metamodel) STRING (Short string)
ClassAbstract INTEGER (Integer)
ClassName STRING (Short string)
ClassVisibleINTEGER (Integer)
External tool coupling (Metamodel) STRING (Short string)
GraphRep (Metamodel) LONGSTRING (Long string)
HlpTxt (Metamodel) STRING (Short string)
Model pointer (Metamodel) STRING (Short string)
Position (Metamodel) STRING (Short string)
VisibleAttrs (Metamodel) STRING (Short string)
WF_Trans (Metamodel) STRING (Short string)
__Start__ (Metamodel)
Start event
Abandon after tolerance waiting time (Metamodel) ENUMERATION (Enumeration)
AnimRep (Metamodel) STRING (Short string)
AttrRep (Metamodel) LONGSTRING (Long string)
Class cardinality (Metamodel) STRING (Short string)
ClassAbstract INTEGER (Integer)
ClassNameSTRING (Short string)
ClassVisible INTEGER (Integer)
External tool coupling (Metamodel) STRING (Short string)
GraphRep (Metamodel) LONGSTRING (Long string)
HlpTxt (Metamodel) STRING (Short string)
Info on results STRING (Short string)
Model pointer (Metamodel) STRING (Short string)
Position (Metamodel) STRING (Short string)
Process calendar (Metamodel) LONGSTRING (Long string)
Quantity (Metamodel) EXPRESSION (Expression)
Time period (Metamodel) ENUMERATION (Enumeration)
Tolerance waiting time (Metamodel) TIME (Time)
VisibleAttrs (Metamodel) STRING (Short string)
WF_Trans (Metamodel) STRING (Short string)
Abandon after tolerance waiting time (Metamodel) ENUMERATION (Enumeration)
AnimRep (Metamodel) STRING (Short string)
AttrRep (Metamodel) LONGSTRING (Long string)
Class cardinality (Metamodel) STRING (Short string)
ClassAbstract INTEGER (Integer)
ClassName STRING (Short string)
ClassVisibleINTEGER (Integer)
External tool coupling (Metamodel) STRING (Short string)
GraphRep (Metamodel) LONGSTRING (Long string)
HlpTxt (Metamodel) STRING (Short string)
Model pointer (Metamodel) STRING (Short string)
Position (Metamodel) STRING (Short string)
Process calendar (Metamodel) LONGSTRING (Long string)
Quantity (Metamodel) EXPRESSION (Expression)
Time period (Metamodel) ENUMERATION (Enumeration)
Tolerance waiting time (Metamodel) TIME (Time)
VisibleAttrs (Metamodel) STRING (Short string)
WF_Trans (Metamodel) STRING (Short string)
__Subgraph__ (Metamodel)

Sub-Process

__Conversion__ LONGSTRING (Long string)
Aggregated costs EXPRESSION (Expression)
Aggregated execution time EXPRESSION (Expression)
Aggregated resting time EXPRESSION (Expression)
Aggregated transport time EXPRESSION (Expression)
Aggregated waiting time EXPRESSION (Expression)
AnimRep (Metamodel) STRING (Short string)
AttrRep (Metamodel) LONGSTRING (Long string)
Class cardinality (Metamodel) STRING (Short string)
ClassAbstract INTEGER (Integer)
ClassNameSTRING (Short string)
ClassVisible INTEGER (Integer)
Description STRING (Short string)
External tool coupling (Metamodel) STRING (Short string)
GraphRep (Metamodel) LONGSTRING (Long string)
HlpTxt (Metamodel) STRING (Short string)
Model pointer (Metamodel) STRING (Short string)
Order INTEGER (Integer)
Position (Metamodel) STRING (Short string)
Referenced subprocess (Metamodel) INTERREF (Inter-model reference)
VisibleAttrs (Metamodel) STRING (Short string)
WF_Trans (Metamodel) STRING (Short string)
AnimRep (Metamodel) STRING (Short string)
AttrRep (Metamodel) LONGSTRING (Long string)
Class cardinality (Metamodel) STRING (Short string)
ClassAbstract INTEGER (Integer)
ClassName STRING (Short string)
ClassVisibleINTEGER (Integer)
External tool coupling (Metamodel) STRING (Short string)
GraphRep (Metamodel) LONGSTRING (Long string)
HlpTxt (Metamodel) STRING (Short string)
Model pointer (Metamodel) STRING (Short string)
Position (Metamodel) STRING (Short string)
Referenced subprocess (Metamodel) INTERREF (Inter-model reference)
VisibleAttrs (Metamodel) STRING (Short string)
WF_Trans (Metamodel) STRING (Short string)
__Activity__ (Metamodel)

Task

Aggregated costs DOUBLE (Floating-point number)
AnimRep (Metamodel) STRING (Short string)
AttrRep (Metamodel) LONGSTRING (Long string)
Average number of participants (Metamodel) INTEGER (Integer)
Class cardinality (Metamodel) STRING (Short string)
ClassAbstract INTEGER (Integer)
ClassNameSTRING (Short string)
ClassVisible INTEGER (Integer)
Continuous execution (Metamodel) ENUMERATION (Enumeration)
Cooperation mode (Metamodel) ENUMERATION (Enumeration)
Cooperative (Metamodel) ENUMERATION (Enumeration)
Costs DOUBLE (Floating-point number)
Done by (Metamodel) STRING (Short string)
Execution interruptable (Metamodel) ENUMERATION (Enumeration)
Execution time (Metamodel) TIME (Time)
External tool coupling (Metamodel) STRING (Short string)
GraphRep (Metamodel) LONGSTRING (Long string)
HlpTxt (Metamodel) STRING (Short string)
Max. resource waiting time (Metamodel) TIME (Time)
Max. start period (Metamodel) TIME (Time)
Min. quota of presence (Metamodel) INTEGER (Integer)

