Hendrik Schön

Role-based Adaptation of Business Reference Models to Application Models

An Enterprise Modeling Methodology for Software Construction

Dissertation

A dissertation submitted by Hendrik Schön, born June 19, 1990 in Dresden,
to the Technische Universität Dresden in accordance with the requirements
of the degree of Doktoringenieur (Dr.-Ing.) in the Faculty of Computer Science.

First Reviewer and Supervisor:	Prof. Dr. Susanne Strahringer
Second Reviewer:	Prof. Jelena Zdravkovic, Ph.D.
Second Supervisor:	Prof. Dr. Frank J. Furrer
Third Supervisor:	Prof. Dr. Uwe Aßmann

Submission Date: April 27, 2020
Defense Date: June 16, 2020

This Ph.D. is funded by the German Research Foundation (DFG) within the Research Training Group "Role-based Software Infrastructures for continuous-context-sensitive Systems" (GRK 1907).

For my family

Abstract

Large software systems are in need of a construction plan to determine and define every concept and element used in order to not end up in complex, unusable, and cost-intensive systems. Different modeling languages, like UML, support the development of these construction plans and visualize them for the system's stakeholders. Reference models are a specific kind of construction plan, used as templates for information systems and already capture business domain knowledge for reuse and tailoring. By adaptation, reference models are tailored to enterprise-specific application models, which can be used for software construction and maintenance. However, current adaptation methods suffer from the limitations of pure object-oriented development (e.g., identity issues, large inheritance trees, and inflexibility). *In this thesis, the usage of roles as the sole adaptation mechanism is proposed to solve these challenges.* With the help of conceptual roles, it is possible to create rich model variations and adaptations from existing (industry standard) reference models, and it is simpler to react to model evolution and changing business logic. Adaptations can be specified with more precision by maintaining or even increasing the model's expressiveness. As a consequence, the role-enriched final application model can be used to describe software systems in more detail, with different perspectives, and, if available, can be implemented with a role supporting programming language. However, even without this step, the application model itself will provide valuable insights into the overall construction plan of a software system by the combination of structure and behavior and a clear separation of relatively stable domain knowledge from its use case specific adaptation.

Preface

A few years ago, I probably never imagined that I would be able to write the preface of my own dissertation thesis. And yet, so it is done right now, as the last part of the thesis. Right after the completion of my diploma thesis, glad to finish it with a satisfactory outcome, my supervisor recommended the Ph.D. position in the RoSI program: a research project dedicated to supporting alumni to work on their Ph.D. degrees. Now, three years later, I am glad to have decided this way and accepted the offer that followed my application. Since then, as a RoSI Ph.D., I encountered all the ups and downs of being a scientist and working on the thesis, including the failures and misconceptions that are often an intrinsic part of the work. Nevertheless, sooner or later, all of this will end hopefully in a good way. To write a dissertation thesis, together with all preliminary research, was definitely a personality-shaping process. I am happy for every experience that I took with me during this time. Right before accepting the Ph.D. position, I asked a friend for his' opinion on whether I should accept the offer. He simply answered me: "Hendrik, this could be the last chance in your life where, free from all obligations, you can do research that matters to yourself." In retrospect, this was entirely true. A Ph.D. project is indeed not easy, but still, a once-in-a-lifetime experience. As a person who really likes creativity and innovation in science, I am grateful that I had the chance to do so. And with that, I conclude my Ph.D. period with all the good memories.

Hendrik Schön

Dresden, April 2020 *Hendrik Schön*

Acknowledgments

A Ph.D. thesis is indeed written by an individual, nevertheless, always supported by many others in the background, surely “on the shoulders of giants”. The thesis at hand is no exception, and I am happy and lucky for the support from several parties during the Ph.D. process.

I want to thank all my supervisors, who enabled the Ph.D. in all its facets. First, I want to give my appreciation to *Susanne Strahringer* for being my primary supervisor. During the years, she provided me with valuable support, feedback, and ideas for all parts of my research so that I can achieve my goals and expectations. I really enjoyed being part of her chair, many thanks for a great time. Further, I want to thank *Frank J. Furrer* for being my second supervisor. After the successful supervision of my diploma thesis, I am happy that he decided to accompany my Ph.D. project as well. Due to his’ experience in the practical field, my research got an impressive impact and connection to solve practical problems. On the same level, I want to thank *Uwe Aßmann* for his inspiring and visionary guidance of my Ph.D. as a supervisor as well. With that, the thesis got much more value considering software engineering. Similarly, many thanks to *Jelena Zdravkovic* for being my second reviewer and supervisor during my stay abroad in Sweden. I am grateful that I had the opportunity to expand my research experience in Sweden in such a great way. I would definitely do it again in the same way at any time.

Further, I want to thank all my co-authors and research partners that supported my research and publications with their direct involvement. Among others, I want to thank *Katja Bley*, *Raoul Hentschel*, *Thomas Kühn*, and *Christopher Werner* explicitly for joining me on my research path; I will miss our marvelous collaborations. Moreover, I want to thank my amazing colleagues that I met during my Ph.D. period. It was a great time working together. Thanks to the colleagues from the RoSI program, it was like a big research family. I really enjoyed our conversations and activities during the plenty of workshops, meetings, and get-togethers. Special thanks to *Dirk Habich*, who convinced me to apply for the RoSI program and who supported me during this time. Also, thanks to my colleagues from the chair of information systems (ISIH), the software technology chair (ST), and the computer science department from Stockholm University (DSV). I found many new and great friends and I hope very much that it remains so for a long time. I enjoyed and will never forget all the working and free time we spend. Without them, my Ph.D. period would not be the same. Further, I want to thank the students that I have mentored, *Sebastian Leichsenring* and *Lars Westerman*, for their contribution to important parts of my research work.

Besides my working environment, I am grateful for the support of my friends. A Ph.D. thesis always has impacts on private life. Thus, I appreciate the perseverance and support of my friends, not only during the last three years. Therefore, big thanks to my long-term friends for their confidence, and also thanks to those who have been with me in all the good and bad times during my studies, which I started many years ago. I also want to thank my hosts and friends from the *Österman* family in Sweden for involving me in all their Swedish culture, events, and way of life.

Last but not least, and most importantly, I want to give my sincerest and honest thanks to my whole family. I am so proud to be a part of our union that we are, the way that we go together, and the time that we share. My deepest appreciation to my sister *Anne* for always being an essential and inspiring part of my life, and my parents *Kerstin* and *Stephan*, who supported me in my life, showed me the world, and gave me the freedom that allowed me to be who I am right now. I love you!

Contents

Acronyms

AAP	Atomic Adaptation Part
AIS	Adaptation Instruction Set
AM	Application Model
BISE	Business and Information Systems Engineering
BPMN	Business Process Model and Notation
BROS	Business Role-Object Specification
C-EPC	Configurable Event-driven Process Chain
C-YAWL	Configurable Yet Another Workflow Language
CROM	Compartment Role Object Model
CSP	Cloud Service Provider
DDD	Domain-driven Design
DSL	Domain Specific Language
DSR	Design Science Research
EM	Enterprise Modeling
EPC	Event-driven Process Chain
ERM	Entity Relationship Model
GPD	General Project Data
GPL	General Purpose Language
IoT	Internet-of-Things
IS	Information System
ISO	International Organization of Standardization
MDSE	Model-driven Software Engineering
OCL	Object Constraint Language
OMG	Object Management Group
OO	Object-orientation
RM	Reference Model
RoSI	Role-based Software Infrastructures for continuous-context-sensitive Systems
SUM	Single Underlying Model
UML	Unified Modeling Language
YAWL	Yet Another Workflow Language

List of Figures

List of Tables

Chapter 1
Introduction

"If developers were unlimited rational beings, the difficulties of software engineering would be dramatically reduced, since such godlike developers would write perfect machine code at once, obviating the need for all other languages." [136, p. 184]

Software engineering could be considered as a kind of applied arts: the creation of functional software in combination with visionary design and architecture. Software systems, created by such a high standard, should be the ideal goal of every software engineer's activities. So the software engineer's tools for achieving the goals are mainly modern methods, principles, languages, and models. Like constructing a spectacular bridge that shall hold for centuries and still be sustainable, reliable, and secure like on the first day, the construction of a software system can be compared with that metaphor. A well-designed, future-proof software system is able to withstand the circumstances that arise from the omnipresent changing context, new requirements (functional and non-functional), new security issues, changing workload, other application areas, and much more that demands highly flexible, resilient, and maintainable systems in order to survive not only a few but many years of usage and further evolution.

1.1 Motivation

The construction of software is a wide-ranged field. The construction process is part of small and large projects equally, differing in its details and extent. The process of software (architecture) construction is highly individual and necessary: every project needs its specialized analysis, planning, and implementation. A careful and thorough construction will lead to a stable and useful software artifact. However, neglecting such is prone to bad design decisions and implementations that result in non-usable and "fossil systems" [90, 221]. To further improve the extensive but important construction process of software systems, this thesis' subject aims for a new paradigm in software engineering: the paradigm of role-based software construction. The concept of roles is a powerful approach that is, nevertheless, often in the offside of attention. Roles are, in contrast to the traditional object-like concepts, dynamic and context-dependent. Instead of a static construction, roles can be used to establish dynamic and context-dependent software systems. The natural understanding of a role motivates this view. A role, as an individual and autonomous concept, is of enormous value when considering the whole software stack, from domain modeling down to the technical infrastructure and implementation. In particular, the field of conceptual software modeling combined with the role-paradigm is in focus of this thesis. It is aimed for a new way of modeling custom enterprise-specific software by adapting reference models via roles towards application models. The role is used as a conceptual (model) element to describe a standardized and uniform object in its different perspectives and contexts. Thus, the role can introduce individual behavior and structure to conceptual items. This fact can be used to improve the individualization of enterprise software, which motivates the thesis' objectives.

By developing a new role-based conceptual modeling approach for software construction, the current issues of the construction processes are tackled. This thesis develops a methodology for the development of enterprise-specific software by adapting reference models with roles towards tailored application models.

In this thesis, the role-based paradigm is introduced and applied on current, as well as new, modeling methods to make them more valuable and profitable for future software engineering projects, especially for the construction of enterprise-specific software systems with reference models.

1.2 Vision

The research project called *Role-based Software Infrastructures for continuous-context-sensitive Systems* (RoSI)[1] is home to this thesis' work. It is host to a multitude of research projects related to establishing the role paradigm into the software development stack. The project and its involved contributors share a common idea:

> "The central research goal of this program is to prove the feasibility of consistent role modeling and its applicability. Consistency means that roles are used systematically for context modeling on all levels of the modeling process. This includes the concept modeling (in meta-languages), the language modeling, and the modeling on the application and software system level." [`http://www.rosi-project.org`, acc. 04.2020]

As a part of the RoSI project, this thesis aims to improve the (conceptual) modeling of a software system and application structures as a sub-domain of software engineering. Within the field of software modeling and reference modeling, the thesis can contribute to the subject to prove and improve the applicability of roles as well as the role concept in general. The once established role concept in the domain of software modeling should be a primary driver of enhanced software quality. Once established, the role concept is an integral part of the software development stack,

[1] `http://www.rosi-project.org/`

this thesis' pioneer work is even more valuable, however, not only in academic achievements but also in a practical and industry-related application contexts.

> In a world where a role is understood and implemented as a central modeling and programming concept, software could be built with more detail, more efficiently, and more appropriateness. The establishment of the role paradigm provides a fundamental improvement of the software engineering discipline.

The thesis is neither suitable nor intended to establish and fulfill the vision of future role-based software engineering on its own. Nevertheless, it contributes its small and vital part towards a consistent and continuous use of the role concept. Role-based conceptual modeling, as presented in this thesis, helps software systems to get a higher added value by providing a more versatile and simple way of implementing and using them. Even if individual contributions of this thesis might lose their importance in the future, the idea and methodology of role-based conceptual modeling for the adaptation of reference models should remain.

1.3 Outline

The remainder of this thesis is as follows.

In Chapter 2, the main aspects of the thesis are presented. It consists of the foundational research coverage (e.g., research questions, problem domain, objectives). The main principles, concepts, and contexts are explained. Further, the chapter is concluded with a summarized and concisely explained overview of the BROS methodology (which are then further explained in the follow-up chapters).

In Chapter 3, the necessary scientific background is described. It contains various sections that provide insights into the modeling area and the subject of software engineering, but also reference modeling, and the derivation of an application model is part of this chapter. Further, the concept of roles is introduced in a more detailed way, and different applications of the role-concept are described. As in the context of this work, the modeling of temporal descriptions, behavior, and events are further considered in depth. Finally, since reference modeling belongs to the discipline of information systems, the work and its results are embedded in the design science research context.

Chapter 4 focuses on the new BROS modeling language, used for reference model adaptations and software system design. It utilizes a (possibly OO-based) structural domain model and specifies the adaptation with the help of roles, scenes, and events. On the one hand, the chapter includes several sections for the detailed specification of all the single modeling language elements, and on the other hand, the possible application and relationships between these elements to finally construct a valid model (or a valid adaptation of a given reference model).

The following Chapter 5 concentrates on the BROS reference model adaptation method with the help of the previously introduced modeling language in Chapter 4. It states the necessary steps for a successful adaptation by establishing multiple adaptation phases in combination with an instruction set. This chapter also addresses best practices and applications.

With Chapter 6, the BROS modeling practice is further specified. This practice includes the three main parts of a comprehensive style guide for models using the BROS language (Chapter 4) and the adaptation method (Chapter 5), the verification and validation rules for adaptation, and the guiding modeling principles when designing role-based models with BROS.

This is followed by Chapter 7, where a comprehensive case study is presented. This chapter shows how to use the language to adapt a reference model (served by a cloud service provider enterprise) towards new business functionality and a changing context on an organizational enterprise level. Besides the adaptation, it shows the benefits and drawbacks of the language and adaptation, compared to a traditional UML-based adaptation.

In Chapter 8, the thesis is concluded with three major points: (a) the contribution that summarizes the content of this thesis and highlights the aspects that should be remembered; (b) the limitations and non-application regarding the language and the adaptation method as a whole; and (c) the outlook into future work and particular research fields that are worth to be explored further.

The Appendices A, B, and C of this thesis introduce developments and side topics that are aligned with the thesis' subject but do not contribute to the main concepts. They cover the related publications, contributions by students' work, and advanced topics (e.g., a modeling editor development).

Chapter 2
Conceptual Foundations

"It appears that the role concept is a truly original one, one that cannot be emulated by any of the better established conceptual or object-oriented modelling constructs." – [244, p. 104]

Abstract Research is always in need of a conceptual background that describes the related context, organization, problem domains, research questions, and other subjects. Driven by these basic needs, the actual research and its artifact can evolve to a substantial and well-founded result. The research, as well as its artifacts, described in this thesis, is based on such a set of foundational subjects to maintain transparency, clarity, and credibility. In this thesis, the role paradigm is used to adapt a reference model towards an individual application model. The artifacts of this research are mainly the language and the method needed for the adaptation. This chapter provides an overview of the foundational research context and the main drivers, namely the research questions, problem domains, the goals, and solutions as central aspects of the developed language and method. Additionally, it will provide a comprehensive overview of the artifact and the results.

2.1 Working Context

The construction of large software systems is always in need of blueprints and construction plans that enable different views onto the future system and allow to analyze various aspects. Therefore, *models* are the common tool of software engineering to handle the upcoming system's complexity and specification. Simply said,

> "A model is a set of statements about some system under study" [230, p. 27]

However, the specified set of statements may vary widely in its granularity and complexity. Models are an important tool for designing, but also developing, a software system or its extensions. Thus, the design and specification of a future software system (or its parts) with models are in focus, especially in the area of software architecture [20], a sub-area of software engineering.

Mastering the art of software engineering enables a powerful design and construction of software systems. A clear structure and well-specified properties of such a system, described as an expressive and powerful model, is a prerequisite for all stakeholders that contribute to development. In general, this thesis uses the software engineering definition from the *International Organization of Standardization* (ISO) [133].

Definition 2.1 Software Engineering "The systematic application of scientific and technological knowledge, methods, and experience to the design, implementation, testing, and documentation of software to optimize its production, support, and quality." [133, p. 418]

Individual solutions of software systems are driven by the significant impact of individual and enterprise-specific business requirements, which can not be satisfied by standard software products. In enterprise modeling, multiple aspects of an enterprise are expressed with not a single but multiple models. The enterprise modeling area has its focus on the development and evolution of the enterprise's software systems structures and behaviors on an organizational level [89]. It is common that existing business requirements (i.e., business processes) are represented in behavioral models that specify the actual behavior of the underlying software system. In contrast, the structural side of the software system often models business objects (instead of business processes) as the major informational elements. Further, the construction and maintenance of the software system are usually combined with software engineering methods, specified and tailored for the enterprise's individual purposes. Often, the system's behavioral models (representing the business processes) are separated from the system's structural models (consisting of, e.g., business objects). Thus, it is a challenge to efficiently apply software engineering methods for the whole enterprise model: to construct (and evolve) the software system's parts with respect to the enterprise-specific behavioral requirements. In general, this thesis' definition of enterprise modeling is derived from Sandkuhl et al. [214].

Definition 2.2 Enterprise Modeling The process of describing the current or future state of an enterprise regarding the commonly shared enterprise knowledge of the stakeholders. The resulting enterprise model consists of a number of related models, each focusing on a particular aspect of the enterprise. (derived from [214])

To individualize generic software system solutions to enterprise-specific versions, so-called reference models, capture business domain knowledge for reuse and tailoring, and are used as templates, either of the structural system part (structural reference model) or behavioral system part (process reference model) [78]. By adaptation, established or given reference models are tailored to the enterprise-specific application models, which can be used for individual software development and maintenance of the enterprise model.

Several adaptation approaches are available to derive the enterprise-specific application model from the reference model. However, especially concerning the structural software system's part, current approaches have issues adhering to the predefined structural reference model and including the enterprise's individual requirements. A solution to this issue is the novel structural reference model adaptation

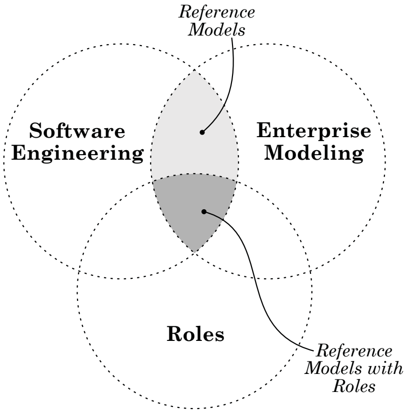


Fig. 2.1 The three main context areas regarding this thesis' subject: *software engineering* as the practice of constructing proper software systems, *enterprise modeling* as the main driver of individual requirements, and *roles* as the new concept and enabler of both. In-between, the application area of reference models

method via the utilization of the role paradigm. The concept of a role (as a new and powerful modeling construct) allows the application model to solve problems that cannot be solved without it (or at least only in a complicated way). Often, software engineers and business analysts are trying to construct a software system in a combined approach, but they need to make trade-offs in order to implement the different stakeholder's ideas. In contrast, roles have the ability to solve this issue. The specified and well-thought software constructs of the software engineers are able to represent different business process contexts without massive changes but dynamic adaptation. Thus, roles exist in both worlds: the static structure and the dynamic behavior. Although the role concept is not a new idea and already established in the last decades, the role paradigm never reached full attention and potential for design time modeling purposes.