Model pointer (Metamodel) STRING (Short string)
 Number DOUBLE (Floating-point number)
 Order CLOB (Character Large Object)
 Performer (Metamodel) EXPRESSION (Expression)
 Position (Metamodel) STRING (Short string)
 Priority (Metamodel) INTEGER (Integer)
 Resting time (Metamodel) TIME (Time)
 Show name ENUMERATION (Enumeration)
 Task stack (Metamodel) ENUMERATION (Enumeration)
 Transport time (Metamodel) TIME (Time)
 VisibleAttrs (Metamodel) STRING (Short string)
 Waiting time (Metamodel) TIME (Time)
 WF_Trans (Metamodel) STRING (Short string)
 AnimRep (Metamodel) STRING (Short string)
 AttrRep (Metamodel) LONGSTRING (Long string)
 Average number of participants (Metamodel) INTEGER (Integer)
 Class cardinality (Metamodel) STRING (Short string)
 ClassAbstract INTEGER (Integer)
 ClassName STRING (Short string)
 ClassVisibleINTEGER (Integer)
 Continuous execution (Metamodel) ENUMERATION (Enumeration)
 Cooperation mode (Metamodel) ENUMERATION (Enumeration)
 Cooperative (Metamodel) ENUMERATION (Enumeration)
 Done by (Metamodel) STRING (Short string)
 Execution interruptable (Metamodel) ENUMERATION (Enumeration)
 Execution time (Metamodel) TIME (Time)
 External tool coupling (Metamodel) STRING (Short string)
 GraphRep (Metamodel) LONGSTRING (Long string)
 HlpTxt (Metamodel) STRING (Short string)
 Max. resource waiting time (Metamodel) TIME (Time)
 Max. start period (Metamodel) TIME (Time)
 Min. quota of presence (Metamodel) INTEGER (Integer)
 Model pointer (Metamodel) STRING (Short string)
 Performer (Metamodel) EXPRESSION (Expression)
 Position (Metamodel) STRING (Short string)
 Priority (Metamodel) INTEGER (Integer)
 Resting time (Metamodel) TIME (Time)
 Task stack (Metamodel) ENUMERATION (Enumeration)
 Transport time (Metamodel) TIME (Time)
 VisibleAttrs (Metamodel) STRING (Short string)
 Waiting time (Metamodel) TIME (Time)
 WF_Trans (Metamodel) STRING (Short string)
 __Decision__ (Metamodel)
 Exclusive gateway
 AnimRep (Metamodel) STRING (Short string)
 AttrRep (Metamodel) LONGSTRING (Long string)
 Class cardinality (Metamodel) STRING (Short string)
 ClassAbstract INTEGER (Integer)
 ClassNameSTRING (Short string)
 ClassVisible INTEGER (Integer)
 External tool coupling (Metamodel) STRING (Short string)
 GraphRep (Metamodel) LONGSTRING (Long string)
 HlpTxt (Metamodel) STRING (Short string)
 Model pointer (Metamodel) STRING (Short string)
 Position (Metamodel) STRING (Short string)
 Variable name (Metamodel) STRING (Short string)
 Variable scope (Metamodel) ENUMERATION (Enumeration)
 Variable type (Metamodel) ENUMERATION (Enumeration)
 Variable value (Metamodel) STRING (Short string)
 VisibleAttrs (Metamodel) STRING (Short string)

WF_Trans (Metamodel) STRING (Short string)
 AnimRep (Metamodel) STRING (Short string)
 AttrRep (Metamodel) LONGSTRING (Long string)
 Class cardinality (Metamodel) STRING (Short string)
 ClassAbstract INTEGER (Integer)
 ClassName STRING (Short string)
 ClassVisibleINTEGER (Integer)
 External tool coupling (Metamodel) STRING (Short string)
 GraphRep (Metamodel) LONGSTRING (Long string)
 HlpTxt (Metamodel) STRING (Short string)
 Model pointer (Metamodel) STRING (Short string)
 Position (Metamodel) STRING (Short string)
 Variable name (Metamodel) STRING (Short string)
 Variable scope (Metamodel) ENUMERATION (Enumeration)
 Variable type (Metamodel) ENUMERATION (Enumeration)
 Variable value (Metamodel) STRING (Short string)
 VisibleAttrs (Metamodel) STRING (Short string)
 WF_Trans (Metamodel) STRING (Short string)
 __Parallelity__ (Metamodel)
 Parallel gateway
 AnimRep (Metamodel) STRING (Short string)
 AttrRep (Metamodel) LONGSTRING (Long string)
 Class cardinality (Metamodel) STRING (Short string)
 ClassAbstract INTEGER (Integer)
 ClassNameSTRING (Short string)
 ClassVisible INTEGER (Integer)
 External tool coupling (Metamodel) STRING (Short string)
 GraphRep (Metamodel) LONGSTRING (Long string)
 HlpTxt (Metamodel) STRING (Short string)
 Model pointer (Metamodel) STRING (Short string)
 Position (Metamodel) STRING (Short string)
 VisibleAttrs (Metamodel) STRING (Short string)
 WF_Trans (Metamodel) STRING (Short string)
 AnimRep (Metamodel) STRING (Short string)
 AttrRep (Metamodel) LONGSTRING (Long string)
 Class cardinality (Metamodel) STRING (Short string)
 ClassAbstract INTEGER (Integer)
 ClassName STRING (Short string)
 ClassVisibleINTEGER (Integer)
 External tool coupling (Metamodel) STRING (Short string)
 GraphRep (Metamodel) LONGSTRING (Long string)
 HlpTxt (Metamodel) STRING (Short string)
 Model pointer (Metamodel) STRING (Short string)
 Position (Metamodel) STRING (Short string)
 VisibleAttrs (Metamodel) STRING (Short string)
 WF_Trans (Metamodel) STRING (Short string)
 __Merging__ (Metamodel)
 Merging gateway
 AnimRep (Metamodel) STRING (Short string)
 AttrRep (Metamodel) LONGSTRING (Long string)
 Class cardinality (Metamodel) STRING (Short string)
 ClassAbstract INTEGER (Integer)
 ClassNameSTRING (Short string)
 ClassVisible INTEGER (Integer)
 External tool coupling (Metamodel) STRING (Short string)
 GraphRep (Metamodel) LONGSTRING (Long string)
 HlpTxt (Metamodel) STRING (Short string)
 Model pointer (Metamodel) STRING (Short string)
 Position (Metamodel) STRING (Short string)
 VisibleAttrs (Metamodel) STRING (Short string)

WF_Trans (Metamodel) STRING (Short string)
 AnimRep (Metamodel) STRING (Short string)
 AttrRep (Metamodel) LONGSTRING (Long string)
 Class cardinality (Metamodel) STRING (Short string)
 ClassAbstract INTEGER (Integer)
 ClassName STRING (Short string)
 ClassVisibleINTEGER (Integer)
 External tool coupling (Metamodel) STRING (Short string)
 GraphRep (Metamodel) LONGSTRING (Long string)
 HlpTxt (Metamodel) STRING (Short string)
 Model pointer (Metamodel) STRING (Short string)
 Position (Metamodel) STRING (Short string)
 VisibleAttrs (Metamodel) STRING (Short string)
 WF_Trans (Metamodel) STRING (Short string)
 AnimRep (Metamodel) STRING (Short string)
 AttrRep (Metamodel) LONGSTRING (Long string)
 Class cardinality (Metamodel) STRING (Short string)
 ClassAbstractINTEGER (Integer)
 ClassName STRING (Short string)
 ClassVisible INTEGER (Integer)
 External tool coupling (Metamodel) STRING (Short string)
 GraphRep (Metamodel) LONGSTRING (Long string)
 HlpTxt (Metamodel) STRING (Short string)
 Model pointer (Metamodel) STRING (Short string)
 Position (Metamodel) STRING (Short string)
 VisibleAttrs (Metamodel) STRING (Short string)
 WF_Trans (Metamodel) STRING (Short string)
 __D_end__ (Metamodel)
 End event
 AnimRep (Metamodel) STRING (Short string)
 AttrRep (Metamodel) LONGSTRING (Long string)
 Class cardinality (Metamodel) STRING (Short string)
 ClassAbstract INTEGER (Integer)
 ClassName STRING (Short string)
 ClassVisibleINTEGER (Integer)
 External tool coupling (Metamodel) STRING (Short string)
 GraphRep (Metamodel) LONGSTRING (Long string)
 HlpTxt (Metamodel) STRING (Short string)
 Model pointer (Metamodel) STRING (Short string)
 Position (Metamodel) STRING (Short string)
 Type (Metamodel) ENUMERATION (Enumeration)
 VisibleAttrs (Metamodel) STRING (Short string)
 WF_Trans (Metamodel) STRING (Short string)
 AnimRep (Metamodel) STRING (Short string)
 AttrRep (Metamodel) LONGSTRING (Long string)
 Class cardinality (Metamodel) STRING (Short string)
 ClassAbstractINTEGER (Integer)
 ClassName STRING (Short string)
 ClassVisible INTEGER (Integer)
 External tool coupling (Metamodel) STRING (Short string)
 GraphRep (Metamodel) LONGSTRING (Long string)
 HlpTxt (Metamodel) STRING (Short string)
 Model pointer (Metamodel) STRING (Short string)
 Position (Metamodel) STRING (Short string)
 Type (Metamodel) ENUMERATION (Enumeration)
 VisibleAttrs (Metamodel) STRING (Short string)
 WF_Trans (Metamodel) STRING (Short string)
 AnimRep (Metamodel) STRING (Short string)
 AttrRep (Metamodel) LONGSTRING (Long string)
 Class cardinality (Metamodel) STRING (Short string)

ClassAbstract INTEGER (Integer)
 ClassName STRING (Short string)
 ClassVisible INTEGER (Integer)
 External tool coupling (Metamodel) STRING (Short string)
 GraphRep (Metamodel) LONGSTRING (Long string)
 HlpTxt (Metamodel) STRING (Short string)
 Model pointer (Metamodel) STRING (Short string)
 Position (Metamodel) STRING (Short string)
 VisibleAttrs (Metamodel) STRING (Short string)
 WF_Trans (Metamodel) STRING (Short string)
 __D_variable__ (Metamodel)
 Variable
 AnimRep (Metamodel) STRING (Short string)
 AttrRep (Metamodel) LONGSTRING (Long string)
 Class cardinality (Metamodel) STRING (Short string)
 ClassAbstract INTEGER (Integer)
 ClassName STRING (Short string)
 ClassVisible INTEGER (Integer)
 External tool coupling (Metamodel) STRING (Short string)
 GraphRep (Metamodel) LONGSTRING (Long string)
 HlpTxt (Metamodel) STRING (Short string)
 Model pointer (Metamodel) STRING (Short string)
 Position (Metamodel) STRING (Short string)
 Variable scope (Metamodel) ENUMERATION (Enumeration)
 Variable type (Metamodel) ENUMERATION (Enumeration)
 VisibleAttrs (Metamodel) STRING (Short string)
 WF_Trans (Metamodel) STRING (Short string)
 AnimRep (Metamodel) STRING (Short string)
 AttrRep (Metamodel) LONGSTRING (Long string)
 Class cardinality (Metamodel) STRING (Short string)
 ClassAbstract INTEGER (Integer)
 ClassName STRING (Short string)
 ClassVisible INTEGER (Integer)
 External tool coupling (Metamodel) STRING (Short string)
 GraphRep (Metamodel) LONGSTRING (Long string)
 HlpTxt (Metamodel) STRING (Short string)
 Model pointer (Metamodel) STRING (Short string)
 Position (Metamodel) STRING (Short string)
 Variable scope (Metamodel) ENUMERATION (Enumeration)
 Variable type (Metamodel) ENUMERATION (Enumeration)
 VisibleAttrs (Metamodel) STRING (Short string)
 WF_Trans (Metamodel) STRING (Short string)
 __D_random_generator__ (Metamodel)
 Random generator
 AnimRep (Metamodel) STRING (Short string)
 AttrRep (Metamodel) LONGSTRING (Long string)
 Class cardinality (Metamodel) STRING (Short string)
 ClassAbstract INTEGER (Integer)
 ClassName STRING (Short string)
 ClassVisible INTEGER (Integer)
 External tool coupling (Metamodel) STRING (Short string)
 GraphRep (Metamodel) LONGSTRING (Long string)
 HlpTxt (Metamodel) STRING (Short string)
 Model pointer (Metamodel) STRING (Short string)
 Position (Metamodel) STRING (Short string)
 Value (Metamodel) STRING (Short string)
 VisibleAttrs (Metamodel) STRING (Short string)
 WF_Trans (Metamodel) STRING (Short string)
 AnimRep (Metamodel) STRING (Short string)
 AttrRep (Metamodel) LONGSTRING (Long string)

Class cardinality (Metamodel) STRING (Short string)
 ClassAbstract INTEGER (Integer)
 ClassName STRING (Short string)
 ClassVisible INTEGER (Integer)
 External tool coupling (Metamodel) STRING (Short string)
 GraphRep (Metamodel) LONGSTRING (Long string)
 HlpTxt (Metamodel) STRING (Short string)
 Model pointer (Metamodel) STRING (Short string)
 Position (Metamodel) STRING (Short string)
 Value (Metamodel) STRING (Short string)
 VisibleAttrs (Metamodel) STRING (Short string)
 WF_Trans (Metamodel) STRING (Short string)
 __D_resource__ (Metamodel)

Machine

AnimRep (Metamodel) STRING (Short string)
 AttrRep (Metamodel) LONGSTRING (Long string)
 Class cardinality (Metamodel) STRING (Short string)
 ClassAbstract INTEGER (Integer)
 ClassName STRING (Short string)
 ClassVisible INTEGER (Integer)
 External tool coupling (Metamodel) STRING (Short string)
 GraphRep (Metamodel) LONGSTRING (Long string)
 HlpTxt (Metamodel) STRING (Short string)
 Model pointer (Metamodel) STRING (Short string)
 Position (Metamodel) STRING (Short string)
 Selection (Metamodel) EXPRESSION (Expression)
 VisibleAttrs (Metamodel) STRING (Short string)
 WF_Trans (Metamodel) STRING (Short string)