This thesis aims to take the next step and proposes a solution for utilizing roles to overcome the discrepancy between software engineering and enterprise modeling. Figure 2.1 illustrates the different contexts.

2.2 Research Coverage

The proper adaptation of reference models is a non-trivial task. Various methods lead to many different application models, each with different benefits and drawbacks. In this thesis, a new methodology is presented that enables a new way to derive application models from reference models in a more flexible and non-invasive way. The development of this new methodology was done following the dedicated research methods (see Section 2.2.4). This includes accurate problem statements, concrete solutions to these problems, and a transparent way towards method development and the assumptions taken. This section covers the main drivers and aspects of the new approach's development.

2.2.1 Problem Domain

The development of a new adaptation method is not only done due to increasing interest. There are major issues in current adaptation regarding structural reference models. These issues are manifold and often range from conceptual down to the technical scope. Three major core problems were identified regarding the current state of reference model adaptation. However, further details of the respective approaches and subjects are given in Chapter 3.

Systematic adaptation First, no method adapts a structural reference model systematically. Often, driven by new business requirements, a structural model is modified in an ad-hoc manner. The lack of a systematic method for adaptation manifests in incomprehensible adaptations, not traceable and invasive modifications of the previously standardized structure. In literature, several works consider the classification of adaptations of reference models (e.g., [77], [126]). However, they do not go into detail regarding the actual adaptation implementation. Further, those works being closest to a systematic adaptation method consider either only behavioral models (e.g., [71, 99, 203, 212]) or models on an abstract level with missing required implementation details (e.g., [24, 25, 205]), or are too specialized for a general application (e.g., [227]).

Predefined structures Second, there is a discrepancy between the aims of software engineers (or, more precisely, the software architects) and business engineers (especially the business analysts) [87]. Software engineers tend to standardization, uniformity and clear separation of concerns. Regardless of either the construction of new systems or modification of an already existing system, a structured way of building new software (or parts of it) is always preferred than ad-hoc construction. The possibility to apply existing and tested patterns, structures, and frameworks are enormously valuable for (further) development of a system. Thus, reference models are also a suitable source for such standardization as they define the basic structure, developed and tested by domain experts. Further individualization via an adaptation method requires, however, the strong coupling with the given structure. In contrast, business engineers often propose a business logic point of view. Processes and activities of the system are the main focus. As a consequence, new behavior functionality must be implemented, e.g., due to new legal issues or new feature requirements. This leads to a situation where the new behavior has to be mapped on existing (or pre-defined) structures. This issue is subject to the conflict between software engineers and business engineers. On the one hand, the new functionality should be derivable from the reference model (or any other template of an existing structure). On the other hand, the given structure must not limit the possibilities of new functionality or system evolution.

Behavior implementation Thirdly, current structural reference models (and their adaptations) do not concern the business logic component (behavior). On the one hand, classic models for software development consist of detailed specifications for structural characteristics. A popular choice is the *Unified Modeling Language* (UML) class diagram [187] or similar models from the *Object Management Group*[1] (OMG). They state the exact implementation details and support a wide range of architectural and design decisions concerning the structure. However, with regard to business logic, structural models lack the possibility to support behavioral changes. On the other hand, business requirements on an organizational level (as the main driver of changing business logic) have to be implemented within often already existing software systems. For example, changing legal requirements may lead to different business logic on how to complete a transaction. However, the new business logic needs to be implemented in a pre-defined structure, which is the already existing banking application. Often, the new behavior is accurately specified as process models, e.g., via *Business Process Model and Notation* (BPMN) diagrams [184] or petri nets. Nevertheless, modeling the behavior often neglects structural specifications and, if at all, is difficult to apply on an already existing structure. Languages focusing on this issue are either not flexible enough (e.g., *Executable UML* (xUML) [174]) or leave out the goal of architectural software construction (e.g., *Business Process Execution Language* (BPEL) [191]). Further, as another issue regarding business logic, temporal information can hardly be included in structural models. However, in order to model accurate business logic in structural models (as described in the second issue), one needs to use statements that are dependent on certain temporal conditions. There are various works, languages, and frameworks that cover the issue of representing temporal information in structural models (e.g., [8, 42, 73, 100, 112, 190, 197]). However, most approaches are not usable for adaptation of existing (reference) models as they are too specific, too abstract, or do not cover temporal information as part of structural models for development. Finally, a structural model that is not able to represent business logic in any way may lead to problems in fulfilling the new requirements, depending on the actual flexibility and adaptability of the software structure.

2.2.2 Research Questions

The three core areas of the problem domain stated above show the difficulties of finding or developing a proper adaptation method. As far as known, no existing adaptation method serves to all three areas simultaneously and sufficiently. However, to use structural reference model adaptation as a software engineering tool (especially for software architecture for the construction of software systems regarding various business requirements), a distinc and reliable adaptation method is required.

The missing adaptation method, its language, application, rules, and guidelines are in the focus of the thesis. To analyze, describe, and solve the adaptation problem is the essential requirement and goal of this work to overcome the issues and achieve the visions stated in Section 1.2. This fact leads to the research question answered within this thesis.

[1] `https://www.omg.org/`

? Research Question

How can the adaptation of a structural reference model be conducted systematically and in strict compliance with the structural specification, including enterprise-specific behavioral information?

The main research question is very broadly defined and needs to be divided further in order to grasp the full potential of the problem domain. Thus, three sub-questions can be derived from the main research question:

1. How to adapt a structural reference model in a systematic way?
2. How to reuse standardized, object-oriented reference models for adaptation?
3. How to express certain business logic in a structural adaptation?

The sub-questions are referring to the individual core issues of the problem domain that are contributing to the main question as a frame for the whole research process.

2.2.3 Research Objectives

In this thesis, the proposed research questions are resolved and explained in detail. For this, three objectives are assigned to the core issues: (a) the introduction of a systematic adaptation method, that (b) may reuse established (industry standard) structural reference models, and that (c) provides a set of powerful modeling concepts for expressive statements. An assignment of related chapters to the three objectives is shown in Figure 2.2.

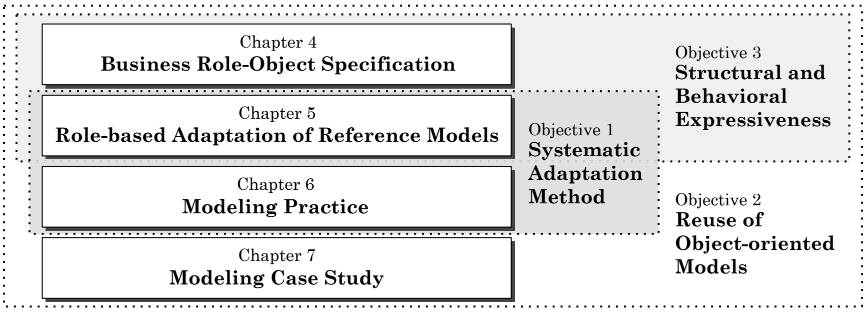

Chapter 4
Business Role-Object Specification
Chapter 5
Role-based Adaptation of Reference Models
Chapter 6
Modeling Practice
Chapter 7
Modeling Case Study
Objective 1
Systematic Adaptation Method
Objective 3
Structural and Behavioral Expressiveness
Objective 2
Reuse of Object-oriented Models

Fig. 2.2 The research objectives assigned to the thesis's chapters

2.2.3.1 Systematic Adaptation Method

The need for a reliable and helpful manual with steps, rules, and guidelines for an accurate adaptation towards enterprise-specific application models is a prerequisite for successful software system construction, especially for large systems. A systematic approach has, in contrast to an ad-hoc adaptation, the advantage of being transparent in its changes and implementation. Further, a systematic adaptation leads to higher quality results as the individual steps are comprehensible and specified, whereas the ad-hoc manner may follow wrong design decisions due to uncoordinated changes.

This objective leads to the requirement that the role-based adaptation method needs to be systematic in its application. Thus, the role-based adaptation method is embedded in a "user manual." It consists of detailed instructions for the adaptation (based on a specific focus) and provides style specifications, adaptation guidelines and rules, patterns, and principles. This solution to the objective is proposed in Chapter 5 and Chapter 6.

2.2.3.2 Reuse of Object-oriented Models

In fact, most enterprises already use their individualized and enterprise-specific IT systems for their in-house processes. Adaptation of structural models, however, only becomes useful if these already given object-oriented (OO) models can be suitably integrated for further construction. This affects both the construction of new systems via the usage of reference models or the advancement of an existing software system (that may also be based on a reference model). Regardless of the (possibly powerful) functionality of a new adaptation method, if it can not be applied to a given object-oriented system structure, it is not appropriate since no one could use it, and enterprise-specific adaptations may not be possible.

Thus, it is an objective to enable the application of an adaptation method on existing and established OO-based models. It has to be possible to adapt object-oriented (reference) models towards enterprise-specific application models. The role-based approach solves this issue by using the role concept for the enterprise-specific adaptation of OO-based models. This adaptation method, together with the needed modeling language and guidelines, is subject of all the core chapters (that are Chapter 4, Chapter 5, and Chapter 6), and is exemplified in Chapter 7.

2.2.3.3 Structural and Behavioral Expressiveness

If a modeling language has too many modeling elements, it can lead to less usability, applicability, and acceptance due to error-prone complexity [11, 179]. In contrast, strong expressiveness is needed for suitable modeling of complex situations. Thus, it is necessary to support a wide range of possible statements while preserving the right balance between complexity and usability. However, for technical modeling (as it is for software architecture), a certain level of complexity must be taken into account for a proper specified software system development.

This thesis introduces several modeling concepts, related to the role-based paradigm, to model statements for both points of view: the technical structure (for software engineering) and the business logic (for business modeling). The thesis' approach strives for a high expressiveness, possibly at the cost of higher complexity, to model structural and behavioral information. This allows the modeler to benefit from more possible statements regarding the targeted system specification. For the role-based approach, the modelers should be enabled to use the maximum of expressiveness on their own responsibility, as they also may choose to limit themselves to certain concepts. The concepts of the approach and its application regarding the expressiveness is introduced in Chapter 4 and in Chapter 5.

2.2.4 Research Methodology

As a fundamental paradigm, the research in this thesis is considered as part of the engineering discipline (more precisely in the area of computer science engineering), rather than as social science, as work on information systems and enterprise modeling is often regarded as such. Current issues in the research field of business and information systems engineering (BISE) [89] and enterprise modeling [86] contribute to the chosen research method. In this thesis, the focus is on practice-driven solutions, which also includes creative design and development of solutions to problems. This pragmatic, "forward"-oriented engineering research might not be the traditional way, however, solves non-trivial problems efficiently.

Nevertheless, the research can be regarded as an instance of *Design Science Research* (DSR), a structured process framework for producing practice-driven artifacts for (organizational) information systems effectively and efficiently [195]. Although the thesis' solution did not follow the DSR process accurately, it was used as a guideline for the new approach's development. In particular, the work of Sonnenberg and vom Brocke [241] was utilized as an iterative guiding method

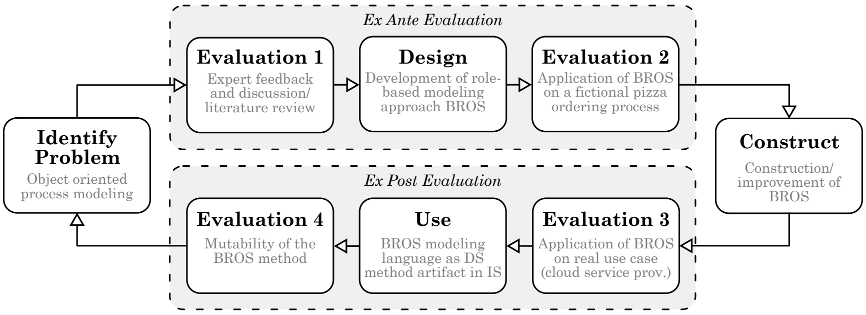

Ex Ante Evaluation
Evaluation 1
Expert feedback and discussion/ literature review
Design
Development of role-based modeling approach BROS
Evaluation 2
Application of BROS on a fictional pizza ordering process
Identify Problem
Object oriented process modeling
Construct
Construction/ improvement of BROS
Ex Post Evaluation
Evaluation 4
Mutability of the BROS method
Use
BROS modeling language as DS method artifact in IS
Evaluation 3
Application of BROS on real use case (cloud service prov.)

Fig. 2.3 DSR development and evaluation activities of BROS ([222], adapted from [241])

for increasing the scientific and practical value of the new approach. The used DSR process describes the usual path from the problem identification, over several iterations of construction and evaluation, until the final release is completed (see Figure 2.3). By following the guidelines, it was possible to

> "[. . .] further improve [the] approach for the implementation in a naturalistic business environment and use by modeling practitioners" [222, p. 4]

A complete review regarding the DSR aspect of the thesis' approach can be found in Schön et al. [222] where the details are stated in-depth.

Concerning theories in software engineering methods, the thesis' approach is positioned in correlation with Dittrich [67] as a

> "[. . .] software engineering research [that] develops principles, guidelines, (mathematical) tools and techniques to be applied by software engineers" [67, p. 228]

Thus, the new approach, in a way considered as a software engineering method, provides

> "[. . .] complex related sets of explicitly formulated understandings (notations, modelling languages, concepts, area of application, coverage, the mathematical side of the theories), explicit rules (processes, task descriptions, techniques, etc.), and more or less explicitly stated teleoaffective structures (principles, perspectives, theories in the sense of what software engineering is about)" [67, p. 226]

In this thesis, it is focused on the former two parts: the understandings and explicit rules for the application of the new method.

2.3 Role-based Reference Model Adaptation

During the implementation of object-oriented software systems, traditional building blocks are classes and their relationships (most importantly, the inheritance). These concepts can be used to successfully model and construct software systems, defining its inner parts and how they interact with each other. Although regarded as an intuitive concept in the object-oriented world, roles are rarely used in object-oriented modeling. However, the role concept is neither a new concept nor a niche concept; many research work is done considering the role concept (e.g., [16, 85, 151, 209, 246]). Nevertheless, due to a non-uniform understanding of "the role" as a concept, roles are used in very fragmented ways [149]. If a common ground is made, roles have the potential to improve structural models by their dynamic nature.

> "Roles can enhance the overall process of system construction as they deliver the possibility of adding structural and behavioral elements to an object, allowing its instances to use this role (and other roles) at some time during their lifetime without changing their identity." [220, p. 1448]

In this thesis, the role is introduced as an adaptation mechanism for reference model adaptation. The envisioned role-based approach was firstly presented in Schön [220].

The usage of the role as a dedicated concept allows multiple advantages compared to the traditional object-oriented modeling:

- Structure can be combined with behavior, due to the role's dynamic nature
- Roles can refine a fixed object dependent on their context, thus, making the application able to adapt to different contexts
- By fine-grained specification, roles allow a transition from conceptual modeling towards technical system design
- Roles can be applied to the most types of objects in models, thus, work on established object-oriented models as well

Several works and research highlight the beneficial usage of roles in modeling and show how roles can improve several applications in software engineering[2]. In conceptual modeling, often, domain models are used to model the system's structure and behavior. These domain models cover the main entities of a domain and their interaction with each other.

> "In an enterprise context, the main objects of domain models (objects regarding the domain, means classes not instances) typically need adaptation depending on context or use case. Specifically, in the case of so-called reference models, which aim to capture domain knowledge at a more generic level, adaptation is needed to derive the so-called application models from the reference." [220, p. 1448]

Thus, in this thesis, the adaptation is made via roles. The role's nature of being dynamic, context-dependent, and behavioral by default makes it a promising mechanism to adapt a given reference model into an enterprise-specific version. For that, the provided objects of the reference model are (context-dependently) adapted by fulfilling a role that refines (or changes) its internal structure and behavior. The native role mechanisms are used as a substitution for traditional OO-based adaptation mechanisms, e.g., ad-hoc and invasive[3] changes. In traditional OO-based modeling, invasive changes can often be avoided by using a frequently used non-invasive concept: inheritance[4]. Nevertheless, inheritance has several significant drawbacks regarding its ability to refine and express the enterprise-specific needs [85, 222]. Additionally, one should always consider that inheritance leads to new (class)types. Thus, the adaptation of single objects while keeping the foundational identity is not possible. In contrast, by adapting an object via a role, the object itself, and its interaction with other model elements are changed without sub-typing. [220]. Further, an applied role can easily change its foundational (playing) object at runtime by proposing its structure and behavior to the object. An object can be adapted by fulfilling a role, which, e.g., adds an extra attribute or method (that the object can use as long as playing the role).

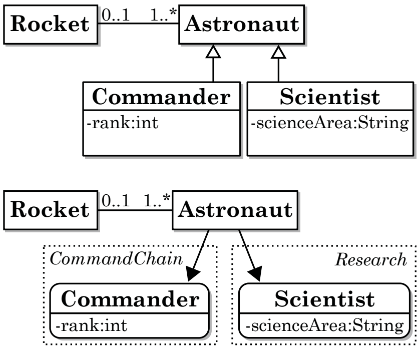
Figure 2.4 illustrates the two different concepts. The original object (the `Astronaut` within a `Rocket`) gets adapted in two different ways. In the upper part of the figure, two subclasses (`Commander` and `Scientist`) are implemented. However, this solution lacks in conceptual accuracy due to the mentioned issues: (a) an instance can be

[2] Section 3.3 states more background information on roles and their application.

[3] That is, merging the adaptation with the standardized template or reference model.

[4] However, other, similar concepts are still possible as well.

Fig. 2.4 An example scenario regarding non-invasive adaptation with inheritance (top) and role fulfillment (bottom)



either commander or scientist, but there can not be a scientist who is a commander simultaneously (type-wise), and (b) the identity is not the basic astronaut's identity but the identity of an astronaut's specific function (e.g., the scientist). In contrast, when modeling this situation with roles (the bottom part of the figure), one keeps the basic identity of the astronaut. Both roles (`Commander` and `Scientist`) are added as the object's characteristics in certain contexts. This also allows the actual astronaut's identity to be part of both functionalities (that is, e.g., the "commanding scientist"). This ensures accurate conceptual modeling while introducing technical depth into the model. The whole adaptation is then made in a non-invasive way, the instances of adapted objects take on the roles but retain their identity, the adaptations can easily be traced back and recorded, and can be applied on current existing OO-based reference models. Additionally, other roles may be introduced for different contexts in the future without major changes to the basic model or its adaptations. For example, in another context, the astronaut may be needed to be adapted to represent a cosmonaut. Thus, a role `Cosmonaut` could be used to adapt the generic version of the `Astronaut` object and keep the remaining (and possibly already adapted) parts unchanged.