Robot

AnimRep (Metamodel) STRING (Short string)
 AttrRep (Metamodel) LONGSTRING (Long string)
 Class cardinality (Metamodel) STRING (Short string)
 ClassAbstract INTEGER (Integer)
 ClassName STRING (Short string)
 ClassVisible INTEGER (Integer)
 External tool coupling (Metamodel) STRING (Short string)
 GraphRep (Metamodel) LONGSTRING (Long string)
 HlpTxt (Metamodel) STRING (Short string)
 Model pointer (Metamodel) STRING (Short string)
 Position (Metamodel) STRING (Short string)
 Selection (Metamodel) EXPRESSION (Expression)
 VisibleAttrs (Metamodel) STRING (Short string)
 WF_Trans (Metamodel) STRING (Short string)

Tool

AnimRep (Metamodel) STRING (Short string)
 AttrRep (Metamodel) LONGSTRING (Long string)
 Class cardinality (Metamodel) STRING (Short string)
 ClassAbstract INTEGER (Integer)
 ClassName STRING (Short string)
 ClassVisible INTEGER (Integer)
 External tool coupling (Metamodel) STRING (Short string)
 GraphRep (Metamodel) LONGSTRING (Long string)
 HlpTxt (Metamodel) STRING (Short string)
 Model pointer (Metamodel) STRING (Short string)
 Position (Metamodel) STRING (Short string)
 Selection (Metamodel) EXPRESSION (Expression)
 VisibleAttrs (Metamodel) STRING (Short string)
 WF_Trans (Metamodel) STRING (Short string)
 AnimRep (Metamodel) STRING (Short string)
 AttrRep (Metamodel) LONGSTRING (Long string)

Class cardinality (Metamodel) STRING (Short string)
 ClassAbstract INTEGER (Integer)
 ClassName STRING (Short string)
 ClassVisible INTEGER (Integer)
 External tool coupling (Metamodel) STRING (Short string)
 GraphRep (Metamodel) LONGSTRING (Long string)
 HlpTxt (Metamodel) STRING (Short string)
 Model pointer (Metamodel) STRING (Short string)
 Position (Metamodel) STRING (Short string)
 Selection (Metamodel) EXPRESSION (Expression)
 VisibleAttrs (Metamodel) STRING (Short string)
 WF_Trans (Metamodel) STRING (Short string)
 __D_container__ (Metamodel)
 __D_swimlane__ (Metamodel)
 Swimlane
 Alignment ENUMERATION (Enumeration)
 Allowed objects (Metamodel) STRING (Short string)
 AnimRep (Metamodel) STRING (Short string)
 AttrRep (Metamodel) LONGSTRING (Long string)
 Class cardinality (Metamodel) STRING (Short string)
 ClassAbstract INTEGER (Integer)
 ClassName STRING (Short string)
 ClassVisible INTEGER (Integer)
 Color STRING (Short string)
 Display water marks ENUMERATION (Enumeration)
 External tool coupling (Metamodel) STRING (Short string)
 Fontcolor EXPRESSION (Expression)
 GraphRep (Metamodel) LONGSTRING (Long string)
 HlpTxt (Metamodel) STRING (Short string)
 Model pointer (Metamodel) STRING (Short string)
 Position (Metamodel) STRING (Short string)
 VisibleAttrs (Metamodel) STRING (Short string)
 WF_Trans (Metamodel) STRING (Short string)
 Allowed objects (Metamodel) STRING (Short string)
 AnimRep (Metamodel) STRING (Short string)
 AttrRep (Metamodel) LONGSTRING (Long string)
 Class cardinality (Metamodel) STRING (Short string)
 ClassAbstract INTEGER (Integer)
 ClassName STRING (Short string)
 ClassVisible INTEGER (Integer)
 External tool coupling (Metamodel) STRING (Short string)
 GraphRep (Metamodel) LONGSTRING (Long string)
 HlpTxt (Metamodel) STRING (Short string)
 Model pointer (Metamodel) STRING (Short string)
 Position (Metamodel) STRING (Short string)
 VisibleAttrs (Metamodel) STRING (Short string)
 WF_Trans (Metamodel) STRING (Short string)
 __D_aggregation__ (Metamodel)
 AnimRep (Metamodel) STRING (Short string)
 AttrRep (Metamodel) LONGSTRING (Long string)
 Class cardinality (Metamodel) STRING (Short string)
 ClassAbstract INTEGER (Integer)
 ClassName STRING (Short string)
 ClassVisible INTEGER (Integer)
 External tool coupling (Metamodel) STRING (Short string)
 GraphRep (Metamodel) LONGSTRING (Long string)
 HlpTxt (Metamodel) STRING (Short string)
 Model pointer (Metamodel) STRING (Short string)
 Position (Metamodel) STRING (Short string)
 VisibleAttrs (Metamodel) STRING (Short string)

WF_Trans (Metamodel) STRING (Short string)
 Storage
 AnimRep (Metamodel) STRING (Short string)
 AttrRep (Metamodel) LONGSTRING (Long string)
 Class cardinality (Metamodel) STRING (Short string)
 ClassAbstract INTEGER (Integer)
 ClassName STRING (Short string)
 ClassVisible INTEGER (Integer)
 External tool coupling (Metamodel) STRING (Short string)
 GraphRep (Metamodel) LONGSTRING (Long string)
 HlpTxt (Metamodel) STRING (Short string)
 Model pointer (Metamodel) STRING (Short string)
 Position (Metamodel) STRING (Short string)
 VisibleAttrs (Metamodel) STRING (Short string)
 WF_Trans (Metamodel) STRING (Short string)
 AnimRep (Metamodel) STRING (Short string)
 AttrRep (Metamodel) LONGSTRING (Long string)
 Class cardinality (Metamodel) STRING (Short string)
 ClassAbstract INTEGER (Integer)
 ClassName STRING (Short string)
 ClassVisible INTEGER (Integer)
 External tool coupling (Metamodel) STRING (Short string)
 GraphRep (Metamodel) LONGSTRING (Long string)
 HlpTxt (Metamodel) STRING (Short string)
 Model pointer (Metamodel) STRING (Short string)
 Position (Metamodel) STRING (Short string)
 VisibleAttrs (Metamodel) STRING (Short string)
 WF_Trans (Metamodel) STRING (Short string)
 __D_agent__ (Metamodel)
 AnimRep (Metamodel) STRING (Short string)
 AttrRep (Metamodel) LONGSTRING (Long string)
 Calendar (Metamodel) LONGSTRING (Long string)
 Class cardinality (Metamodel) STRING (Short string)
 ClassAbstract INTEGER (Integer)
 ClassName STRING (Short string)
 ClassVisible INTEGER (Integer)
 External tool coupling (Metamodel) STRING (Short string)
 Format (Metamodel) STRING (Short string)
 GraphRep (Metamodel) LONGSTRING (Long string)
 HlpTxt (Metamodel) STRING (Short string)
 Information text (Metamodel) STRING (Short string)
 Model pointer (Metamodel) STRING (Short string)
 Position (Metamodel) STRING (Short string)
 Scope (Metamodel) STRING (Short string)
 Visible (Metamodel) ENUMERATION (Enumeration)
 VisibleAttrs (Metamodel) STRING (Short string)
 WF_Trans (Metamodel) STRING (Short string)
 __LibraryMetaData__
 __APListChangeCounter__ INTEGER (Integer)
 __ModelListChangeCounter__ INTEGER (Integer)
 __UserListChangeCounter__ INTEGER (Integer)
 AnimRep (Metamodel) STRING (Short string)
 AttrRep (Metamodel) LONGSTRING (Long string)
 Class cardinality (Metamodel) STRING (Short string)
 ClassAbstract INTEGER (Integer)
 ClassName STRING (Short string)
 ClassVisible INTEGER (Integer)
 External tool coupling (Metamodel) STRING (Short string)
 GraphRep (Metamodel) LONGSTRING (Long string)
 HlpTxt (Metamodel) STRING (Short string)