The benefits of using roles in conceptual modeling for software system design are advantageous, especially for the case of reference model adaptation (e.g., preserving the object identities, flexibility, and hierarchy-independent conceptions, combination of structure and behavior). With the thesis' role-based approach, the typical approach of the two tiers within the adaptation process (see Figure 2.5) is followed. This two-layered system of models ensures separation of the initial reference model with its template elements (objects and relationships), as well as their user-defined

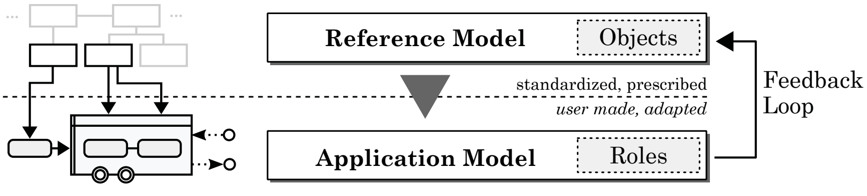


Fig. 2.5 The concept of role-based adaptation of reference models (figure adapted from [220])

adaptation, as long as no parts of the original reference model are modified invasively. The idea behind a strict separation is twofold: (a) the once adapted application model is fully covered, and adaptations can be traced back due to non-invasive adaptation processes, and (b) the user adaptation is handled with the possibility of multi-usage and variant development [220].

The Reference Model Tier

The reference model maps the overall domain knowledge in terms of the domain's (business) objects. As a reference model is often standardized, it is, in most cases, in the hand of the reference model owner and, therefore, out of control of the model's user (modelers who adapt the reference model) [220]. In Figure 2.5, this model in above the horizontal dotted line. Thus, the reference model can evolve independently from the expectations of the model's user. A non-invasive approach (like the role-based adaptation) allows for tracking and transferring changes of the reference model towards the application model easily.

The Application Model Tier

The application model represents the lower section in Figure 2.5, below the dotted line. The application model represents the individualized, enterprise-specific variant of the reference model. Following a selection of relevant (business) objects, the adaptation implementation tailors the generic reference model. Roles, as proposed in this thesis, are the main concepts for the adaptation. Selected objects are adapted by fulfilling contextual roles, allowing them a refinement (or modification) of their structure and behavior. A feedback loop towards the reference model owner can be used by the user to indicate a necessary change of the reference model (instead of modifying the reference model directly, e.g., missing objects) [220]. This guarantees an independent and non-invasive adaptation as far as possible.

2.4 The BROS Methodology

BROS

To execute the role-based reference model adaptation, several components are necessary. This thesis develops the individual concepts needed for an initial approach. They are combined under the term *Business Role-Object Specification* (BROS), which states the approach of modeling role-based application models by reference

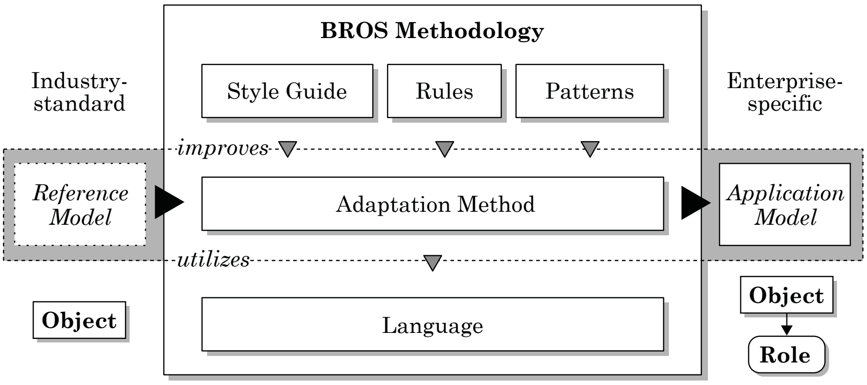


Fig. 2.6 The components of the BROS methodology

model adaptation. The full name of BROS is based on the idea of using roles as a specification of business objects. The idea is to transfer "regular" business objects in business role-objects in a specified way, which serves the adaptation purpose. The role attached to the business objects allows it to contribute and participate in several use cases in different ways (structure and behavior), depending on the use case. Thus, the Business Role-Object Specification unites this idea at its core.

Driven by the idea of solely using roles for adaptation of regular business objects, the BROS components contribute to the overall shared goal, called the *BROS methodology*. For this, the definition of the *Lexico Oxford Dictionary* is followed:

Definition 2.3 Methodology "A system of methods used in a particular area of study or activity." [`https://www.lexico.com/definition/methodology`, acc. 04.2020]

Thus, the BROS methodology consists of a "system of methods" that aims to solve the role-based reference model adaptation as "a particular area of activity." The combined set of BROS components unite the role paradigm in conceptual modeling and adaptation, illustrated in Figure 2.6, as described in the previous Section 2.3. Specifically, the current main components of the BROS methodology are the following:

1. An industry-standard reference model, consisting of objects, is adapted by the *BROS adaptation method* towards a role-based and enterprise-specific application model (cf. Chapter 5). The adaptation method provides the main ingredient to derive the application model step-by-step
2. The adaptation method relies on the *BROS language* to express the model statements within the adaptation (cf. Chapter 4). With the help of the role-featured modeling language, the role-based application model can be expressed (for both cases during the adaptation and as a result)
3. The *BROS modeling practice* (containing the style guide, rules, and patterns) is used to verify, validate, and improve the adaptation process (i.e., the adaptation method process) and, therefore, the resulting application model (cf. Chapter 6)

The BROS methodology, as a framework for role-based reference model adaptation, identifies itself by its goals and concepts. The individual components can be substituted, however, the whole BROS methodology stands for a much more visionary idea of role-based conceptual modeling.

The individual concepts are explained in-depth in the further chapters of this thesis. All components of BROS contribute to the role-based adaptation paradigm, called as the BROS methodology. If components are changed, replaced, or no longer used in the future anymore, the BROS methodology and its implications may still hold and introduce the role-paradigm into the reference model adaptation subject to fulfill the thesis' vision stated in Section 1.2.

Chapter 3
Extended Background

"Minds are limited by what they know." – *[13, p. 106]*

Abstract Technology, mainly IT-related technology, is subject to frequent and massive change. Thus, the technologies that are established today might not be state-of-the-art in a couple of years anymore. New technologies (that includes, e.g., methods, tools) may vanish or emerge over a few years. The development and understanding of the role-based adaptation method and its language, which is described in this thesis, needs further input from different related areas regarding current end existing technologies. As a consequence, this thesis' method for software engineers itself is a step further into new technology, that possibly makes already established technology (partly) obsolete. Since the method is an output of research, the related work together with their benefits and drawbacks is also highlighted. This chapter serves two major purposes:

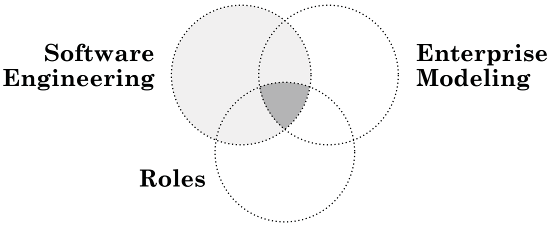
1. to provide the necessary knowledge required for the new language and adaptation method described in this thesis, and
2. to present related work that inspired and enhanced the new contributions.

In general, the background subjects are oriented at the three main context areas described in Section 2.1: *software engineering*, *enterprise modeling*, and *roles*. Further, a top-down approach is utilized by becoming more specific in the background subjects successively instead of covering the entire (but not in-depth) field of research. Specific related research and work, which influenced this thesis' work, are highlighted in the last Section 3.4.

3.1 Model-driven Software Engineering

This section provides an overview of current methods of using models for software engineering. Thus, it describes the main properties of models and their application in different use cases and by various stakeholders.

The research field of *model-driven software engineering* (MDSE) is vast, and many different works specify and improve this area (e.g., [12, 37, 175, 194, 238, 268]). Viable related terms are used in different use cases (e.g., model-driven software development, model-driven development, model-driven architecture). The whole subject of MDSE considers models as more than a pure documentation of functionality and design of software [231]. Instead, models are a core mechanism to deploy software artifacts on a platform-independent base that not only describe but define the software, possibly to overcome repetitive and automatable development process steps. Thus, MDSE is often mentioned in combination with the practice of model transformation [233], in order to transform high-level and abstract models ideally into the source code (models) or something in-between. The *Object Management Group* (OMG) defined a set of standardized methods to implement MDSE in practice, consisting of several other standards [185]. Nevertheless, in this thesis, software development based on detailed models is considered to be part of the MDSE because the software is built around the model (not the model as a simple description of the software). The focus is on using models for software architecture and design as a precondition for later implementation based on those models.



3.1.1 Models for Software Design and Architecture

Modeling is an essential part of software engineering and enterprise architecture. Using models for the development of software has the advantage of

> "[. . .] using concepts that are much less bound to the underlying implementation technology and are much closer to the problem domain relative to most popular programming languages." [231, p. 20]

The mentioned problem domain, considered as business logic, is the main driver of requirements and changes of newly build (or existing) software systems [181]. Such business logic (and its models) is often separated from the final architectural model (that depicts the structure). On the one hand, behavioral models (that describe the domain's behavior) are typically the models developed first in MDSE. Conventional models for representing the domain's business logic are the OMG's UML behavioral models like BPMN, use case diagrams, or activity diagrams. In a more advanced state of software development, those behavioral models often take over a more passive role for checking back the right functionality of produced artifacts and implementation, however, they are not used for blueprints of the system's structure. On the other hand, the commonly used structural models of the OMG (like UML class diagrams, component diagrams, profile diagrams, or object diagrams) are positioned in the later phases of software development (and can partly be derived from the behavioral models). Possible transitions are behavioral models close to implementation like state diagrams or sequence diagrams. Still, this separation could lead to several problems that were described in the previous chapter's section 2.2.1.

Like behavioral models, the structural models may differ in their granularity and complexity. When using models, especially structural models, it is crucial to consider the use case and stakeholders for the model. While the programmers might need a fine-grained and completely specified structural model, the architects and designers of software often deal with more abstract models that cover coarse-grained and under-specified views of the structure, which allows both, a better overview on the whole system's parts and their connection to each other. While the developers of the software usually use the fine-grained models as a blueprint for their work, the software architects and designers are used to develop those models as part of the software architecture process [97, 167]. Thus, they are responsible for constructing the software structure and implementing their design decisions into the model. That way, wrong decisions in that development stage are hard to fix with the further progress of the development. In contrast, these initial structural models and their design decisions regarding the software architecture are essential for their business value [90, 221]. Well developed and maintained structural models are more than just a blueprint for developers. They are also an essential factor and decision support system for the future development of an enterprise (e.g., [28, 32, 128]) as well as the enterprise's market dominance and ability to innovate. An appropriately developed and used software system can then be an asset of significant value for the enterprise [90].

Thus, models that represent and specify vital design decisions should enable clear communication among various stakeholders and should be used to control the development process according to the taken design decisions and their metrics. Furrer [90, p. 273] specifies a set of properties that a model should be opted for:

1. **Clarity.** "The concepts, relationships, and their attributes are unambiguously defined and understood by all stakeholders."
2. **Commitment.** "All stakeholders have accepted the model, its representation and the consequences (agreement)."
3. **Communication.** "The model truly and sufficiently represents the key properties of the real world to be mapped into the IT-solution."
4. **Control.** "The model is used for the assessment of specifications, design, implementation, reviews and evolution."

A model that does not support *clarity* is ambiguous, and an ambiguous model can lead to severe errors. The strict and precise definition of all model's elements and their relationships are required to be able to use the model reliably and to avoid wrong decisions. The model needs the *commitment* of all relevant stakeholders for both: understanding the model and the decisions taken. This is crucial, as an ignored stakeholder may lead to wrong individual expectations of the outcome. In contrast, if every stakeholder is committed to the model, all of them are aware of the decisions and the consequences. The *communication* aspect of models is not linguistic communication but considers the viability of the modeled concepts towards the real world. Therefore, a final model should be able to transfer the requirements of the real world into its elements, as well as be consistent with the concepts of the real world. Last, the model's purpose is to *control* development. The controlling considers both: the initial decision-making phase as well as the assessment of the results. A model that can not be used for controlling is often not usable for MDSE but at most as an aid to understanding.

Further, as another set of model properties, Selic [231, p. 22] also proposes five major properties that define a proper model for MDSE:

1. **Abstraction.** "By removing or hiding detail that is irrelevant for a given viewpoint, it lets us understand the essence more easily. [...] Abstraction is almost the only available means of coping with the resulting complexity."
2. **Understandability.** "It isn't sufficient just to abstract away detail; we must also present what remains in a form [...] that most directly appeals to our intuition. [...] A good model provides a shortcut by reducing the amount of intellectual effort required for understanding."
3. **Accuracy.** "A model must provide a true-to-life representation of the modeled system's features of interest."
4. **Predictiveness.** "You should be able to use a model to correctly predict the modeled system's interesting but non-obvious properties [...] This depends greatly on the model's accuracy and modeling form."
5. **Inexpensive.** "A model [...] must be significantly cheaper to construct and analyze than the modeled system."

These five properties of Selic [231] correlate with the previous four properties proposed by Furrer [90] (e.g., *accuracy* is nearly equivalent to *clarity*). Interestingly, the point of *inexpensiveness* is not often mentioned regarding models for software architecture and design. Especially considering MDSE, this aspect may differ since the "additional" effort may pay off in later implementation. Outside MDSE (that is using models only for descriptive purposes), the *inexpensiveness* is more important.

Many other works classify the properties of models and take a look at the (philosophical) nature of models (e.g., [39, 105, 153, 214, 230]), and the resulting research field of models, modeling, MDSE, and how to use models for software development is vast [50], especially for software architecture and design.

3.1.2 Conceptual Modeling and Domain Models

In a more upstream development process, there might be the need for an abstracted model to identify the core parts of the domain's structure (especially the business objects). For that case, the so-called *conceptual models* are often used to represent an overall and abstract view on the system's structure and behavior in a formalized or systematic way [271] intended to provide easy understanding and communication about the domain's concepts for all stakeholders.

3.1.2.1 Conceptual Models

Conceptual models are more abstract and not intended for implementation but providing an overview of the future system. Krogstie [146] describes conceptual models as

> "[. . .] a description of the phenomena in a domain at some level of abstraction, which is expressed in a semi-formal or formal visual (diagrammatical) language." [146, p. 1]

Thus, a conceptual model can be regarded as a domain representation in a non-uniform but specified way. Derived from Mylopoulos [182], this thesis uses the following definition of conceptual models:

Definition 3.1 Conceptual Model A formal description of an aspect of a system for the purposes of understanding and communicating the semantics of the system. (derived from [182])

According to this definition, conceptual models are not intended to be understandable and readable by machines but humans primarily. The social aspect of conceptual models might be the most important one as a conceptual model represents the future software system in a manner that can be read by human beings, thus, it can describe the stakeholder's expectations and requirements. The resulting conceptual models are regarded as beneficial for several purposes [210], e.g.,

- Minimization of possibly incomplete, unclear, or inconsistent requirements
- Acting as a basis for validation
- Providing documentation and guides the objectives

Conceptual models are, by definition, rather abstract and generic. Conceptual models can differ in their level of abstraction and detail:

> "In terms of content, different models provide different levels of abstraction: some of them define a few high-level components only, whereas others decompose these into more primitive components and services, and provide more details about the interconnection and interaction of these components and services. The level of abstraction of a model is closely related to its level of detail, or granularity: less abstract models tend to provide finer granularity than the more abstract ones." [178, p. 488]

There are so many different conceptual models in the software and information system domain that there is hardly any full-fledged definition that would meet all different kinds. Nevertheless, they share the common idea of representing the related domain in a defined way.

> "When we assume that a domain consists of objects, relationships, and concepts we commit ourselves to a specific way of viewing domains. [. . .] this commitment to viewing domains in a particular way is called the conceptual model." [189, p. 11]

The high abstraction is not an issue when working in the conceptual modeling area. According to Guizzardi [105], conceptual models are intended

> "[. . .] to support understanding (learning), problem-solving, and communication, among stakeholders about a given subject domain." [105, p. 26]

Dependent on their intended degree of abstraction and content, this allows conceptual models a wide spectrum in their specification. Ultimately, this also serves the purpose since conceptual models should be understood and interpreted by people and, therefore, do not necessarily have to be based on technically sound definitions. This purpose is already implied by the name of the model type (conceptual model), where *concept* can be used with the following definition from *Lexico Oxford Dictionary*:

Definition 3.2 Concept "An idea or mental image which corresponds to some distinct entity or class of entities, or to its essential features, or determines the application of a term (especially a predicate), and thus plays a part in the use of reason or language." [`https://www.lexico.com/definition/concept`, acc. 03.2020]

Despite their inaccuracy, conceptual models are expected to follow certain standards and definitions. Conceptual models (in order to be called models) should adhere to be

> "[. . .] a repository of highlevel conceptual constructs and knowledge specified in a variety of communicative forms [. . .]" [17, p. 961]

Additionally, according to their purpose, they should serve as

> "[. . .] a purposeful abstraction and simplification of reality, capturing constraints and assumptions resulting in a conceptualization of the problem to be solved, the environment in which it has to be solved, and relevant actors, their behavior, and the relationships of interest to solve the problem." [262, p. 298]

The abstract nature of conceptual models leads to a large variety in their expressivity, layout, structure, and granularity. This variety ranges from very abstract and generic conceptual models (e.g., [72]) up to rather technical and consistently specified conceptual models, often based on UML or other OO-based modeling methods (e.g., [1, 170]). Although both representation sides of a conceptual model are different, both serve the intention of conceptual models as stated above. Additionally, an ontology can serve as a foundational and formal basis of conceptual models (e.g., [107]) that describes the domain's concepts of the conceptual model in a structured way.

Despite their abstraction, conceptual models can be specified in their own way by formulation model statements based on the provided model's concepts. This is illustrated by Guizzardi [105] in Figure 3.1. The conceptualization is representing the concepts that are provided by the model's domain, while the modeling language (often called specification language, too) is able to specify a concrete conceptual model that represents the domain. For the practical use and application of conceptual models, only the model and its specification in the bottom part of Figure 3.1 are used.

The application areas of conceptual models are manifold [76]. Especially for the information systems discipline, the rather non-philosophical requirement is applied that a conceptual model should provide its conceptualization based on a formal specification [210, 257]. The subject of conceptual models is a prominent issue in information systems as the conceptual models, as well as the information systems research, has their focus on the interaction of software systems and humans. Thus, the research field regarding conceptual models and their application in information systems is huge (e.g., [70, 117, 188, 189, 243, 265]).

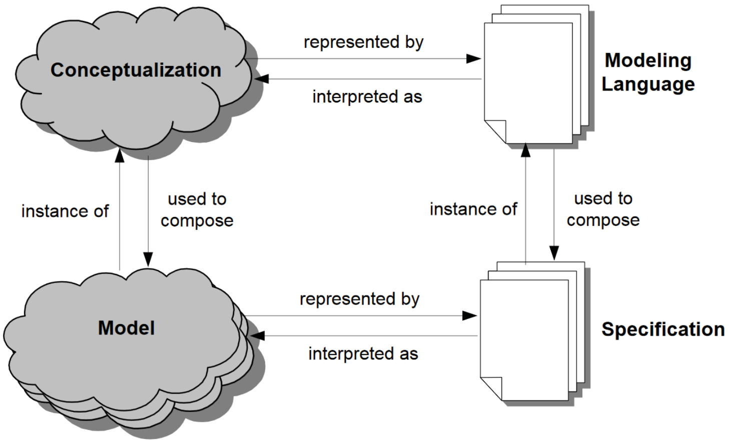


Fig. 3.1 Guizzardi [105]: conceptualization and model specification

While the nature of conceptual levels can differ widely, a general partitioning of the conceptual model's usage can be made:

- conceptual models that can be used either preliminary for software development (only responsible for communication, understanding, requirements, and acceptance tests), or
- conceptual models as an integrated part of the development process (thus, acting as guidelines).

The two basic natures of conceptual models (as one of their main properties) is also considered by Robinson et al. [210], who propose two major qualities that must be addressed by any conceptual model:

1. "The conceptual model must be sufficiently transparent so that all stakeholders in the project are comfortable in using it as a means for discussing [. . .]" [210, p. 2814]
2. "The conceptual model must be sufficiently comprehensive so that it can serve as a specification for developing a computer program [. . .]. One of the important purposes of the conceptual model is to provide the prerequisite guidance for the software development task." [210, p. 2814]

hereas the first property is of generic validity according to conceptual models, the second property is, nevertheless, not always established and provided by a conceptual model. If a conceptual model is intended to guide the implementation, a more fine-grained, standardized, and implementation-related model has to be used. However, both properties are not necessarily mutually exclusive. Furthermore, the life cycle of a conceptual model can start with the first nature (to be informative) and then evolve to the second nature (the be technically specific) afterward.

> "Once a sufficient level of understanding and agreement about a domain is accomplished [via the conceptual model], then the conceptual specification is used as a blueprint for the subsequent phases of a system's development process." [105, p. 26]

This, however, depends on the type of conceptual model and whether it allows a template functionality. More fine-grained and detailed models are in advantage concerning their usage as blueprints. For this purpose, domain models, as a subclass of conceptual models, are often in charge of solving this issue.

3.1.2.2 Domain Models

To grasp the full potential of an (individual) enterprise within a software product, one utilizes so-called *domain models* that capture domain knowledge (provided by, e.g., a conceptual model) in a structured and technical way. Domain models cover, as the name implies, the business domain in a specific structure or behavior. The domain, an essential influence factor when designing software systems, is used as the main background: every concept, structure, and behavior must be consistent with respect to the related domain. This thesis uses the domain definition from Reinhartz-Berger and Sturm [207]:

Definition 3.3 Domain "An area of knowledge that uses common concepts for describing requirements, problems, capabilities, and solutions." [207, p. 1275]

The business landscape (the domain) described by a domain model can be used for several purposes, e.g., simple requirement overview, feature selection, and development, or software architecture purposes. The term domain model is typically used as a collective name for more technical representation of a domain's structure or behavior (which can, however, be represented in many different ways, e.g., drawings, text, source code [125]). Thus, domain models, as a subclass of conceptual models, are often derived from more generic conceptual models. In this thesis, the ISO definition of a domain model is used:

Definition 3.4 Domain Model "A product of domain analysis that provides a representation of the requirements of the domain." [133, 146]

Usually, the domain model is modeled during the process called *domain analysis*, which identifies a domain's concepts and captures its ontology [258]. It gathers all required concepts needed to represent and explain the whole business landscape regarding both its structure and behavior (business processes). For example, a car rental enterprise would probably use the `Car` and `Client` objects in its domain model. As a result of a domain model analysis, the domain model consists of invariant entities that are ubiquitous in the subject. Among different domain analysis approaches, e.g., the *Application Domain Modeling* (ADOM) method is rather known and established [205, 207, 240].

This invariant property of expressed objects is then used as a starting point for a multitude of different applications as every individual application needs to deal with such objects. This reuse aspect is also highlighted by Valerio et al. [258]:

> "Domain analysis processes existing software application [and the application landscape in general, ed. Note] in order to extract and packet reusable assets.." [258, p. 5]

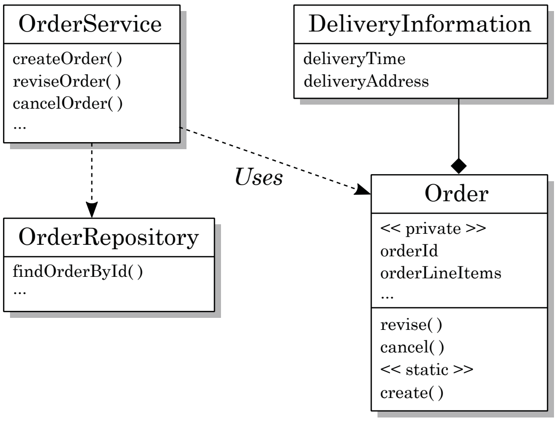


Fig. 3.2 Richardson [208]: an example (partial) domain model for delivery service (figure adapted from original)

A more technical focus is used by the domain model creation pattern of Richardson [208], who describes the domain model as a pattern for an microservice architecture:

> "Organize the business logic as an object model consisting of classes that have state and behavior." [208, p. 150]

The importance of a working domain model for an appropriate development process is tremendous regarding both a short time-to-market and a functional and efficient software system. Both aspects will likely have an immense impact on the positive business value of the software system [90].

In general, the paradigm of *Domain-driven Design* (DDD) [75] makes heavy use of domain models as core artifacts for development purposes. Everything that is in any way connected with the domain is oriented on the established domain model. Especially in the microservice domain, the domain model design is often considered to be a crucial part (e.g., [66, 124, 199]) since the loosely coupled concepts and parts of microservice systems are often separated regarding their domain part. Therefore, a domain model orchestrates the individual service parts of the system. Domain-driven design is, thus, an opportunity for successful construction. This is also done in Richardson [208], where an example delivery service is constructed with the help of a domain model via a microservice architecture. The domain model was built in OO-based UML style and is (partly) represented in Figure 3.2. It contains entities (`Order`), sub-entities (`DeliveryInformation`), service entities (`OrderService`), and data items (`OrderRepository`).

In general, a domain model is often based on the OO paradigm as it provides a natural understanding of the domain's concepts. The domain model consists of entities and relationships that are based on the formerly analyzed invariant objects of the domain.

> "In an object-oriented design, the business logic consists of an object model, a network of relatively small classes. These classes typically correspond directly to concepts from the problem domain. In such a design some classes have only either state or behavior, but many contain both, which is the hallmark of a well-designed class." [208, p. 150]

With this, there is already the term "class" used, as in onward software development progress, the domain model objects are used as central elements, as classes, for the internal software structure[1]. The object-oriented domain model design has several advantages [208]:

- the design can be easily understood and maintained, as there is no single object that does all the work but smaller and individual parts (objects/classes),
- the design is easier to test as concrete functionality only concerns a specific part of the domain and each part can be tested independently, and
- extension of the design is simpler since the relationships are clearly specified, and a new feature can be implemented without considering unrelated concepts.

Nevertheless, a domain model can not always be easily transformed into a class diagram. While a UML class diagram would mainly consist of classes and relationships, the domain model has completely different kinds of objects, e.g., entities, value objects, factories, repositories, and services [208]. Defined and thorough software design decisions are needed in order to implement these objects into technical classes. For such a specification of the domain model, *domain specific languages* (DSL) are often in charge. A DSL is specialized for the domain and able to represent domain-specific model statements in an explicit (and easy) way[2] in order to increase productivity, quality, validation and verification, communication tolling, efficiency, and other properties regarding the software development within the chosen domain [268]. Like domain models, the structure of a DSL can vary a lot regarding its nature (structural or behavioral), used entities and relationships, and the information content (some examples of different DSLs are given in, e.g., [40, 41, 88, 137, 142]).

> "DSLs allow domain experts to specify their knowledge in familiar terminology and representations such as textual, graphical or mathematical notations." [125, p. 83]

Their relationship and strong bond towards the domain-specific concepts distinguish DSLs from *general purpose languages* (GPL) that are domain-independent but still able to model a domain in a more abstract manner[3].

> "DSLs sacrifice some of the flexibility to express *any* program in favor of productivity and conciseness of *relevant* programs in a particular domain." [268, p. 30]

Thus, in contrast, a GPL is considered to be the opposite of a DSL. Therefore, a GPL is being used for a broader spectrum of possibilities but less domain-specific. Often

[1] Nevertheless, this is not a one-to-one mapping but the domain object may consist of several (technical) classes if necessary (e.g., the domain's *Order* may be mapped as `Order` and `OrderItem` in combination)

[2] DSLs are, in fact, specialized versions of the general "modeling language" in Figure 3.1.

[3] In the case of this thesis, the BROS language is regarded as a GPL as the role concept is able to model domains but does not make use of any domain-dependent concepts.

the GPLs (e.g., UML or BPMN) are used as a basis for DSLs. For further information about DSLs and their development, Voelter [268] is recommended.

The advantage of domain models is their relationship with all perspectives of related stakeholders (customer's requirements, business engineer's processes, architect's structure). A domain model acts as an interdisciplinary form of communication while being precise enough to fulfill technical purposes. Additionally, a domain model is, as the name implies, specified for the chosen domain and can express an enterprise's individual model statements. Thus, a domain model in healthcare is usually completely different from, e.g., an IoT or Industry 4.0 environment. Dependent on the used modeling paradigms (and their degree of granularity and detail), the domain model can be used as a guideline for further (technical) software development.

> "Those [domain] models make domain knowledge explicit, which is on the one hand relevant for a certain functional or architectural concern of the application under study, and on the other hand is important to be represented during a particular development phase." [125, p. 84]

Nevertheless, the research field of domain models and their usage is vast and established for many years (e.g., [4, 114, 135, 171, 204, 285]) such that it can hardly be covered by this thesis.

Due to their foundation on experts' knowledge and an OO-based structure, domain models are often used as generic templates for the domain-specific software development use case. They provide the necessary (and often non-technical) details on a more general and abstract level. Thus, they are a central tool for the early stage of software development since they may represent design decisions regarding required functionality derived from the business requirements. To meet the enterprise's specific requirements, the domain model template gets adapted and tailored. A domain model that is used in such a way and serves as a template (to be adapted afterward) is considered to be a *reference model*[4]. Reference models are introduced in Section 3.2.

3.1.3 Modeling of Structure and Behavior

Models are used to represent the structure and behavior of an intended software system in a specific way, describing its properties and functionalities (cf. Section 3.1.1). Domain models, as introduced in the previous section 3.1.2, can be used to fulfill that task in an abstract and generic way. Domain models themselves can be of different kinds. They can range from specific source code schema to abstract boxes with lines. Contrary to their representation, a domain model (and most of the other model types as well) can be categorized as structural models (e.g., UML class diagrams) or behavioral models (e.g., BPMN process models). There also exist model types in-between that cover the behavioral and structural nature of a software system (e.g., UML sequence diagrams), however, they are rather rare[5].

[4] This term is rather generic; in this thesis, a reference model is considered to be a subclass of domain models that are used as templates for adaptation.

[5] And, even then, they can be divided into their structural and behavioral part [189].

3.1.3.1 Structural Models

Structural models (and the respective languages) are commonly used to represent the technical details for later implementation (see section 3.1.1). A structural model usually consists of entities and relationships that describe the inner composition of the system and lead to an idea of how the system should look like. The used entities and relationships can be regarded at design time (type) as well as runtime (instance). Further, all elements of the model adhere to certain concepts that they are (or should) represent.

> "The state of a particular domain [. . .] consists of a set of objects, a set of relationships, and a set of concepts into which these objects and relationships are classified." [189, p. 10]

For example, the concept of a *Customer* can be implemented as the design time entity type `Client` with its runtime instance `Alice`. The concept term is directly associated with the concept definition used by conceptual models (cf. Section 3.1.2), therefore, making structural models a foundational item of conceptual modeling as they are able to represent the domain with an ontology of entities and relationships.

The use of structural diagrams highlights and specifies the composition of the necessary components of the targeted software system. They provide a static view of the system as a baseline for later implementation. The structure described by a structural model is separating the whole domain in its sub-parts, that is, sub-domains, and therefore contributes as a possible tool for separation of concerns and distributed (but fixed) management of concerns. For example, a car rental domain, consisting of several concepts, can be structured by a structural model into different entities and associations (that is, objects and relationships), e.g., the `Driver` is `driving` the `Car`. Thus, every stakeholder involved in construction, management, maintenance, and usage of the system is able to comprehend this correlation between the three concepts *Car*, *Driver*, and *Drive* (under the assumption that the structural model represents this statement accordingly, like shown in Figure 3.3). As a consequence, the structural model represents the dimensions and aspects of a system as a collaboration of concepts within dedicated domains and sub-domains.

There are different kinds of structural models available, dependent on the domain and purpose of the representation content. E.g., the OMG defined a set of different kinds of structural models (e.g., class diagrams, component diagrams, or object diagrams) for different layers of abstraction. As stated earlier, structural models often represent objects and their relationships in combination. This is usually done in a graph-based layout consisting of nodes and edges (in Figure 3.3, the nodes are the objects, and the edge is the relationship). This property is independent of the structural model type. Nevertheless, the meaning of a node and an edge in such a graph can vary widely. The interpretation of a graph-based structural model heavily depends on the context and the model type.

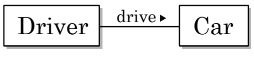


Fig. 3.3 Structural model example

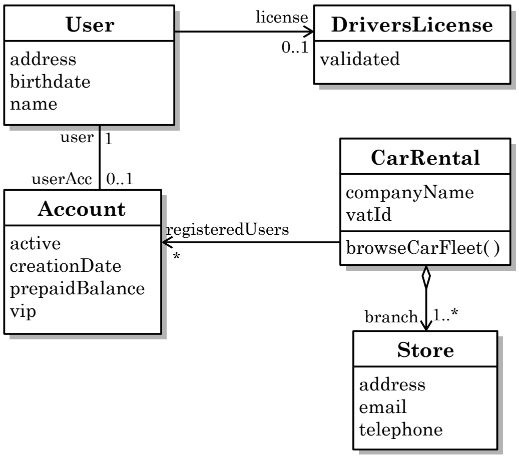


Fig. 3.4 UML class diagram example

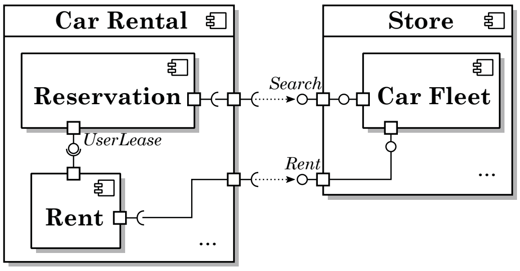


Fig. 3.5 UML component diagram example

Regarding the different structural model types, the most widespread structural model type is the *class diagram* of the OMG's UML model repertoire [11, 83, 187]. UML class diagrams specify the structure as an object-oriented composition of entities and their relationships in a node-edge-graph. Thus, one can express the foundational structure for implementation. An example of a small but fine-grained UML class diagram is illustrated in Figure 3.4. It consists of several entities (the classes, e.g., `User` and `Account`) that are connected via relationships (e.g., an association between them both). The classes are representing objects and, therefore, concepts of the domain (e.g., the `DriversLicense` object represents the *Drivers License* concept in the car rental domain). The objects (i.e., classes) are refined by their state (that is, fields like `address`) and behavior (that is, operations like `browseCarFleet()`)[6]. This UML class diagram could be used (with some further refinement) for implementation. The specification via UML class diagrams can be a subsequent step of a domain model. While the domain model represents the main business objects in their overall relationships, the class diagram specifies them further in a form that can be implemented (this sometimes sacrifices readability and understandability of the model's domain).

[6] Please note that this model is only an example and does not represent a full-fledged and detailed UML model.

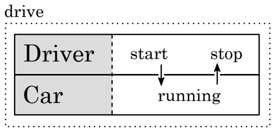


Fig. 3.6 Process model as example for behavioral models

Aside from UML class diagrams, component diagrams are very popular among system designers, too. They provide a more general perspective on the structure. An example of a car rental component diagram is illustrated in Figure 3.5. Instead of specification of single objects, the component diagram is used to represent the packages and containers (as nodes) that are dedicated to certain tasks (e.g., `Car Fleet`) as well as the detailed specification of their interfaces (e.g., `Search`). The edges in this graph-based structural diagram are then the used and required interface relationships. Other structural models and diagrams are used in practice as well, dependent on the used domain and the structure that needs to be modeled (e.g., package diagrams, object diagrams, or simple node-edge-graphs).

A complementary diagram category to structural models is behavioral models that state these concepts that are not able to be represented by structural models: the dynamics of processes and states, and how model elements interact on each other.

3.1.3.2 Behavioral Models

Behavioral models are, in contrast, focusing on the behavior of the system components (often called business process models, too). Instead of stating the components in a graph-based and object-oriented manner with nodes and edges, the behavioral models do make use of a flow-based or chain-based diagram style. The behavioral modeling is required to fulfill the customer's requirements as they are often stated as activities (e.g., "A user can make a reservation of a specific car.") than structural constructs. Due to their nature, structural models have the drawback that they can not represent the time and process flow natively. Although some works introduce behavior into structural models (e.g., [73, 174, 191, 232]), it is often not sufficient as a resonable behavioral model. However, to model the structure and behavior separately from each other is not necessarily a decrease in quality, but they have to be modeled accordingly to be not inconsistent. Especially the process models (i.e., behavioral models that represent concrete business processes of the enterprise) should be adhered to by software structure, as business processes are the key to handling complex software systems. In this thesis, a business process is defined as follows:

Definition 3.5 Business Process "A set of one or more linked procedures or activities executed following a predefined order which collectively realize a business objective or policy goal, normally within the context of an organizational structure defining functional roles or relationships.“ [48, p. 126]

Behavioral models are intended to model the dynamic view of a system, which is temporal states and transitions (e.g., as shown in Figure 3.6). Behavioral models can be specified for a multitude of purposes, e.g., requirements, internal exception flows, interaction of microservices, temporal accessibility of components, communication of participants, or state changes of a processed product. Complex behavior can indeed have impacts on the structure of a software system, however, often implicitly. E.g., a use case described by a behavioral model states the implications of the participating entities and the kind of relationships they have.

> "Conceptual modelling of business processes is deployed on a large scale to facilitate the development of software that supports the business processes [. . .]." [2, p 129]

The behavior is surrounding the software development stack: it is placed in the early and late phase of software development. The former to define the specifications needed, and the latter to test and qualify the resulting software system. In-between, behavioral models serve as a reference for the implementation of functionality. In such a case, the constructed structural models are the first reference (e.g., a class `Car`, its fields, and operations are needed to be implemented). The behavior side, in contrast, leads the "how" the respective structure should be working (e.g., how the operations of `Car` and `Customer` should executed). In general, a behavioral model can be regarded as a conceptual model as well since it matches its definition (formal description of an aspect of the system for the purpose and understanding and communication of the semantics). Although not communicating how the system should be structured and built, they communicate how the system should act and react between all related stakeholders [26]. In the information systems discipline, this is an important fact, as the related (enterprise) systems are human-centered. Behavioral models are considered to be *action systems*, stating the process part of the information system that describes all the needed and defined business processes. Thus, an information system consists of both the software system and the action system, and both lead the functionality of the respective complete system.