homedir STRING (Short string)
 Model pointer (Metamodel) STRING (Short string)
 Position (Metamodel) STRING (Short string)
 VisibleAttrs (Metamodel) STRING (Short string)
 WF_Trans (Metamodel) STRING (Short string)
 __ModelTypeMetaData__
 __GfxThumb__ LONGSTRING (Long string)
 AnimRep (Metamodel) STRING (Short string)
 AttrRep (Metamodel) LONGSTRING (Long string)
 Class cardinality (Metamodel) STRING (Short string)
 ClassAbstract INTEGER (Integer)
 ClassName STRING (Short string)
 ClassVisible INTEGER (Integer)
 External tool coupling (Metamodel) STRING (Short string)
 GraphRep (Metamodel) LONGSTRING (Long string)
 HlpTxt (Metamodel) STRING (Short string)
 Model pointer (Metamodel) STRING (Short string)
 Position (Metamodel) STRING (Short string)
 VisibleAttrs (Metamodel) STRING (Short string)
 WF_Trans (Metamodel) STRING (Short string)
 process-landscape-construct
 external-partner
 Supplier
 AnimRep (Metamodel) STRING (Short string)
 AttrRep (Metamodel) LONGSTRING (Long string)
 Class cardinality (Metamodel) STRING (Short string)
 ClassAbstract INTEGER (Integer)
 ClassName STRING (Short string)
 ClassVisible INTEGER (Integer)
 Description STRING (Short string)
 External tool coupling (Metamodel) STRING (Short string)
 GraphRep (Metamodel) LONGSTRING (Long string)
 HlpTxt (Metamodel) STRING (Short string)
 Model pointer (Metamodel) STRING (Short string)
 Position (Metamodel) STRING (Short string)
 VisibleAttrs (Metamodel) STRING (Short string)
 WF_Trans (Metamodel) STRING (Short string)
 Client
 AnimRep (Metamodel) STRING (Short string)
 AttrRep (Metamodel) LONGSTRING (Long string)
 Class cardinality (Metamodel) STRING (Short string)
 ClassAbstract INTEGER (Integer)
 ClassName STRING (Short string)
 ClassVisible INTEGER (Integer)
 Description STRING (Short string)
 External tool coupling (Metamodel) STRING (Short string)
 GraphRep (Metamodel) LONGSTRING (Long string)
 HlpTxt (Metamodel) STRING (Short string)
 Model pointer (Metamodel) STRING (Short string)
 Position (Metamodel) STRING (Short string)
 VisibleAttrs (Metamodel) STRING (Short string)
 WF_Trans (Metamodel) STRING (Short string)
 AnimRep (Metamodel) STRING (Short string)
 AttrRep (Metamodel) LONGSTRING (Long string)
 Class cardinality (Metamodel) STRING (Short string)
 ClassAbstract INTEGER (Integer)
 ClassName STRING (Short string)
 ClassVisible INTEGER (Integer)
 External tool coupling (Metamodel) STRING (Short string)
 GraphRep (Metamodel) LONGSTRING (Long string)

HlpTxt (Metamodel) STRING (Short string)
 Model pointer (Metamodel) STRING (Short string)
 Position (Metamodel) STRING (Short string)
 VisibleAttrs (Metamodel) STRING (Short string)
 WF_Trans (Metamodel) STRING (Short string)

Value block

AnimRep (Metamodel) STRING (Short string)
 AttrRep (Metamodel) LONGSTRING (Long string)
 Class cardinality (Metamodel) STRING (Short string)
 ClassAbstract INTEGER (Integer)
 ClassName STRING (Short string)
 ClassVisible INTEGER (Integer)
 Display name and reference ENUMERATION (Enumeration)
 External tool coupling (Metamodel) STRING (Short string)
 GraphRep (Metamodel) LONGSTRING (Long string)
 HlpTxt (Metamodel) STRING (Short string)
 Model pointer (Metamodel) STRING (Short string)
 Order INTEGER (Integer)
 Position (Metamodel) STRING (Short string)
 Referenced process INTERREF (Inter-model reference)
 Subprocessname EXPRESSION (Expression)
 VisibleAttrs (Metamodel) STRING (Short string)
 WF_Trans (Metamodel) STRING (Short string)

AnimRep (Metamodel) STRING (Short string)
 AttrRep (Metamodel) LONGSTRING (Long string)
 Class cardinality (Metamodel) STRING (Short string)
 ClassAbstract INTEGER (Integer)
 ClassName STRING (Short string)
 ClassVisible INTEGER (Integer)
 External tool coupling (Metamodel) STRING (Short string)
 GraphRep (Metamodel) LONGSTRING (Long string)
 HlpTxt (Metamodel) STRING (Short string)
 Model pointer (Metamodel) STRING (Short string)
 Position (Metamodel) STRING (Short string)
 VisibleAttrs (Metamodel) STRING (Short string)
 WF_Trans (Metamodel) STRING (Short string)

product-structure-construct

Product

AnimRep (Metamodel) STRING (Short string)
 AttrRep (Metamodel) LONGSTRING (Long string)
 Class cardinality (Metamodel) STRING (Short string)
 ClassAbstract INTEGER (Integer)
 ClassName STRING (Short string)
 ClassVisible INTEGER (Integer)
 External tool coupling (Metamodel) STRING (Short string)
 GraphRep (Metamodel) LONGSTRING (Long string)
 HlpTxt (Metamodel) STRING (Short string)
 Model pointer (Metamodel) STRING (Short string)
 Position (Metamodel) STRING (Short string)
 VisibleAttrs (Metamodel) STRING (Short string)
 WF_Trans (Metamodel) STRING (Short string)

Component

AnimRep (Metamodel) STRING (Short string)
 AttrRep (Metamodel) LONGSTRING (Long string)
 Class cardinality (Metamodel) STRING (Short string)
 ClassAbstract INTEGER (Integer)
 ClassName STRING (Short string)
 ClassVisible INTEGER (Integer)
 External tool coupling (Metamodel) STRING (Short string)
 GraphRep (Metamodel) LONGSTRING (Long string)