> "The realisation of efficient business information systems recommends the joint analysis and design of the software system and the corresponding action system." [87, p 942]

It is, however, in a software project not necessary to stick to a single model that describes every business process in a software system, and many different models may be necessary [282]. Large software projects can involve up to several hundred process models [26]. Also, different behavioral model types can be used within a single project, e.g., a business process can be modeled for business developers and for software developers in different ways, dependent on their view and requirements [281]. Neither is it required to model every behavior in such a software system with a behavioral model (e.g., but only the important business processes).

Like all other kinds of conceptual models, behavioral models can vary widely in their expressiveness, appearance, formalization, and functionality. A behavioral model can be of a very abstract and generic nature, which makes it applicable as an overview that is easy to understand for external stakeholders. In contrast, a behavioral model can be very specific, too, e.g., by describing every detail of a business process and

Fig. 3.7 BPMN diagram example

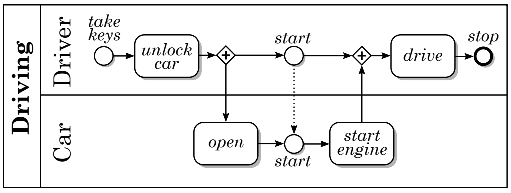
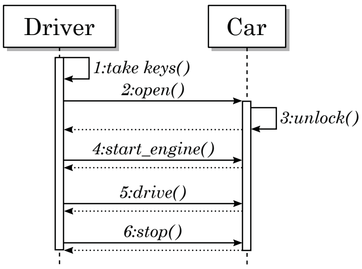


Fig. 3.8 UML sequence diagram example



what happens at any time during its execution (including, e.g., roles, responsibilities, resources) [95]. Behavioral models can be derived from natural language that is stated in requirements (e.g., "The customer can rent a specific car."). A more abstract and generic behavioral model would probably model this as a series of activities (the customer's activity "rent" towards a car). An explicit and fine-grained behavioral model would use more details and expressions to state the behavior (e.g., specific start and endpoints in time, different privileges, different views for a customer as a person and the car as a resource). The level of abstraction of a behavioral model has a direct effect on its formalization: a behavioral model that keeps general information is easier to understand but hardly able to be formalized and validated. In contrast, a detailed and full-fledged behavioral model can make use of formalized expressions and is able to be checked formally (e.g., [157, 180, 272]). Verified and validated behavioral models are, e.g., required for critical software systems in domains like aircraft, medicine, or banking. The probably most popular behavioral modeling language BPMN (*Business Process and Model Notation*) [184] is in-between both sides. It can model processes in much detail, however, its inaccuracy and ambiguous elements are implied by a lack in its formalization [281].

Dependent on its degree of abstraction, a behavioral model is usually a chain-based flow of activities that capture the related business process:

> "For software development process typically diagrammatic notation is required, for capturing a legible and understandable view of the business process." [2, p 131]

The aim of the behavioral model should be a description of the business process as an interaction of participants with each other and resources. Figure 3.6 states a generic example of a behavioral model as an interaction of two participating entities and their respective states. This thesis can not cover the full BPMN documentation; however, a comprehensive guideline of the BPMN standard can also be found in Allweyer [7]. Most behavioral models include participants that interact with each other while a certain time passes. Therefore, time often plays an important role in behavioral models. E.g., in BPMN, the passing of time is represented by chaining the activities from left to right. It is also chain-based and connects participants via activities and temporal events during their lifetime. For exact time values, BPMN uses the concept of events to mark specific points in time that affect other model elements. An example of a simple BPMN process is given in Figure 3.7. A UML sequence diagram, for example, has a top-bottom approach, and the passing of time is connected to the vertical lifelines of objects (which, again, interact with each other). Figure 3.8 states the example from Figure 3.7 as a possible sequence diagram. There are multiple other possible behavioral models (and languages) that can be used in practice, dependent on the context. *Event Process Chains* are often used to describe a process flow via chained event occurrences (e.g., [176]). *Business Process Execution Language* (BPEL) [191] is recommended for development purposes due to its technical alignment. Further, other UML models can be used as well, like state machines (chaining the possible system's states and transitions), or use case diagrams (assigning tasks to participants).

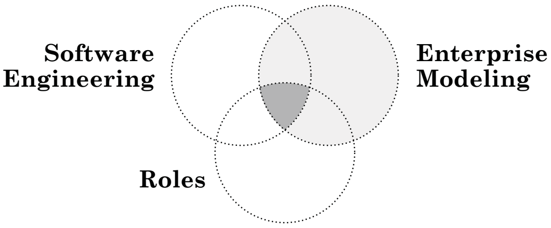
In this thesis, there is a distinction between a *behavioral model* and a *process model*. Although not consistently used in literature, it is assumed that a process model is a subclass instance of behavioral models. While behavioral models are a class of models that states every kind of activity and behavior possible, a process model is focusing on specified business processes stated by business developers (that is, a chain of specified actions and events to solve a problem). Thus, behavioral models[7] are more generic and cover more abstract activities.

Finally, the research on business processes, their models, and languages is a vast subject that is evolving continuously. Many different works cover the different parts of behavioral models (i.e., business process models) and their impact on software development and enterprise modeling in particular (e.g. [14, 26, 131, 211, 201, 217, 274]).

3.2 Enterprise Reference Modeling

This section provides an overview of the topic of reference models. Reference models are template models that are modeled by domain experts and used by modelers to adapt them towards specific application models in order to include individual requirements in an efficient and effective way [126]. The resulting individualization via the variability of reference models leads to a huge benefit and chances for enterprises [250]. For that, structures and processes stated in the template reference, which overcomes common domain problems and providing standards, can be reused and adapted for enterprise-specific variants.

[7] Nevertheless, this thesis makes no essential distinction between *behavior* models and *behavioral* models.



3.2.1 Nature of Reference Models

Reference models are, as implied by their name, proposing a reference for a specific domain. Via adaptation, application models are derived from the reference models. Although it is commonly agreed that a reference model should act as a template or blueprint, the term reference model is a very generic term. It only describes its template functionality of the related model. The related model can be, in fact, of any shape and type, as long as it can be used to guide the modeling as a template. In terms of their nature, reference models are a subclass of domain models (which are, in turn, conceptual models) [280] as they provide descriptive guidance [13] for a certain domain as a structural or behavioral model. Thus, in this thesis, a reference model is rather considered to be a "domain reference model." In contrast, not every conceptual or domain model is a suitable reference model [78]. In theory, every domain model can be used as a reference to derive the application model. Nevertheless, a dedicated domain reference model is modeled with this intention and, often, proposes a set of adaptation points for individualization.

> "A reference model is a description [. . .] a set of conceptual entities, and their relationships, plus a set of rules that govern their interactions. At the highest level of abstraction, a reference mode1 is a standard. It prescribes the highest level of architectural structure contained in the model." [13, p. 98]

Reference models provide, like patterns and standards, a generic solution for common problems in terms of model constructs. The term reference model is often used but often neither understood nor used in the same way [255]. In this thesis, the reference model definition of Mišic and Zhao [178] is used:

Definition 3.6 Reference Model "Describes a standard decomposition of a known problem domain into a collection of interrelated parts, or components, that cooperatively solve the problem." [178, p. 484]

Software system construction can make use of reference models and adapt them to the enterprise's individual requirements [280]. The adaptation of reference models is the major process intended and implied by using a reference model [78]. As reference models are often constructed by domain experts, the possibility of modeling errors is reduced, and chances to develop more efficient models is increased. As written in Section 3.1.1, models used for software development should have the right accuracy and communication (that is, model quality regarding the real-world concepts). To support that, reference models support the software development process with established (and often proven) solutions for a wide range of issues, and to aid modeling of specific requirements [126, 178]. According to Fettke and Loos [78], reference models provide three major benefits:

> "However, different distinguishing features are discussed in the literature:
>
> 1. *Best practices*: A reference model provides best practices for conducting business.
> 2. *Universal applicability*: A reference model does not represent a particular enterprise, but a class of domains. Hence, a reference model is valid for a class of domains.
> 3. *Reusability*: Reference models can be understood as blueprints for information systems development. Thus a reference model is a conceptual framework that could be reused in a multitude of information system projects." [78, p. 2]

The three priorities of best practices, universal applicability, and reusability are fundamental paradigms of a reference model [126, 280]. These properties do not limit the type, layout, granularity, or shape explicitly. It is the intention and purpose that makes a conceptual model (a domain model in the best case) a reference model. Especially the universality of a reference model is often highlighted, e.g., by Thomas [255]:

> "The fact however, that the universality of a reference model can not be understood in the sense of the model's claim to absoluteness, i.e. a claim to universal validity, often goes unrecognized. A reference model can only be (universally) valid with regard to a certain category of applications, for example a category of enterprises or a category of projects."[255, p. 488]

Thus, although universally applicable (and providing best practices as well as reusability), reference models are always dependent on a certain domain, often even more specific sub-domains or problem areas. But within their domain, it is expected that a reference model proposes a stable and reliable modeling baseline for application models [103].

Further, concluding the best practices and reusability enables a recommendation character for reference models [255], providing a standard (solution) for a specific issue. However, the models claimed to be reference models require the modelers' and developers' acceptance. This is valid for both sides: the set of declared reference models (as a subset of information models) is overlapping with the set of accepted reference models that the users actually adhere to. This is illustrated by Thomas [255] in Figure 3.9: the overlapping part is the ideal spot for a developed reference model:

$$RM_{Declaration} \cap RM_{Acceptance}$$

Further, this schema identifies another type of reference model: the used but not declared reference models. These models are not intended to be reference models, however, they provide a useful universality and recommendation (best practices and reusability) characteristic [255]:

$$\overline{RM_{Declaration}} \cap RM_{Acceptance}$$

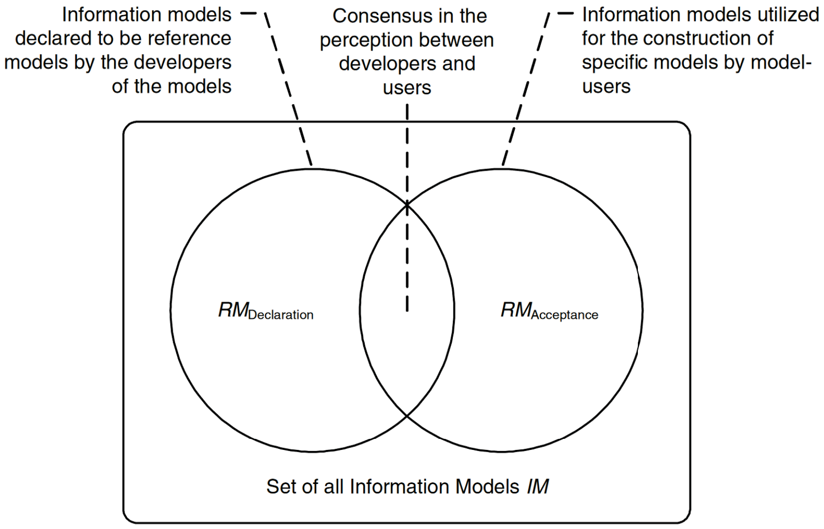

Information models declared to be reference models by the developers of the models
Consensus in the perception between developers and users
Information models utilized for the construction of specific models by model-users
$RM_{Declaration}$
$RM_{Acceptance}$
Set of all Information Models *IM*

Fig. 3.9 Thomas [255]: the set of declared reference models and accepted reference models

The third case (declared reference models that are not going to be accepted and used) is the worst, as it implies a failure in the model's recommendation or universality characteristic, and needs to be avoided.

The use of a reference model enables the modeling of individual enterprise requirements based on proved solutions. Nevertheless, the reference model itself can be provided in two fundamental ways:

1. as a *public* reference model, provided by, e.g., a standardization organization or expert group in order to unify the domain across different enterprises, or
2. as an *in-house* reference model, which is only accessible for the enterprise's internal users and state the enterprise-specific best practices to common problems [103] to unify the enterprise's business and software landscape.

Both approaches are valid and used in the field of enterprise modeling (or software systems in general). However, both ways come with different implications. While the public model is often proven more in-depth (due to the knowledge contribution of the public), it is also out of the hand of the enterprise. An owner is then an external person (or committee), and individual requirements and changes of the reference model are more complicated. In contrast, an in-house reference model is more convenient to change and individualize to fit the enterprise's goals but could suffer from lower standardization across the enterprise's boundaries. Thus, due to possible over-engineering, smaller enterprises tend to use their in-house reference model, while larger enterprises probably get more benefits from using and adapting the public reference model (or even both types simultaneously) for their internal software systems (e.g., to comply international obligations).

Reference models exist for several areas and are often based on industry-related subjects or management functions (business models) [279]. Like domain models, the reference models' natures can represent these areas in both types: structural or behavioral. Dependent on what the reference model is built on, the adaptation process is used to refine the template reference model into enterprise-specific application models: either structural or behavioral application models.

> "Although the discussion of reference models mostly is focused on process models, reference models can cover other fields like data structures, functions, even metamodels [. . .]." [280, p. 1561]

Reference modeling is a core subject of the information systems and enterprise modeling discipline [79]. The strong focus of the information systems discipline on business processes leads to a higher prioritization of behavioral reference models (that allow the customization of template processes) than structural reference models (which allow the customization of structure). Behavioral reference models and their application are therefore more widespread and are frequently used in research and application of enterprise models (e.g., [24, 93, 98, 218, 236]). For example, a banking transaction (behavioral) reference model would provide a set of processes that describe the general, efficient, and secure execution of any banking transaction process. Due to the individual requirements of each bank, the transaction reference model can be adapted towards the banks' individual requirements. The behavioral application model is derived via adaptation of the processes to fit a bank's own needs. The same use case can be done regarding structural reference models, where a bank can use a general reference model to derive its own structural model via adaptation. The only difference is that this version works on structures (that is, e.g., classes, components, data, actors) instead of processes (see Section 3.1.3) and provides information about a suitable banking transaction software design. Thus, structural reference models are suitable for implementation purposes, dependent on their degree of granularity and abstraction (e.g., [3, 166, 198, 219, 227]). Nevertheless, due to the completely different nature of behavioral and structural models, the adaptation process of the respective reference models (to achieve the enterprise's application model) is very different as well (see Section 3.2.2). A comprehensive study on reference model classification is given in Fettke et al. [79].

An important facet of a reference model is the utilized modeling language. A reference model is not determined by its used language and can cover any language [279] as long as it can be used as a template. Nevertheless, typical model types (and their language) used by behavioral reference models are *Event-driven Process Chains* (EPC), *Entity Relationship Models* (ERM), BPMN, and other UML behavior diagrams [79]. For structural reference models, the UML class diagram is often in place [80]. Other possible variants are component diagrams or less systematic models (and languages) like domain models. The huge variety in the different types of reference models allows a very diverse landscape of reference models, and there is hardly a way to point at the one right choice of model type and language.

> "Although a number of reference models for different types of systems have been (and are currently being) proposed by individual companies and organizations and industrial consortia, they offer vastly different viewpoints and feature sets." [178, p. 485]

3.2.2 Adaptation

Enterprises are often in need of individual solutions regarding their software system. A reference model can help them build their individual software system according to an established solution. However, a reference model as a template is only of use when there is a sufficient approach of how to adapt it towards an enterprise-specific application model. Via adaptation, the reference model is changed to fit specific needs. For example, a reference model for a car rental enterprise can be adapted towards a caravan and camper rental enterprise by adapting several business objects (e.g., `Car` to `Camper` and `Caravan`). With that, a reference model adaptation (i.e., the application model as a result) can be compared with a variant of the initial reference model that fits specific needs [135, 226]. The adaptation is the main process of interacting with the reference model. An ideal adaptation is based on the following properties:

- it results in an application model that meets the enterprise's requirements,
- it does not modify the underlying reference model, and
- it can be executed systematically.

There are many different ways to adapt a reference model based on the reference model type, abstraction, and purpose. Hofreiter et al. [126] classify the different adaptation approaches of reference models into five distinct classes (see Figure 3.10):

Configuration (by selection) Deriving a model by making choices from a greater variety of offered alternatives. The reference model domain is completely described, including all possible adaptations.

Instantiation (by embedding) Placing one or more original models into a framework with generic placeholders. The reference model domain is covered by a framework and uses placeholders for parts that can not be unified by a reference model (and need adaptation).

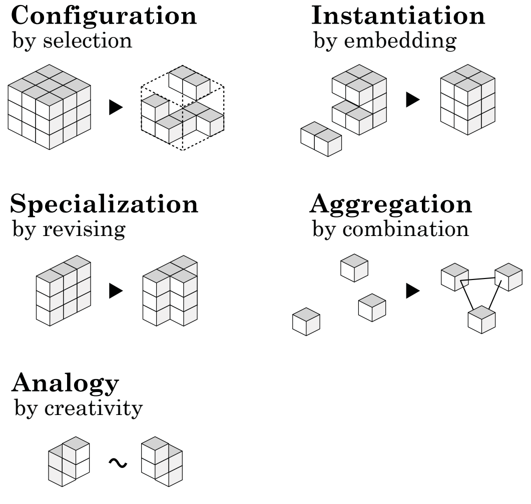


Fig. 3.10 Hofreiter et al. [126]: classification of adaptation approaches (figure adapted from original)

Specialization (by revising) All statements from the template are taken over and possibly more specified (but not deleted). The reference model domain is covered by a core solution that is intended to be extended.

Aggregation (by combination) Combination of several models parts to create an individual composition. The reference model domain is described partly, and the application model emerges from combining the relevant ones.

Analogy (by creativity) The template serves as a guideline and orientation for the individual model construction. The reference model domain is described as patterns that have to be replenished.

The different classes are independent of domain and language. Additionally, different adaptation classes can be used for different reference model parts (and different classes can be used for the same model part as well) [126]. A comprehensive overview and further detailed explanation regarding the classification is given in vom Brocke [269] and Hofreiter et al. [126]. Alternatively, Balko et al. [19] propose five different categories of (behavioral) adaptation method natures[8] in their handling of the reference model by the adaptation: from-scratch modeling, patching, blueprinting, configuration, and ad-hoc changing. While from-scratch modeling can hardly be regarded as reference modeling (as the reference is missing), configuration and blueprinting are comparable with configuration and specialization of Hofreiter et al. [126].

Despite the large establishment of the adaptation concept related to reference models, the actual adaptation process is often not in the focus of reference modeling research and application. Although the previously described classification is accepted in research (e.g., also used in [24, 240, 269]), actual approaches that describe and implement an explicit adaptation method are rare.

> “However, while much attention has been given to the content of these models, the actual process of reusing this knowledge has not been extensively addressed.” [205, p. 50].

Most reference models are considered to be of the configurable type [126]. The respective reference model is, according to the configuration category, proposed as a full-fledged domain model that covers every specific domain part. Adaptation is done via selection and can be compared (metaphorically) with carving a sculpture in arts: the provided subject can be tailored into individual solutions by removal of parts, however, additive features or corrections are hardly possible. Nevertheless, the configuration of reference models is an efficient adaptation approach as it is easily accessible and limits the possible choices the users face. A configurable reference model is detailed enough that it covers the whole domain in its boundary and, therefore, can be validated and verified by domain experts without the need to adapt it beforehand. Thus, the majority of research and work is done regarding reference models that are intended to be configured (e.g., [23, 161, 172, 212, 219, 225, 202, 254]). Information systems research (and enterprise modeling in general) is rather focusing on business processes, therefore, the configurable reference models are often behavioral ones.

[8] The authors intended to use these categories to explain different “flexibility techniques.” However, based on that, they argue on adaptation method natures of (behavior) reference models.

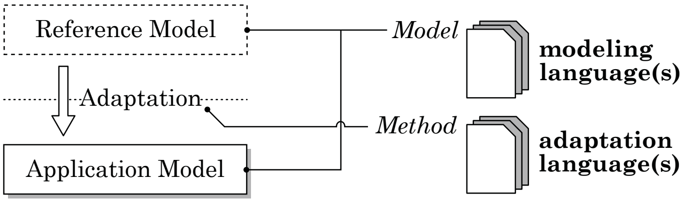


Fig. 3.11 The different parts of an adaptation

The used adaptation method is determined by the choice of the reference model type. Reference models of the configuration class are using adaptation methods that adapt the reference model by choosing between different and pre-defined options, e.g., different path executions, operators, or task assignments (e.g., using an EPC-based configurable reference model [71, 203, 206, 212]).

However, besides configuration, the model adaptation landscape is sparsely populated. Especially regarding structural reference models, which are at least necessary to implement the system, reliable and systematic adaptation approaches are rare. Often, an ad-hoc modification of the reference model (modeled as, e.g., a UML class diagram) is used as an enterprise-specific "solution."

> "In the simplest case, a reference model is duplicated manually without any technical assistance, which does not restrict its adaptation in any way. However, problematic redundancies and inconsistencies between the original and duplicated model can arise." [78, p. 2]

The ad-hoc modification comes along with several side effects. Although it is the easiest way and provides short-term benefits, crucial issues can emerge from this ad-hoc modification (low flexibility and safety, high redundancy, technical debts, and other unwanted software properties caused by non-managed architecture [90, 163, 221]). To avoid the mentioned problems and anticipate unnecessary redundancy and structural conflicts, the adaptation methods have to be more powerful and systematized [78]. Especially the independence of the application model of the reference model has to be fostered by the adaptation method. For example, Schütz and Schrefl [227] propose a systematic adaptation method approach for data warehouse items that make use of generalized operations to add, change, and remove data attributes and dimensions accordingly. Another example is the work of Gottschalk et al. [99]. Although based on behavioral reference models (in this case, process models in *YAWL*[9]), they propose an adaptation method (based on configuration) that uses input and output ports for process activities. By using different operators for ports (activate, block, hide), the overall reference model process flow can be composed individually by configuration.

Reference models use, for their adaptation purpose, a certain model type in a specific modeling language. Usually, conventional modeling languages [212], like, e.g., BPMN, EPC, or UML, are in charge and used for reference models. More

[9] `https://yawlfoundation.github.io`

specific reference modeling languages (also known as variant modeling languages) support the adaptation of reference models by built-in tailoring features.

> "Reference modeling languages must therefore be created so that they support model-variant management." [255, p. 493]

Examples are *C-YAWL* [99], *Configurable Event-driven Process Chains* (C-EPC) [154, 212], *Provop* [110], or *variant BPMN* (vBPMN) [69], which provide a standard set of configurable operators and mechanisms for the behavioral reference model and support (a) the reference model owner to build a standardized reference model, and (b) the user to adapt the model. A comparison can be found in, e.g., Döhring et al. [68]. For structural reference models, no such dedicated approaches are available.

Together with a reference model and the adaptation method, an adaptation language is required in order to describe the adaptation method implementation formally and systematically. This is often handled by a graphical and systematic notation. A language that is able to state the adaptation process can be used to increase traceability and serve as a basis for further validation of the resulting application model. However, little research is done in that area as often only the reference models (and their contents) are in focus, leaving a gap of adaptation methods and their languages:

> "[. . .] our analysis of the obtained results points to open research questions. For instance, the development of language constructs for reusing and customizing of model elements in the scope of the model's application or the evaluation of languages before designing a reference model." [79, p. 478]

The lack of adaptation languages can only be assumed in this thesis. A missing definition and concrete boundary lead to ambiguous statements, that is, misusing the term "adaptation language" for other subjects than a language for explicit and systematic adaptation of a reference model following an adaptation method. Further, there might be no high interest in an adaptation language as ad-hoc adaptation (simple and pragmatic changes modification the reference model) does not make use of such a language. Additionally, the use of such an adaptation language is limited, as it is only of use when tailoring the reference model towards an application model. After that, the application model can be used without any adaptation language. Besides, an adequate adaptation language requires an in-depth, described adaptation method. General classes (like, e.g., configuration) and other abstract methods are not sufficient enough to act as a foundational basis for an adaptation language. That said, this thesis does not state an explicit adaptation language either[10]. Although several statements of a related DSL are given in Appendix C.1, they can not be regarded as a complete adaptation language. Future research should focus on adaptation languages as well. Inspiration can be taken from current approaches in other domains, e.g., *delta programming* [52, 109, 216]. The adaptation (the "delta") is made explicit in source code functionality. This addresses the issue of changing an underlying model by adaptation statements. Further, Cazzola et al. [45] propose several functionalities to evolve an OO-based model with certain change operators.

[10] The thesis' BROS language is considered to be a *modeling language* to state models, not an *adaptation language* to state the adaptation process. Although the BROS *adaptation method* is developed explicitly, it does not come with a formal language, yet.

3.2.3 Evolution

The evolution of a software system is a subject of larger scales than reference modeling. Software system evolution occurs in many different situations, e.g., when the software needs to be changed towards new business requirements. The evolution of software is usually a subject of software architecture as it impacts crucial architecture areas like software business value, changeability, and dependability [90] and to avoid, e.g., technical debt, component landscape development, and non-functional requirements [163] simultaneously. However, reference model evolution is always of interest when the regarding software system architecture is based on such a reference model. Large software systems are complex, and the right architecture can reduce this complexity [163]. Using a reference model, as an architectural instrument, helps to deal with high complexity. Domain issues and patterns are provided in a reviewed manner and can be used for individual solutions. Thus, using a reference model is a fundamental design decision of the software architecture process and, therefore, needs to be tackled when the evolution of software structure is ahead.

It is a tremendous advantage of software engineering that, in comparison with other engineering disciplines, the structure and overall architecture of software artifacts can be adapted during the process and after finalization [250]. The need for change, modification, or adaptation as a source of evolution is diverse. Especially in long-term software systems, where the lifespan has no (defined) end but only parts being replaced, evolution occurs with the continuous flow of new requirements and functionality [181]. For example, the following situations might trigger a software evolution:

1. New business requirements
2. Changed business processes
3. Legal requirements
4. Outdated or updated dependencies

Regarding the use of reference models, the last point can be applied: the dependency on an (external) reference model. If it is assumed that a (reference model adapted) application model is implemented in an (information) software system, then the consequence is strict adherence to the evolutionary step of the reference model, or one would lose the benefit of using a reference model at all. So, the evolution of a reference model has to be handled with care. On the one hand, in an ideal case, the adaptation was done without any modification of the reference model and independently. Then, the reference model evolution can be executed in the application model as well, and changes in the adaptation are limited to a minimum. On the other hand, when the adaptation is strongly interfering with the reference model (or, the worst case, the reference model itself was modified), all changes need to be traced back, the adaptation withdrawn, the evolution step executed, and the adaptation applied again. This is both prone to errors and bad modeling style. Therefore, a maximum of independence regarding the adaptation is of an advantage when a reference model evolution occurs. Please note that the evolution still takes place at the reference model's owner side, and the user (only) has to follow the changes. The users never

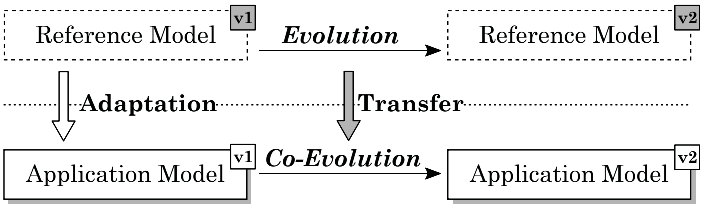


Fig. 3.12 The concept of reference model co-evolution

trigger a reference model evolution by themselves. Thus, the application model needs to be updated as the reference model evolves, which can be a difficult task. And reference models change in the course of their lifetime, just like any other model, which can make adaptation challenging. Therefore, the best modeling paradigm is to be as non-invasive as possible from the reference model (that is, no deletions and no modifications to the original template).

A concept that handles this double work for parallel executed evolution is often called *co-evolution*. The basic idea of co-evolution is to describe the evolution as a set of operators that are then transferred towards the co-evolution and executed there as well (see Figure 3.12). All changed data and structures are propagated towards the co-evolving associate [58]. This prevents a necessary re-adaptation of the whole new (updated v2) reference model again. This, however, requires a dedicated language to formalize these processes. Additionally, co-evolution is not restricted on models but can also be used to different domains (e.g., co-evolving a method and a related tool). Up to now, co-evolution is not in the focus of reference modeling and little covered yet (e.g., [58]). Still, aside from the information systems discipline and reference modeling, co-evolution has a long tradition in research. Several works consider co-evolution of models and metamodels (e.g., [51, 116, 122, 270]). The work of Balko et al. [19] introduces a "patch" mechanism on process reference models, which is similar to the co-evolution approach:

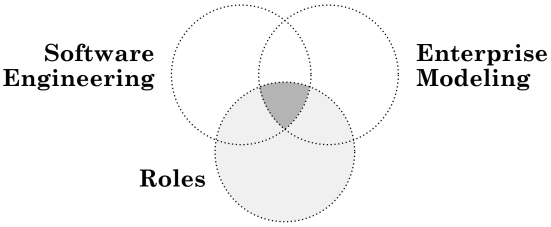
> "Reference processes may be patched (bug-fixed, updated) by the vendor only. Customers receive reference processes as read-only shipped content which is only updated as part of a software release." [19, p. 5]

Based on the experiences in the co-evolution research area, the co-evolution of a reference model and an application model can be of advantage as complex adjustments are thus prevented. This thesis does not cover the co-evolution subject as a core part[11]. Nevertheless, the fact of co-evolving an application model with a reference model influenced some design decisions and strengthened the argument to strive for a maximum independent and non-invasive modeling language and adaptation method.

[11] It is, instead, future work.

3.3 Role-based Modeling

Roles are the main part of the solution space to issues proposed in Section 2.2.3. Role-based modeling uses the role's characteristics to represent flexible and dynamic software structures. In this thesis, it is intended to utilize roles as a sole adaptation mechanism for reference models [220]. This section provides an overview of the role paradigm as well as current role-based technologies and concepts used for software engineering and enterprise modeling.



3.3.1 Nature of Roles

Roles are a fundamental concept of the natural language and are often intuitively used for the description of (software) systems and related models [248]. It is used to denote entities with a (temporal) structure or behavior to accomplish a certain task in a specific context. An often-used natural role example is the `Employee`, which is the role of the `Person` entity in the context of `Work`. The role concept is neither a new nor an underrepresented concept in literature. Research related to roles is established for several decades. Although there is a lot of research done in the role subject (see the following Sections 3.3.2 and 3.3.3), it is still ambiguous in its use for software construction and hardly represented in practice. From different definitions and ontological embedding towards plenty of ways of implementation, roles are a highly diverse subject.

3.3.1.1 Definition

There are several different definitions of a role available, depending on the area, use, and purpose of the defined role concept. Some prominent examples from the domain of the role concept are outlined here. Already in the seventies, Bachman and Daya [16] stated roles as identification of an entity. Roles are used to assigning new behavior to a static entity in data models (see Figure 3.13).

> "[The role] is a defined behavior pattern which may be assumed by entities of different kinds. Furthermore, a particular entity may concurrently play one or more roles. Hence, the existence of all the roles of interest for a given entity characterize that entity. [16, p. 465]

The role is a representative of the entity, encapsulation the entity's attributes (data), and handles its relationships. Thus, an entity can manage to have distinct "natures." In Frank [85], the role concept is used as a new concept identifier for delegation in modeling.

> "[A role] does not only dynamically "inherit" a role filler object's interface (as it would be with inheritance, too) but also represents the particular role filler's [that is, objects, ed. Note] properties. In other words: It allows for transparent access to the role filler's services *and* state." [85, p. 16]

With that, a role is regarded as a delegation mechanism and differs from inheritance (that is, generalization and specialization). This is particularly of importance when designing domain models as they often make use of the role concepts naturally. Another viewpoint of roles as a new type of interfaces towards entities is proposed by the work of Steimann [245] that describes a role as follows:

> "Roles are bridges between instantiable classifiers and their associations, be it in the scope of a collaboration or outside the context of any interaction. But roles are no association ends (or the names thereof) – a role (more precisely, a role type) is a classifier like a class, only that it does not have instances of its own. Instead, a role recruits its instances from those classifiers that are declared compatible with the role, that is, that comply with the partial specification that makes the role." [245, p. 200]

Roles are considered as a new element between an entity and its relationships. In turn, a relationship between two entities is not connected to the entities themselves but between the used roles of the entities. Riehle and Gross [209] state a generic definition of roles in order to establish a role model.

> "A role type describes the view one object holds of another object. A role type is a type, so it can be described using an appropriate type specification mechanism. [. . .] One also says that the object plays a role specified by a role type." [209, p. 118]

A role type (the design time variant of a runtime role) is used to annotate a relationship between two objects that play the respective roles in order to contribute to the relationship. Many other definitions of a role can be found in the literature (e.g., [144, 151, 209, 239, 248]). It depends on the domain of how the role is recognized and used as a modeling concept.

Due to the broad bandwidth of a role's spectrum, the definition of a role in this thesis is tailored for the thesis' approach, but based on the well-established works and viewpoints in the literature described above:

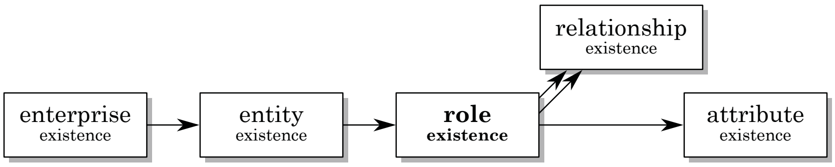


Fig. 3.13 Bachmann and Daya [16]: the role concept in structural domain (data) models (figure adapted from original)

Definition 3.7 Role A contextual modeling construct with state and behavior that is fulfilled by an object or its roles to represent it in the user's context and extend or change its corresponding specifications and interactions. However, a role does not modify identity: when either permanently or temporarily fulfilling a role instance by an object instance, the object instance's identity remains unchanged. (see [220])

The role has, according to this definition, a state and behavior, but no identity on its own. Instead, a role uses and represents a fulfilling object's identity. There can be multiple roles for an object, and each role can be fulfilled by multiple objects simultaneously (at design time[12]). This is, e.g., differently defined in other works (e.g., [16, 85]). Further, per definition, a role is assigned to a context in which the role acts with its structure and behavior as a participant [56] and representative of the fulfilling object.

3.3.1.2 Scope

The term of a role is already established on diverse modeling languages. E.g., in UML class diagrams, a role is used as a name for association ends [187, 245]. For BPMN, a role is used to state participants, although it has changed and is now called a swim lane [184]. In both cases, the role is reduced to be a label for individual viewpoints from one entity to another (e.g., one `Person` can perceive another `Person` as `Employee`). Several other diagrams make use of the role concept as well, e.g., use case diagrams or state charts [245]. Very often, the role concept is used for

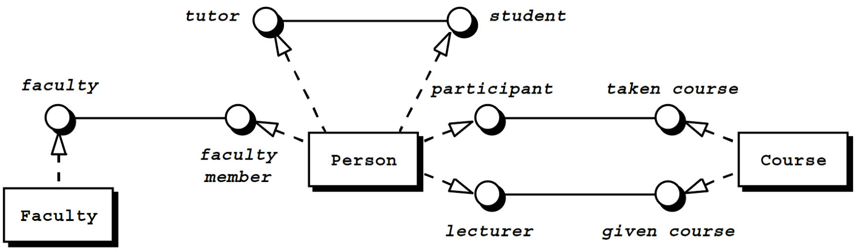


Fig. 3.14 Steimann [245]: example of the proposed role concept in UML

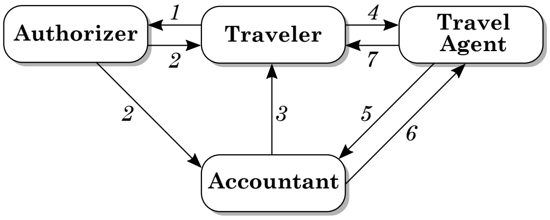


Fig. 3.15 Zhao [285]: role interaction diagram for business processes (figure adapted from original)

[12] At runtime, each object's instance has its own instance of the role type attached.

individualized labels, which is not wrong per se. With that, no further meaning is set in the role's semantics. Nevertheless, this is, without a doubt, far behind the potential functionality a role can provide if treated as an individual and unique modeling concept [16, 54, 85, 144, 209, 221]. E.g., for the UML case, Steimann [245] proposes a revised approach for the role concepts in which the role is an individual concept and objectified like a new class in the UML metamodel (between classes and their associations ends; see Figure 3.14). The approach of Zhao [285] uses an explicit role concept for modeling sequence diagrams with more precision towards individual business use cases and processes (see Figure 3.15). Additionally, the explicit role concept provides advantages on a technical level as structure and interaction are mainly specified independently of the classes that deliver instances and implementation [245]. Their collaborative nature improves the implementation of software systems if roles are made available explicitly [56, 151]. Roles can also be used to replace complex constructs like multi-inheritance and stacking interfaces [85]. The dynamics of roles can be used to express temporal statements of fixed entities [239]. The application areas are manifold and often concerning flexible and collaborative environments since this is the nature of a role.

The semantics of the role concept used in this thesis is stated in Section 4.4.2, where the BROS role concept is introduced explicitly. For further reading, an in-depth analysis and explanation of a role's semantics are contributed by the work of Kühn [149]. It describes the role as a fundamental modeling and programming concept and introduces further information and surveys about role semantics. Riehle and Gross [209] introduce roles as modeling construct with analysis of benefits and notations compared to traditional object-orientation. The works of Almeida and Guizzardi [9] and de Cesare et al. [61] give detailed insights into the role concept in its foundational and ontological theories. Further work regarding foundational role semantics can be found in, e.g., [10, 27, 101, 144].