HlpTxt (Metamodel) STRING (Short string)
 Model pointer (Metamodel) STRING (Short string)
 Position (Metamodel) STRING (Short string)
 VisibleAttrs (Metamodel) STRING (Short string)
 WF_Trans (Metamodel) STRING (Short string)
 Raw material
 AnimRep (Metamodel) STRING (Short string)
 AttrRep (Metamodel) LONGSTRING (Long string)
 Class cardinality (Metamodel) STRING (Short string)
 ClassAbstract INTEGER (Integer)
 ClassName STRING (Short string)
 ClassVisible INTEGER (Integer)
 External tool coupling (Metamodel) STRING (Short string)
 GraphRep (Metamodel) LONGSTRING (Long string)
 HlpTxt (Metamodel) STRING (Short string)
 Model pointer (Metamodel) STRING (Short string)
 Position (Metamodel) STRING (Short string)
 VisibleAttrs (Metamodel) STRING (Short string)
 WF_Trans (Metamodel) STRING (Short string)
 AnimRep (Metamodel) STRING (Short string)
 AttrRep (Metamodel) LONGSTRING (Long string)
 Class cardinality (Metamodel) STRING (Short string)
 ClassAbstract INTEGER (Integer)
 ClassName STRING (Short string)
 ClassVisible INTEGER (Integer)
 External tool coupling (Metamodel) STRING (Short string)
 GraphRep (Metamodel) LONGSTRING (Long string)
 HlpTxt (Metamodel) STRING (Short string)
 Model pointer (Metamodel) STRING (Short string)
 Position (Metamodel) STRING (Short string)
 VisibleAttrs (Metamodel) STRING (Short string)
 WF_Trans (Metamodel) STRING (Short string)
 AnimRep (Metamodel) STRING (Short string)
 AttrRep (Metamodel) LONGSTRING (Long string)
 Class cardinality (Metamodel) STRING (Short string)
 ClassAbstract INTEGER (Integer)
 ClassName STRING (Short string)
 ClassVisible INTEGER (Integer)
 External tool coupling (Metamodel) STRING (Short string)
 GraphRep (Metamodel) LONGSTRING (Long string)
 HlpTxt (Metamodel) STRING (Short string)
 Model pointer (Metamodel) STRING (Short string)
 Position (Metamodel) STRING (Short string)
 VisibleAttrs (Metamodel) STRING (Short string)
 WF_Trans (Metamodel) STRING (Short string)
 Relation classes
 Is inside (Metamodel) __D_construct__ --> __D_container__
 AutoConnect (Metamodel) STRING (Short string)
 Subsequent (Metamodel) __D_event__ --> __D_event__
 AnimRep (Metamodel) STRING (Short string)
 AttrRep (Metamodel) STRING (Short string)
 Comment (Metamodel) STRING (Short string)
 Connector number (Metamodel) INTEGER (Integer)
 GraphRep (Metamodel) STRING (Short string)
 HlpTxt (Metamodel) STRING (Short string)
 Positions (Metamodel) STRING (Short string)
 Transition condition (Metamodel) STRING (Short string)
 Transition probability (Metamodel) STRING (Short string)
 VisibleAttrs (Metamodel) STRING (Short string)
 Visualization (Metamodel) ENUMERATION (Enumeration)

Visualized values (Metamodel) ENUMERATION (Enumeration)
 Sets variable (Metamodel) __D_random_generator__ --> __D_variable__
 AnimRep (Metamodel) STRING (Short string)
 AttrRep (Metamodel) STRING (Short string)
 Connector number (Metamodel) INTEGER (Integer)
 GraphRep (Metamodel) STRING (Short string)
 HlpTxt (Metamodel) STRING (Short string)
 Positions (Metamodel) STRING (Short string)
 VisibleAttrs (Metamodel) STRING (Short string)
 Visualization (Metamodel) ENUMERATION (Enumeration)
 Sets (Metamodel) __D_random_generator__ --> __D_variable_assignment_object__
 AnimRep (Metamodel) STRING (Short string)
 AttrRep (Metamodel) STRING (Short string)
 Connector number (Metamodel) INTEGER (Integer)
 GraphRep (Metamodel) STRING (Short string)
 HlpTxt (Metamodel) STRING (Short string)
 Positions (Metamodel) STRING (Short string)
 VisibleAttrs (Metamodel) STRING (Short string)
 Visualization (Metamodel) ENUMERATION (Enumeration)
 Parameter (Metamodel) __D_variable__ --> __Start__
 AnimRep (Metamodel) STRING (Short string)
 AttrRep (Metamodel) STRING (Short string)
 Connector number (Metamodel) INTEGER (Integer)
 GraphRep (Metamodel) STRING (Short string)
 HlpTxt (Metamodel) STRING (Short string)
 Index (Metamodel) INTEGER (Integer)
 Positions (Metamodel) STRING (Short string)
 VisibleAttrs (Metamodel) STRING (Short string)
 Visualization (Metamodel) ENUMERATION (Enumeration)
 Call parameter (Metamodel) __Subgraph__ --> __D_variable__
 AnimRep (Metamodel) STRING (Short string)
 AttrRep (Metamodel) STRING (Short string)
 Connector number (Metamodel) INTEGER (Integer)
 GraphRep (Metamodel) STRING (Short string)
 HlpTxt (Metamodel) STRING (Short string)
 Index (Metamodel) INTEGER (Integer)
 Positions (Metamodel) STRING (Short string)
 VisibleAttrs (Metamodel) STRING (Short string)
 Visualization (Metamodel) ENUMERATION (Enumeration)
 Uses (Metamodel) __Activity__ --> __D_resource__
 AnimRep (Metamodel) STRING (Short string)
 AttrRep (Metamodel) STRING (Short string)
 Connector number (Metamodel) INTEGER (Integer)
 GraphRep (Metamodel) STRING (Short string)
 HlpTxt (Metamodel) STRING (Short string)
 Positions (Metamodel) STRING (Short string)
 VisibleAttrs (Metamodel) STRING (Short string)
 Visualization (Metamodel) ENUMERATION (Enumeration)
 Is component of _product-structure-construct_ --> _product-structure-construct_
 AttrRep STRING (Short string)
 GraphRep STRING (Short string)
 Name STRING (Short string)
 Positions STRING (Short string)
 association __D-construct__ --> __D-construct__
 AttrRep STRING (Short string)
 GraphRep STRING (Short string)
 Name STRING (Short string)
 Positions STRING (Short string)
 material flow __D-construct__ --> __D-construct__
 AttrRep STRING (Short string)

GraphRep STRING (Short string)
 Name STRING (Short string)
 Positions STRING (Short string)
 tool flow __D_resource__ --> __D_event__
 AttrRep STRING (Short string)
 GraphRep STRING (Short string)
 Name STRING (Short string)
 Positions STRING (Short string)
 information flow Task --> Storage
 AttrRep STRING (Short string)
 GraphRep STRING (Short string)
 Name STRING (Short string)
 Positions STRING (Short string)

Appendix C

__S_construct__ (Metamodel)
__S_group__ (Metamodel)
__S_container__ (Metamodel)
__S_swimlane__ (Metamodel)
Allowed objects (Metamodel) STRING (Short string)
AnimRep (Metamodel) STRING (Short string)
AttrRep (Metamodel) LONGSTRING (Long string)
Class cardinality (Metamodel) STRING (Short string)
ClassAbstract INTEGER (Integer)
ClassName STRING (Short string)
ClassVisibleINTEGER (Integer)
External tool coupling (Metamodel) STRING (Short string)
GraphRep (Metamodel) LONGSTRING (Long string)
HlpTxt (Metamodel) STRING (Short string)
Model pointer (Metamodel) STRING (Short string)
Position (Metamodel) STRING (Short string)
VisibleAttrs (Metamodel) STRING (Short string)
WF_Trans (Metamodel) STRING (Short string)
__S_aggregation__ (Metamodel)
AnimRep (Metamodel) STRING (Short string)
AttrRep (Metamodel) LONGSTRING (Long string)
Class cardinality (Metamodel) STRING (Short string)
ClassAbstract INTEGER (Integer)
ClassName STRING (Short string)
ClassVisibleINTEGER (Integer)
External tool coupling (Metamodel) STRING (Short string)
GraphRep (Metamodel) LONGSTRING (Long string)
HlpTxt (Metamodel) STRING (Short string)
Model pointer (Metamodel) STRING (Short string)
Position (Metamodel) STRING (Short string)
VisibleAttrs (Metamodel) STRING (Short string)
WF_Trans (Metamodel) STRING (Short string)
AnimRep (Metamodel) STRING (Short string)
AttrRep (Metamodel) LONGSTRING (Long string)
Class cardinality (Metamodel) STRING (Short string)
ClassAbstractINTEGER (Integer)
ClassName STRING (Short string)
ClassVisible INTEGER (Integer)
External tool coupling (Metamodel) STRING (Short string)
GraphRep (Metamodel) LONGSTRING (Long string)
HlpTxt (Metamodel) STRING (Short string)
Model pointer (Metamodel) STRING (Short string)
Position (Metamodel) STRING (Short string)
VisibleAttrs (Metamodel) STRING (Short string)
WF_Trans (Metamodel) STRING (Short string)
Department
AnimRep (Metamodel) STRING (Short string)
AttrRep (Metamodel) LONGSTRING (Long string)
Class cardinality (Metamodel) STRING (Short string)
ClassAbstractINTEGER (Integer)
ClassName STRING (Short string)
ClassVisible INTEGER (Integer)
External tool coupling (Metamodel) STRING (Short string)
GraphRep (Metamodel) LONGSTRING (Long string)
HlpTxt (Metamodel) STRING (Short string)
Model pointer (Metamodel) STRING (Short string)
Position (Metamodel) STRING (Short string)

VisibleAttrs (Metamodel) STRING (Short string)
 WF_Trans (Metamodel) STRING (Short string)
 Role
 AnimRep (Metamodel) STRING (Short string)
 AttrRep (Metamodel) LONGSTRING (Long string)
 Class cardinality (Metamodel) STRING (Short string)
 ClassAbstract INTEGER (Integer)
 ClassName STRING (Short string)
 ClassVisible INTEGER (Integer)
 External tool coupling (Metamodel) STRING (Short string)
 GraphRep (Metamodel) LONGSTRING (Long string)
 HlpTxt (Metamodel) STRING (Short string)
 Model pointer (Metamodel) STRING (Short string)
 Position (Metamodel) STRING (Short string)
 VisibleAttrs (Metamodel) STRING (Short string)
 WF_Trans (Metamodel) STRING (Short string)
 AnimRep (Metamodel) STRING (Short string)
 AttrRep (Metamodel) LONGSTRING (Long string)
 Class cardinality (Metamodel) STRING (Short string)
 ClassAbstract INTEGER (Integer)
 ClassName STRING (Short string)
 ClassVisible INTEGER (Integer)
 External tool coupling (Metamodel) STRING (Short string)
 GraphRep (Metamodel) LONGSTRING (Long string)
 HlpTxt (Metamodel) STRING (Short string)
 Model pointer (Metamodel) STRING (Short string)
 Position (Metamodel) STRING (Short string)
 VisibleAttrs (Metamodel) STRING (Short string)
 WF_Trans (Metamodel) STRING (Short string)
 _S_person_ (Metamodel)
 Performer
 AnimRep (Metamodel) STRING (Short string)
 AttrRep (Metamodel) LONGSTRING (Long string)
 Availability (Metamodel) EXPRESSION (Expression)
 Calendar (Metamodel) LONGSTRING (Long string)
 Capacity DOUBLE (Floating-point number)
 Class cardinality (Metamodel) STRING (Short string)
 ClassAbstract INTEGER (Integer)
 ClassName STRING (Short string)
 ClassVisible INTEGER (Integer)
 External tool coupling (Metamodel) STRING (Short string)
 GraphRep (Metamodel) LONGSTRING (Long string)
 HlpTxt (Metamodel) STRING (Short string)
 Hourly wages (Metamodel) DOUBLE (Floating-point number)
 Info on results DOUBLE (Floating-point number)
 Model pointer (Metamodel) STRING (Short string)
 Personnel costs DOUBLE (Floating-point number)
 Position (Metamodel) STRING (Short string)
 VisibleAttrs (Metamodel) STRING (Short string)
 WF_Trans (Metamodel) STRING (Short string)
 Workload DOUBLE (Floating-point number)
 AnimRep (Metamodel) STRING (Short string)
 AttrRep (Metamodel) LONGSTRING (Long string)
 Availability (Metamodel) EXPRESSION (Expression)
 Calendar (Metamodel) LONGSTRING (Long string)
 Class cardinality (Metamodel) STRING (Short string)
 ClassAbstract INTEGER (Integer)
 ClassName STRING (Short string)
 ClassVisible INTEGER (Integer)
 External tool coupling (Metamodel) STRING (Short string)