3.3.1.3 Context

Across the disciplines, a role is considered to be contextual. Context defines the role's intention and meaning. A role without context is possible, but (at least) questionable. A role's context has a direct impact on whether the role can be played and with which other roles it is connected. Also, for (conceptual) modeling, the role's context can help to define and understand a role's purpose and task. Additionally, context-dependent applications can easily adapt to specific requirements at runtime by using the context's roles [149]. A popular context definition is stated by Dey [64]:

Definition 3.8 Context "Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves." [64, p. 2]

In practice, the context is often substituted with other terms, like collaboration, environment, institution, team, relations, stage, or ensemble [151]. Also, there is a mismatch between the different understandings of context in different areas of computer sciences [152]. On the one hand, the definition of Dey [64] identifies a context as "any information [. . .] to characterize situation of an entity" [64, p. 2]. This implies that the context neither is objectified nor make use of "inner parts" that are attached to the entity to characterize it (that is, there are no concepts like roles within a context) but simply chunks of information. On the other hand, the context definition of Kamina and Tamai [138] specifies a context as

> "Each context consists of a set of roles that represents collaborations performed in that context." [138, p. 1]

Thus, a context is specified for roles and is used as a construct that combines nested roles for collaboration. It neither states explicit information content for a playing entity of the roles nor states other elements within a context. In order to develop a specified role modeling language, the context is required to be specified and unified.

The ambiguous use of context in many models and their languages can be regarded as the overall acceptance of context-dependency of roles but no concrete specification of context. In Kühn et al. [151], the concept of *compartment* is introduced, which summarizes the different understandings of context under a common concept:

> "In turn, we use the term *Compartment* as a generalization of these terms to denote an objectified collaboration with a limited number of participating roles and a fixed scope." [151, p. 146]

A compartment, as an objectified context, is able to define a role's context in a concrete way. The compartment can make use of role features as well (e.g., use fields and operations, play other roles and can be played, can be nested in other compartments). A summary of a literature survey shows that most modeling and programming languages neglect an appropriate implementation of an objectified context [151]. The resulting modeling language in Kühn [149] makes use of compartments as an individual modeling language concept (see Section 3.3.2), which is of further use in this thesis as a static context (see Section 4.5.2).

In general, most software is based on contextual business logic [149]. The context-dependency is often underrepresented in current modeling approaches (models and languages), thus, there is a compulsory gap of what has to be modeled and what is actually modeled. Without context, most model statements are not possible or, at least, not without complex statement constructs. As a consequence, context (with roles) allows objects to act differently without losing their identity or type. Finally, this allows the software to adapt and change for specific use cases at runtime.

3.3.1.4 Classification and Specification

Roles, as a modeling concept, are known in their (linguistical) meaning and effects on objects. However, the concrete specification of a role requires a set of properties, rules, and characteristics that identify the role.

According to Kristensen [144], the role concept has several defining characteristics, which can (or should) be found in every application of the role concept. The following six basic properties of a role concept are outlined:

Visibility The visibility and access to an object can be restricted by solely using its fulfilled roles.

Dependency The role is always dependent on the fulfilling role and derives its structure and behavior.

Identity The object and its roles have only one common identity (that is, the roles share the object's identity).

Dynamicity Roles can be played and dropped by the object at any time and gain temporal access to the roles' structure and behavior.

Multiplicity An object can have multiple instances of the same role at the same time (accordingly, different objects can have their instances of the same role at the same time).

Abstractivity Roles can be classified and organized by aggregation and generalization hierarchies.

These characteristics are foundational guidelines and adopted widely in the role-based modeling world. Despite their fundamental nature, some of the characteristics are not used (or, at least) not as stated). E.g., abstractivity is often neglected (e.g., [149]) as inheritance and aggregation between roles can lead to semantic inconsistencies, which is problematic for technical implementation.

The work of Kühn et al. [151] goes beyond the six characteristics and state a set of 27 features that can be applied to a multitude of role-based modeling languages. Often, the features go along with the six characteristics stated by Kristensen [144]. The first 15 features are directly derived from Steimann [244]:

1. Roles have properties and behaviors
2. Roles depend on relationships
3. Objects may play different roles simultaneously
4. Objects may play the same role (type) several times
5. Objects may acquire and abandon roles dynamically
6. The sequence of role acquisition and removal may be restricted
7. Unrelated objects can play the same role
8. Roles can play roles
9. Roles can be transferred between objects
10. The state of an object can be role-specific
11. Features of an object can be role-specific
12. Roles restrict access
13. Different roles may share structure and behavior
14. An object and its roles share identity
15. An object and its roles have different identities

These stated features are substantiated by other literature and often used in role-based modeling. Nevertheless, the previously mentioned context-dependency of a role is missing. As written, a context is vital for a role's meaning and functioning. Therefore, Kühn et al. [151, p. 146] extend the basic set with 12 additional features that take an (objectified) context into consideration:

16. Relationships between roles can be constrained
17. There may be constraints between relationships
18. Roles can be grouped and constrained together
19. Roles depend on compartments
20. Compartments have properties and behaviors
21. A role can be part of several compartments
22. Compartments may play roles like objects
23. Compartments may play roles which are part of themselves
24. Compartments can contain other compartments
25. Different compartments may share structure and behavior
26. Compartments have their own identity
27. The number of roles occurring in a compartment can be constrained[a]

[a] This feature was added later in Kühn [149, p. 34].

This set extends the basic set of 15 features from Steimann [244] and can be aggregated as a feature tree. The additional features are derived by an analysis of role-based modeling and programming languages that make use of the context as an individual concept [151]. As the set is intended to be a feature set, the can be composed in a way that is needed to build a role-based modeling language (and the related model type). Nevertheless, some features exclude each other (e.g., feature 14 and feature 15) and can not occur simultaneously. This is not a mistake but a necessary limitation in order to capture all different kinds of role-based families and describe them with the feature set. In Kühn [149], further description and reasons for features are listed.

As follow-up work and aside from the six characteristics and the 27 features, Kühn [149] describes the role concept in its semantics across the different applications and summarized three different role natures:

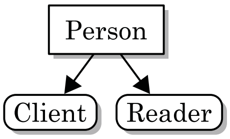
1. Behavioral nature
2. Relational nature
3. Context-dependent nature

They are stated in an abstract and generic way, in order to be applied to every role concept that can be used for role modeling. As they are partly overlapping the characteristics and features above, they are stated concisely.

> "In short, the **behavioral nature** emphasizes the role's ability to change the player's behavior, the **relational nature** highlights that roles are usually related to other roles, and finally the **context-dependent nature** captures that roles are defined within a certain action, stage, or more generally context. These three natures provide a simple yet sufficient classification scheme, as each nature focuses on an orthogonal aspect." [149, p. 23]

In general, the work of Kühn [149] is recommended for an in-depth analysis of the role's natures (including examples) and their foundational research in literature.

Fig. 3.16 Behavioral nature of roles: the object `Person` can have different structure and behavior, dependent on its runtime roles `Client` and `Reader`



Behavioral Nature

According to Kristensen [144], roles can be played and dropped at any time. In combination with the features set from Steimann [244], it is implied that roles enable to change an object's structure and behavior at any time as long as the object plays the role. Playing a role gives an object the possibility to participate in a specific context and get access to new structures and behavior. The behavioral nature can be summarized to the following facts that apply at runtime: (a) the role instance has exactly one player object instance, (b) one object instance can play several role instances, (c) if a role is fulfilled at design time by multiple objects, each object instance plays its own role instance, and (d) an object instance might play the same role multiple times (as multiple played instances) [149]. These properties, also partly represented on the features and characteristics above, can be derived by other works as well (e.g., [16, 244]).

> "In its core, the behavioral nature highlights that multiple roles can be played by unrelated objects and that all assumed roles of an object affect its state and behavior." [149, p. 25]

The special *fulfillment* relationship allows objects[13] to play roles at runtime, whereas the fulfillment relationship is placed at design time, the play relationship is used for runtime (when the object's instance actually plays the role's instance). By fulfillment, an object can make use of roles to change its structure and behavior (see Figure 3.16). In literature, many different terms are used as well (e.g., "role filling" by Frank [85] or "is-a relationship" by Snoeck and Dedene [239]). However, all the different variations are basically stating the fulfillment of a playing object towards a role.

Relational Nature

The relational nature of roles is often highlighted (e.g., [43, 85, 144, 245]) as roles are used to express the interaction between objects (ideally in contexts). The high prevalence of a role's relational nature was also noted by Kühn [149] when analyzing several existing role-based modeling and programming languages. It is referred to Chen [47] where the relational nature was precisely described at first:

> "The role of an entity in a relationship is the function that [the entity] performs in the relationship." [47, p. 12]

Based on that and concluding the statements of Steimann [244] regarding a role's behavioral nature, Kühn [149] argues:

[13] In several cases, roles can play other roles, too, called *deep roles* (cf. Section 4.4.2).

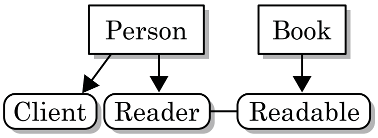


Fig. 3.17 Relational nature of roles: the interaction between `Person` and `Book` is established via a relationship of their roles `Reader` and `Readable`

> "[. . .] the relational nature requires that both roles and relationships are first-class citizens, that roles can depend on relationships [. . .], and that roles have their own properties" [149, p. 26]

In general, the roles are intended to denote endpoints of relationships. In an ideal case, the interaction between objects is based on a role-to-tole interaction (within a context). E.g., a relationship between the objects `Person` and `Book` could be placed in the context `Reading`, which uses the role `Reader` (for `Person`) and `Readable` (for `Book`). This relationship is only between the object's roles and placed in a context (see Figure 3.17). This is intersected with the context-dependent nature, as not only roles but also relationships between roles are established in a defined environment, a context that is used to state the circumstances of the interaction of roles. For that, the relationships of roles are assigned to a context as well [149].

> "To resolve this dilemma, it must be understood that roles can be context-dependent, as well as relationships themselves." [149, p. 27]

Context-dependent Nature

The context of roles was already introduced previously. That a context is necessary for a role's meaning is widely accepted. The context of a role provides an environment that defines (a) the collaboration of roles within, and (b) the participation of objects in a specific use case via these roles [56]. Currently, more recent models and modeling languages make use of the context concept as the resulting software has to be adaptive by changing contexts [149]. Identified by the feature set above, the objectified context should ideally consist of their own identity, properties, behavior, is able to play roles, and can be played by other entities [151]. There is one issue with a compartment that is going to be of additional interest within this thesis. In essence, the compartment, used by KÜHN [149] as an objectified context for roles, is stated to be used for every type of role context (e.g., situation, process, organization, collaboration). However, for a dynamic business process, the temporality of roles and their context is necessary

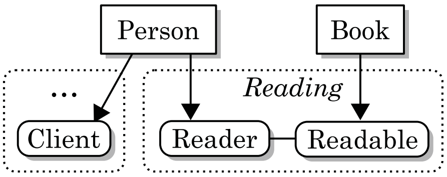


Fig. 3.18 Context-dependent nature of roles: the relationship and its roles are part of the context `Reading`, whereas the role `Client` role could be, e.g., part of a context `Buying`

in order to state behavioral collaborations like processes and flows. A compartment, with its static nature and without any temporal properties, is hardly appropriate for a process-like behavior context. Nevertheless, it captures the overall meaning. This is the reason why, in this thesis, a *scene* is introduced as a new modeling element type (see Section 4.4.3). As a result, with compartments and scenes, both types of contexts are covered.

> "In conclusion, the context-dependent nature of roles enables the representation of the individual collaboration, process or compartment a set of roles depends on. In short, it captures the existential dependency between roles and compartments." [149, p. 29]

3.3.1.5 RoSI – Role-based Software Infrastructures

Roles are a subject that affects every part of a software stack and lifecycle. Roles can be used on a conceptual and theoretical level but also as part of a direct or implicit technical solution. The consistency and continuity of roles are, nevertheless, not yet established. Within the single different layers and domains of software systems, the role concept might be established, but there is no *common* understanding of a role concept that is used in every layer and domain.

The program of *Role-based Software Infrastructures for continuous-context-sensitive Systems* (RoSI)[14] contributes to overcoming this gap in research and practice by developing solutions on all different layers and domains of the software technology and development stack:

> "The central research goal of this program is to prove the feasibility of consistent role modeling and its applicability. Consistency means that roles are used systematically for context modeling on all levels of the modeling process. This includes the concept modeling (in meta-languages), the language modeling, and the modeling on the application and software system level." [`http://www.rosi-project.org`, acc. 04.2020]

The vision to have a unified and consistent definition and implementation of a role drives the various projects related to this program. The projects are distributed across many disciplines (software engineering, database technology, systems and network architecture, security and privacy, business informatics and information systems, computational theory and logic, compiler construction), whereas the business informatics and information systems discipline is host to this thesis' project.

Across all involved disciplines, there are three core areas in which the projects are divided:

1. Roles in conceptual and language modeling
2. Roles in software engineering
3. Roles at runtime

The first core area is covering the research projects that analyze and develop theoretical foundations of the role concept (including its meaning, semantics, dissemination, and interpretation). Project outcomes of this area are usually more abstract and generic,

[14] `http://www.rosi-project.org`

however, usable by the other areas (e.g., [15, 49, 46, 150, 256]). The second area, software engineering, is the environment for this thesis. Although the thesis's artifacts are contributing to the first area (conceptual and language modeling), the intention and purpose is an improved software development process on the abstract level. Other project outcomes in this area are related to the overall software engineering process as well, establishing the role concept in both software development processes and artifacts. The aim of this area is to use the role (semantically established by the first area) continuously and consistently along the development process and through the whole software stack (e.g., [120, 134, 143, 159, 228, 276]). The second area is partly overlapping with the third, roles at runtime, as there is an ambiguous borderline between both. The constructs and technologies developed in area two are directly implemented and tested in area three, along with the development of new technologies to support the role execution at runtime itself and to develop new solutions based on runtime roles (e.g., [62, 148, 229, 252, 275, 283]).

The combination of the three areas allows continuous and consistent use of the role concept through the software development and technology stack, using the role as a concept down to a technological part of the implementation. The new role paradigm allows creating improved and extended software infrastructures. So, this thesis does contribute to the unified role concept as well, stating a way for using roles in conceptual modeling and software architecture. Never before has the role concept been explored in such a coherent way.

As a conclusion of the role's nature, the role subject, in general, has a substantial foundational and extensive research that can not be covered by this thesis completely. For a complete overview of the nature of roles, Kühn [149, p. 23–34] is recommended. The following sections give an insight into two thesis related areas where roles are of significant interest.

3.3.2 Roles in Software Engineering

The role concept is already established in software engineering disciplines for a longer period. Many facts about the role concept were already stated in the previous Section 3.3.1. This section summarizes the usage of roles for different software engineering subjects, modeling and programming languages, and technologies as well as systems that support the role-based paradigm.

Roles are a well-known (but differently used) paradigm in the software engineering area. In the past decades, the work of Bachman and Daya [16] used the role concept to describe properties of data objects that are not belonging to the core entity but a separate part of the core. This is followed by several applications of roles within software engineering. A popularly cited researcher in this area is Steimann (e.g., [245, 247, 248]). Related research is focusing on the representation of roles in the general software modeling and engineering discipline, e.g., revamping the role concept in the UML class diagram [245], implementing roles as OO-based interfaces [246], or modeling role data objects [248]. In Colman [54], roles are highlighted as a collaboration mechanism within software structures. The concepts of "player-centric" and "organization-centric" roles are introduced, where the former role type is oriented on the player object, and the latter is focused on the organizational boundary of business logic as a context. The work of ZHAO [285] introduces roles are collaborations as well, with respect to their interaction with each other. The work uses the context to represent a specific task in which the roles are involved. A linear execution of role interactions leads to a sequence diagram-like structure with roles.

Roles are partly represented in common modeling and programming languages as well, even if not in their full potential. Several role-based modeling languages (and respective models) use the role as a unique modeling element (a "first-class citizen"), e.g., [59, 111, 119, 149, 237, 244, 286]. Although they use their own interpretation of a role, it is possible to instantiate the role as a unique entity and use it with regard to the business logic. Further, various programming languages are able to treat a modeled role as a unique element as well. This allows easy access, implementation, and handling of roles when actually implementing the software system. The domains of programming languages, like the modeling languages, are diverse and range from data to structure to functionality (e.g., [18, 102, 113, 115, 121, 134, 160]). This thesis will not go into detail regarding the different modeling and programming languages, as an in-depth analysis and survey of the different perspectives, advantages, and drawbacks are given in KÜHN [149], always including a running example of a banking domain model for the specific modeling and programming languages. The work compared the languages with each other and identified the 27 features in a feature table. As a result of the survey, it is stated that

> "[. . .] the research field on role-based modeling and role-based programming languages suffer from fragmentation and discontinuity [. . .]. Especially, as most approaches reinvent the notion of roles without taking previous definitions into account." [149, p. 94]

This is regarded as a consequence of a lack of a common understanding and interpretation of the role concept [149]. As a follow-up result, only two features (the first "roles have properties and behaviors" and the third "objects may play different roles simultaneously") are shared and acknowledged by all role-based approaches. The remaining features differ between the surveyed approaches. This is, according to KÜHN [149], interesting as many early works already standardized the role concept (e.g. [245]). As a summarize, the following research results are stated:

> "[. . .] the SLR uncovered the following problems in the research fields on RMLs and RPLs:
>
> 1. There is neither a *common understanding* nor *common feature set* shared among the different contemporary role-based modeling and programming languages.
> 2. The research fields on RMLs and RPLs are characterized by an ongoing *discontinuity* and *fragmentation*. Specifically, most approaches reinvent the role concept without taking the definitions of preceding related approaches into account.
> 3. Only four RMLs provide a sufficient *formal foundation* for roles able to incorporate *all natures of roles* [. . .]. Regardless, none of them is able to *support all features of roles*.
> 4. Last but not least, most role-based modeling and programming languages are *not readily applicable*, due to their complexity, ambiguous terminology, and/or missing tool support. Even though *OT/J* represents a feature rich, practically usable programming language, there is no corresponding *readily applicable modeling language*." [149, p. 95]

Despite their different understanding of a role, in literature, several developments support the execution of roles. This can be done by providing a runtime environment for roles (e.g., [251]), or frameworks that support the use of roles in a software system. In Kühn et al. [152], a framework is developed, which transform role models (CROM [149]) into inter-operable *Java* source code. This can be used to introduce roles in projects that do not support role-based programming languages. Other technical domains also profit from the use of roles. For example, in Werner et al. [277], managed roles are used to synchronize multiple models by applying roles to the used model concepts. In Werner and Assmann [276], this approach is extended for the use of a *single underlying model* (SUM), which is a completely specified reference used for various other model types and views. SUM-related models get adapted at a technical level and act as different views of the SUM. In Colman and Han [55], roles are used for synchronization as well. The developed approach makes use of roles and contracts to adapt software at runtime. Several works make use of roles for establish runtime collaborations (e.g., [62, 235, 275, 283]).

3.3.3 Roles for Enterprise Modeling

In enterprise modeling and information systems, the role concept is established as well, although not stated as an individual concept as early as in software engineering. Since enterprise modeling and information systems are partly based on scientific research of software engineering, there are overlapping works that combine both aspects. In turn, the foundational concepts of a role provided mostly by the conceptual modeling area can be used by the more traditional aspects of software engineering.