GraphRep (Metamodel) LONGSTRING (Long string)
 HlpTxt (Metamodel) STRING (Short string)
 Hourly wages (Metamodel) DOUBLE (Floating-point number)
 Model pointer (Metamodel) STRING (Short string)
 Position (Metamodel) STRING (Short string)
 VisibleAttrs (Metamodel) STRING (Short string)
 WF_Trans (Metamodel) STRING (Short string)
 __S_resource__ (Metamodel)
 AnimRep (Metamodel) STRING (Short string)
 AttrRep (Metamodel) LONGSTRING (Long string)
 Class cardinality (Metamodel) STRING (Short string)
 ClassAbstract INTEGER (Integer)
 ClassName STRING (Short string)
 ClassVisible INTEGER (Integer)
 External tool coupling (Metamodel) STRING (Short string)
 GraphRep (Metamodel) LONGSTRING (Long string)
 HlpTxt (Metamodel) STRING (Short string)
 Hourly wages (Metamodel) DOUBLE (Floating-point number)
 Model pointer (Metamodel) STRING (Short string)
 Position (Metamodel) STRING (Short string)
 VisibleAttrs (Metamodel) STRING (Short string)
 WF_Trans (Metamodel) STRING (Short string)
 __S_agent__ (Metamodel)
 AnimRep (Metamodel) STRING (Short string)
 AttrRep (Metamodel) LONGSTRING (Long string)
 Calendar (Metamodel) LONGSTRING (Long string)
 Class cardinality (Metamodel) STRING (Short string)
 ClassAbstract INTEGER (Integer)
 ClassName STRING (Short string)
 ClassVisible INTEGER (Integer)
 External tool coupling (Metamodel) STRING (Short string)
 Format (Metamodel) STRING (Short string)
 GraphRep (Metamodel) LONGSTRING (Long string)
 HlpTxt (Metamodel) STRING (Short string)
 Information text (Metamodel) STRING (Short string)
 Model pointer (Metamodel) STRING (Short string)
 Position (Metamodel) STRING (Short string)
 Scope (Metamodel) STRING (Short string)
 Visible (Metamodel) ENUMERATION (Enumeration)
 VisibleAttrs (Metamodel) STRING (Short string)
 WF_Trans (Metamodel) STRING (Short string)
 AnimRep (Metamodel) STRING (Short string)
 AttrRep (Metamodel) LONGSTRING (Long string)
 Class cardinality (Metamodel) STRING (Short string)
 ClassAbstract INTEGER (Integer)
 ClassName STRING (Short string)
 ClassVisible INTEGER (Integer)
 External tool coupling (Metamodel) STRING (Short string)
 GraphRep (Metamodel) LONGSTRING (Long string)
 HlpTxt (Metamodel) STRING (Short string)
 Model pointer (Metamodel) STRING (Short string)
 Position (Metamodel) STRING (Short string)
 VisibleAttrs (Metamodel) STRING (Short string)
 WF_Trans (Metamodel) STRING (Short string)
 Relation classes
 Is inside (Metamodel) __S-construct__ --> __S_container__
 AutoConnect (Metamodel) STRING (Short string)
 Is manager Performer --> Department
 AttrRep STRING (Short string)
 GraphRep STRING (Short string)

| | |
|---------------------|-------------------------------------|
| Positions | STRING (Short string) |
| Has role | Performer --> Role |
| AttrRep | STRING (Short string) |
| GraphRep | STRING (Short string) |
| Positions | STRING (Short string) |
| Has cross-reference | __S-construct__ --> __S-construct__ |
| AttrRep | STRING (Short string) |
| GraphRep | STRING (Short string) |
| Positions | STRING (Short string) |
| Belongs to | Performer --> Department |
| AttrRep | STRING (Short string) |
| GraphRep | STRING (Short string) |
| Positions | STRING (Short string) |

Abstract

Business process modeling (BPM) is a core discipline in today's business process management activities. It aims at graphically describing the ever-more complex and interdependent processes within a company. Many general-purpose BPM languages often fail to address the specific needs of the respective application domain. Here, the concept of domain-specific modeling helps to bridge this gap by increasing the level of abstraction of BPM and enabling the inclusion of domain-specific concepts, rules, and functionality.

When it comes to domain-specific BPM method design, there is no approach so far, which considers the design, implementation, and modeling phase specifically in the light of the domain. This thesis presents a life-cycle framework for domain-specific BPM, which focuses on the systematic evaluation of the domain in order to extract domain-specific concepts, rules, and functionality. The framework builds on the well-established modeling method engineering approach, which uses the concept of meta-modeling. It is implemented by using the technology of meta-modeling platforms.

The hypothesis, that domain-specificity is located on the meta-level of the language, is proven within this thesis by a literature analysis of existing domain-specific modeling languages. As a methodology, the Design Science Research approach is used for the development of the Domain Integration Framework (DIF) as the artifact. The DIF is applied and evaluated in a case study on the development of a domain-specific BPM tool for modeling an automotive assembly line process.

Keywords: Business Process Modeling, Domain-specific, Modeling Method Engineering, Modeling Tool

Zusammenfassung

Geschäftsprozessmodellierung (GPM) ist eine Hauptaktivität im heutigen Umfeld des Geschäftsprozessmanagements. Ihr Ziel ist es, die zunehmend komplexen und voneinander abhängigen Prozesse eines Unternehmens grafisch darzustellen. Viele allgemeingültige GPM Sprachen sind oftmals nicht in der Lage, die spezifischen Bedürfnisse der jeweiligen Anwendungsdomäne darzustellen. Hier hilft das Konzept der domänenspezifischen Modellierung dabei, diese Lücke zu schließen, indem das Abstraktionsniveau der GPM auf ein höheres Level gehoben wird, um domänenspezifische Konzepte, Regeln und Funktionalitäten einzubeziehen.

Wenn es darum geht, domänenspezifische GPM Methoden zu entwickeln, gibt es bis dato keinen Ansatz, der die Entwicklungs-, Implementierungs- und Modellierungsphase speziell mit Fokus auf die Domäne betrachtet. Diese Arbeit stellt ein Lebenszyklus-Framework für domänenspezifische GPM vor, welche sich auf die systematische Evaluation der Domäne fokussiert, um domänenspezifische Konzepte, Regeln und Funktionalitäten zu extrahieren. Das Framework baut auf dem bewährten Ansatz des Modellierungsmethoden Engineerings auf, welcher das Konzept der Metamodellierung verwendet. Die Technologie für die Implementierung bilden Metamodellierungs-Plattformen.

Die Hypothese, dass Domänenspezifität auf der Meta-Ebene der Sprache angesiedelt ist, wird innerhalb dieser Arbeit durch eine Literaturanalyse bereits existierender domänenspezifischer Modellierungssprachen bewiesen. Als Methodik wird der Design Science Research Ansatz verwendet, um das Domain Integration Framework (DIF) als das Artefakt zu entwickeln. Das DIF wird im Zuge dieser Arbeit am Beispiel der Entwicklung eines domänenspezifischen GPM-Tools für die Modellierung eines Automobil-Montageprozesses angewendet und evaluiert.

Stichworte: Geschäftsprozessmodellierung, Domänenspezifisch, Modellierungsmethoden Engineering, Modellierungs-Tool