The role term and its concepts are subject to research in several disciplines of enterprise modeling and information systems (e.g. [168, 169]). Many approaches use the preliminary work of, e.g., Steimann [244] and Wieringa et al. [278] as an initial interpretation and definition of the role concept. Particularly Wieringa et al. [278]

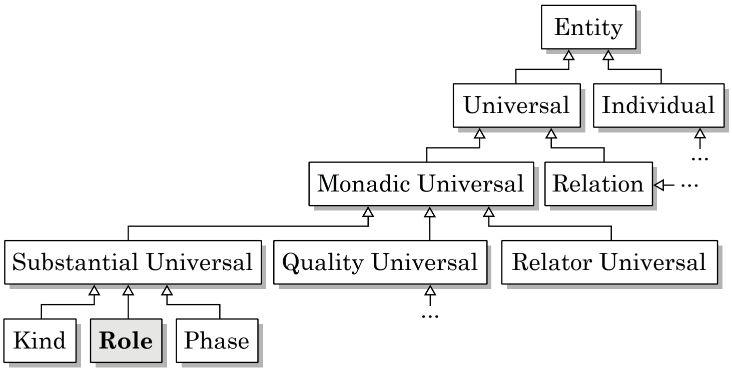


Fig. 3.19 Guizzardi [106]: excerpt of the UFO-A ontology with roles (figure adapted from original)

is often used as argumentation for a required role concept in enterprise modeling. In this work, the necessity of roles is demonstrated by the different roles of the entity during its lifetime, and a taxonomic approach is developed to describe the role's nature in an object-oriented world. The ontological basis of the role concept was further researched by, e.g., Guizzardi [106], who introduced the role concept (called "agent roles") into an ontology for structural conceptual models[15] (see Figure 3.19). With that, the role is combined with related ontology concepts of kinds (which can be regarded as sort of objects) and phases, a temporal property of a role and kinds (e.g., `Childhood` phase of the kind `Person`). As a result of the ontological analysis, the role is defined as

> "Thus, the material ***Roles*** employed both in conceptual modeling and natural language (e.g., Student, Customer, Supplier, Husband, Patient) are defined here as *anti-rigid and relationally dependent substantial sortals*." [106, p. 154]

The UFO-A ontology with roles is well-known in conceptual modeling so far, as it specifies the role concept (and related concepts) strictly. This foundational ontology was further of use for Almeida and Guizzardi [9] and Almeida et al. [10]. In these, the role enhanced ontology is introduced as an independent concept in combination with enterprise and conceptual modeling issues. The established role definition within the UFO-A ontology is compared with the feature set proposed by Steimann [244] and identified as consistent. Other works contribute to the ontological definition of a role as well (e.g., [34, 84]). In general, due to the strong emphasis on the theoretical foundation of the role concept in enterprise modeling, there is often a more coherent understanding of the role concept than it is in the general software engineering field. Nevertheless, the role concept in enterprise modeling (and conceptual modeling in general) is often only considering personal roles (e.g., `Employee` or `Sales Manager`, and not roles for things like `Car`), which can possibly be concluded by the human-focused alignment of the research field. The work of Fox et al. [84] uses the concept of *empowerment* as a status change activity that combines a role with an object dynamically at runtime in order to achieve a combination of structure and behavior. Övergaard [193] describes formal collaboration contexts for roles. This is combined with a behavioral model stating the sequences of messages between roles (similar to Zhao [285]). In Lê and Ghose [158], roles are interpreted as a combination of business contracts and goals of entities. It can be concluded that

> "[. . .] the role's responsibility and rights can be captured by contractual rules and goals, respectively." [158, p. 263]

However, this work only considers personal roles (that is, roles for human entities, and not for things). Roles for non-personal things in conceptual modeling is highlighted by Steimann [247]:

> "Conceptually, there is no difference between a document's being printable and a person's being employable; both require that the objects have certain properties that enable their functioning in the context defining the role. These properties are comprised in a corresponding role type. " [247, p. 173]

[15] The basic ontology was introduced by Guizzardi [105] as *Unified Foundational Ontology* (UFO) and its specialized variant UFO-A for "endurants".

Additionally, this work compares the roles with aspects, and it is argued that modeling aspects as roles is only possible when the roles are isolated from each other. This is, however, not intended by the role's behavioral nature (cf. Section 3.3.1). Olivé [189] reviews the role concept in a concise way, proposing its meaning as an individual entity type in conceptual modeling. The semantic of a role is considered to be a set of properties that enriches an object. This is stated by a pragmatic definition of a role:

> "A role *r* is a set of properties that characterize a situation which the instances of an entity type *E* may be in at a given time. We say that *r* is a role of *E* or that *E* plays the role *r*. Normally, the instances of *E* are in a given situation only temporarily, and not all instances of *E* need to be in the same situation." [189, p. 148]

In Boella and Steimann [33], roles are divided into relational roles, processural roles, and social roles. This categorization allows the distinction between the purpose and functionality of the role in combination with its player. Regarding this thesis, the processural role is of interest as they are stating a representative concept for entities that collaborate in a certain process context. Nevertheless, with an objective view of roles, all three role types still can be represented with the common role concept.

Besides the definition of roles in the conceptual modeling domain, several works cover the application of the role concept in the enterprise modeling and information systems domain. Based on the work of Ould [192], the work of Caetano et al. [43] introduces roles for business process modeling. The key point in its argumentation is that business objects interact with each other not directly but via roles [43]. The so-called concerns (that is, context) define an aspect of an involved business object, and the role carries the respective business object's behavior in that concern. Concerns might change dynamically during the process execution. As described in Caetano et al. [43], in Zhao [285], roles are used to define behavioral domain models and related business objects as well. The business objects are considered to play roles and interact with each other to fulfill a business process flow. This is done by introducing roles in a uniform combination of a conceptual domain model, RRC cards (specialized role CRC cards [22]), and sequence diagrams. The work of Almeida et al. [10] shows and compares five different conceptual modeling languages (*ArchiMate*, *DoDAF*, *ARIS*, *BPMN*, and *RM-ODP*) according to their suitability and implementation of the UFO-A role concept in enterprise modeling. As a result, it is stated that

> "In all approaches, roles are used to represent the participation of actors in particular behaviours or processes, decoupling the definition of these behaviour or processes from particular instances of actors. None of the approaches, however, discuss the dynamics of role playing or provide modelling elements to describe how actors are assigned to roles dynamically (except the RM-ODP)" [10, p. 275]

Interestingly, it is implied that the "scope of a role" is often neglected by the conceptual modeling languages, which can be compared with the context of a role. The modeling of such a scope could improve the use of the modeling languages, especially in large organizations [10]. For the general conceptual modeling area, several other works contribute to the introduction and specification of the role concept as well (e.g. [61, 85, 105, 239, 264]). Often, they are based on the same role idea, thus, they have a more common understanding of the role concept.

Last, various works are related to roles as they use established conceptual modeling role concepts for other domains. Nevertheless, this subject is sparsely populated with related work as, apparently, research is more focusing on the definition and foundational establishment of roles instead of using them in practical use cases and applications. The approach of Bley and Schön [30] uses the conceptual role for stating an enterprise's domain in a maturity model. Nevertheless, the used role definition is hardly consistent with the previously described role concepts in conceptual modeling.

> "[The] role specifies the selection and allocation of factors from the pool to the maturity levels [. . .]. As a role is utilized to represent a class or sector of enterprises (e.g., a role for the class of "small-sized non-international transport logistic enterprises"), it can be chosen or exchanged depending on the model user's intention." [31, 740]

Thus, it is rather used as a tool than a role in common sense. Another applied role-based use case is done in Schön et al. [224], where conceptual requirements and capability models are enhanced with a role concept. The work shows how requirement and capability modeling can be improved when combined with the role concept. Finally, it has to be noted that, as far as known, no reference model adaptation approach makes explicit use of the role concept so far.

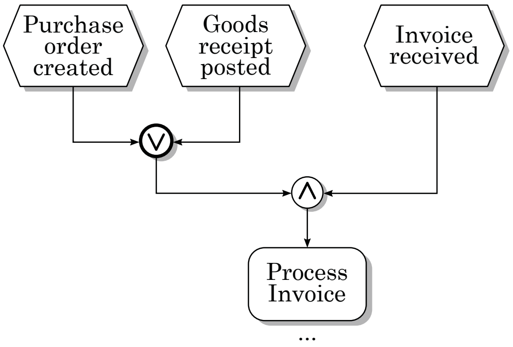
3.4 Related Work and Research

This thesis proposes a new reference model adaptation methodology called *Business Role-Object Specification* (BROS). This section states the most important works that are feature-wise connected and influenced its development, often already mentioned in the previous sections. The most significant related work of reference model adaptation is concisely stated in more detail and sorted in the two main areas: systematic and dynamic adaptation. Still, not every related approach can be covered.

3.4.1 Systematic Adaptation

Often, approaches for a systematic adaptation are based on processes and workflows, as adaptation and configuration are a core part of enterprise modeling. Nevertheless, the used mechanisms for adaptation can partly be of use when adapting structural models as well.

Fig. 3.20 Dreiling et al. [71]: example of a C-EPC workflow (figure adapted from original)



3.4.1.1 Configurable Workflow Languages and Models

The landscape of configurable models is large and has a lot of related research (in Döhring et al. [68], a comprehensive comparison is given). This section states the most relevant related work approaches for the thesis.

Configurable Event-driven Process Chains (C-EPC)

The probably most common adaptation approach is using C-EPCs. The C-EPC workflow is, obviously, adapted by the configurable class [126] (cf. Section 3.2.2). Thus, the reference model (the C-EPC process) is covering the whole domain and has points for configuration implemented. Via selection, the application model (a configured C-EPC or standard EPC) can then be used by the enterprise. Related work in this area are, e.g., [71, 154, 177, 203, 206, 212]. A small example is given in Figure 3.20. The bottom function `Process Invoice` is triggered by the left *AND* right event path above. The *AND* is set by the non-configurable circle above (with the ∧ in it). The right event path (`Invoice received`) is needed as well as the left event path. The left path uses two events that are combined via a *configurable OR*: it allows the configuration into the following alternatives: *OR*, *AND*, or *XOR*, dependent on the enterprise's requirements [71]. The C-EPC language and its models allow multiple configurable parameters like these (also functions) to derive the individual application model. This adaptation method is often used due to its configuration approach: the reference model is stated by domain experts, and the adaptation points are explicitly declared. Thus, every possible application model is coherent with the standards. Nevertheless, this paternalism by only using defined adaptation points also leads to a limited adaptation space for the enterprise's needs. If the given adaptation point set is not sufficient for the enterprise's needs, the whole reference model can not be used. Further, this approach lacks any combination with the structure of the underlying software system. Last, the available adaptation statements are restricted to functions and operators and often provide Boolean decisions only. More complex adaptations are hardly possible.

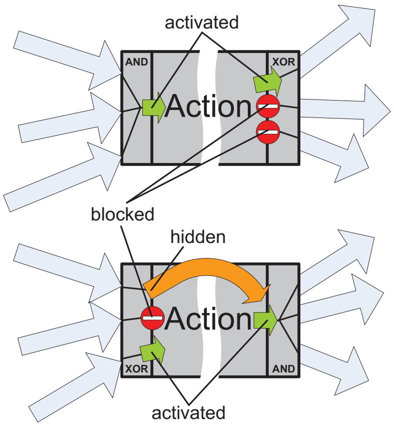


Fig. 3.21 Gottschalk et al. [99]: example of two configurable actions in C-YAWL

Configurable Yet Another Workflow Language (C-YAWL)

The C-YAWL modeling [99] language and its models follow the same paradigm as C-EPCS: a configurable adaptation. It is based on the regular YAWL language [260] and includes multiple configurable parameters additionally. The reference model process is given by a completely specified process, and adaptations are made via selection. A simplified example of two configurable operators (i.e., activities) is given in Figure 3.21. The upper activity has three *AND*-combined inputs and three *XOR* outputs. Contrary, the bottom activity has three *XOR*-combined inputs and three *AND* outputs. The inputs and outputs are "ports," where other activities may be connected in a process flow. The configuration of both activities is made via configuration tokens: the green arrows (activate), red stop signs (block), and yellow arrows (hidden). The activation directly calls the activity's content, blocking disables the flow for that input, and hidden allows the process flow to continue without any execution of the activities content. A configuration is made by assigning the tokens to the input and output ports. In the upper example, the activity is executed due to the activation of the *AND* ports. However, only one of the three outputs is allowed, the others blocked. In the bottom example, the first input is directly rerouted to the output, the second input is blocked, and the third activates the activity's content. By assigning the tokens and configuring the process flow, the reference model's flows can be adapted towards individual needs. Due to its close affinity with C-EPCs, the C-YAWL takes advantage of the same benefits but also suffers from the same drawbacks. Its higher amount of configurable parameters (i.e., more options to choose from) allows this approach to be more enterprise-specific. Although it has more configurable parameters than C-EPCs, it still depends on the initial modeled reference model as every adaptation is made via selection. Processes that do not reflect the enterprise's requirements can not be used for adaptation.

Provop

Other approaches are also available, however, not that prominent in practice. For example, the *Provop* approach [110] uses similar configurable operators like C-EPC, but its models have more similarities with BPMN models. The purpose of Provop is to derive different variants from a given template. For that, it allows changing the process flow with regard to deletion, modification, and insertion of activities (see Figure 3.22. While this approach has several configurable elements (delete, modify, insert, adjustment points, connectors, mappings, anchors), is has nearly the same advantages and drawbacks as the other configurable approaches.

Virtual Business Process and Model Notation (vBPMN)

Another example is *Virtual Business Process and Model Notation* (vBPMN) [69] that allows adaptation (i.e., a configuration of variants) of standard BPMN processes. For adaptation (that is, deriving variants from the template), it uses adaptation patterns and segments that apply to the BPMN process. The patterns and segments then get weaved into the BPMN process in order to achieve the finally adapted application (process) model (see Figure 3.23). Therefore, this approach can be regarded as instantiating rather than configuration [126]. Additionally, this approach comes with the support of a DSL that specifies the adaptation at runtime as a set of rules with the notation stated in the Listing 3.1.

```
ON entry-event
IF <data-context>
THEN APPLY [<pattern((parameter=value)*)>]*
```

Listing 3.1 Döhring et al. [69]: Runtime adaptation rule

This allows a description of the adaptation in textual form and, therefore, enables an easier transformation towards implementation.

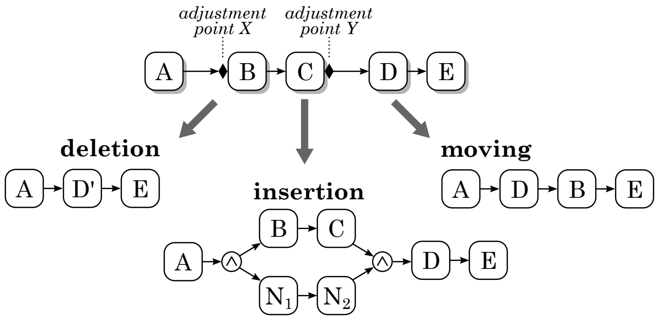


Fig. 3.22 Hallerbach et al. [110]: example of a configurable Provop workflow (figure adapted from original)

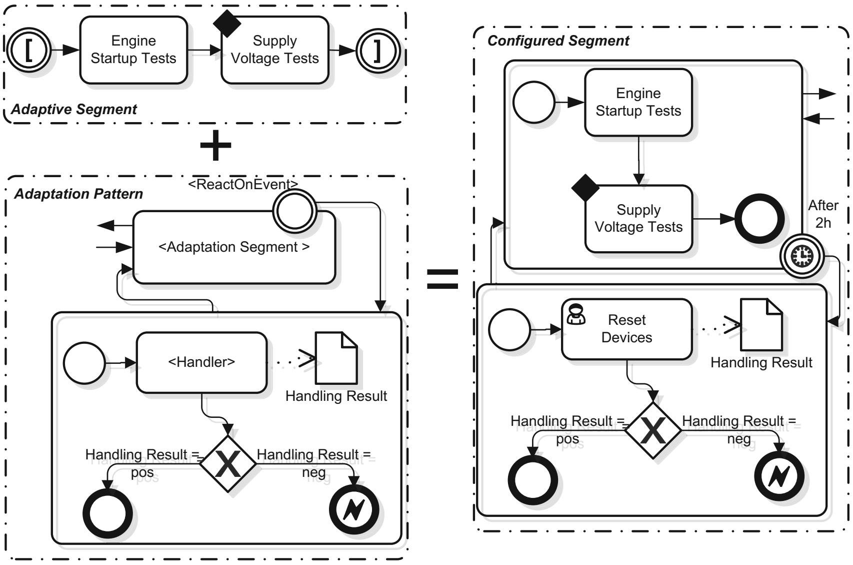


Fig. 3.23 Döhring and Zimmermann [69]: example of a configurable vBPMN workflow

3.4.1.2 Structural Languages and Models for Adaptation

Besides process models and languages, there are rather few approaches that can be used to adapt reference structures according to enterprise-specific needs.

Data Warehouse Items

One example is the work of Schütz and Schrefl [227] that describes the structural adaptation of single items on a data warehouse system (the items can also be regarded as business objects, nevertheless). Their approach supports the basic adaptation operations like adding, removing, or modifying different object dimensions. Therefore, one can, e.g., remove the value `quarter` of the object's `Time` dimension (see Figure 3.24). This serves the purpose of adapting a given structural data entity towards enterprise-specific needs. The possibilities of adaptations of the data values are categorized into four levels: dimension and facts classes, calculated measures, data marts, and roll-up data marts. These four levels of values provide different degrees of abstraction to customize the application model further. With this approach, the user of a reference model can adapt it towards the enterprise-specific application model (in this case, a data warehouse model). This approach has the advantage that it is suitable to be implemented as it delivers additional technical information. For that, the necessary database schema generation for the modeled data items is proposed as well. On the contrary side, it misses possibly related behavior information (e.g.,

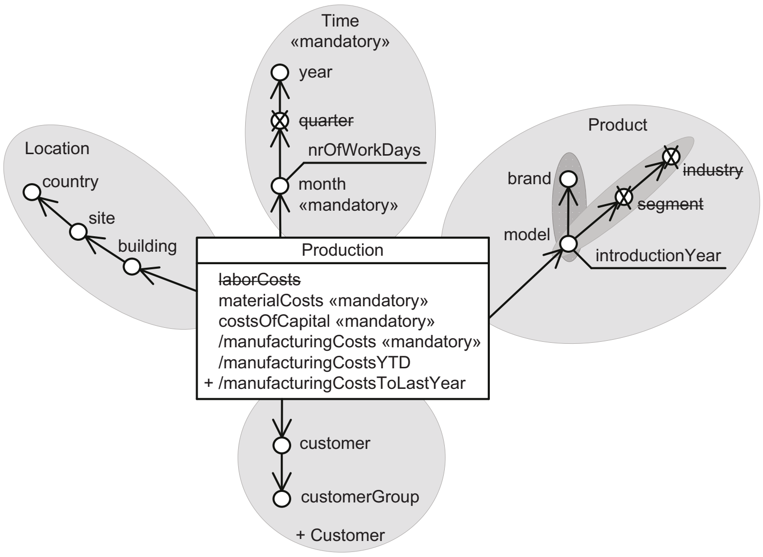


Fig. 3.24 Schütz and Schrefl [227]: example of an adaptation of a data warehouse item

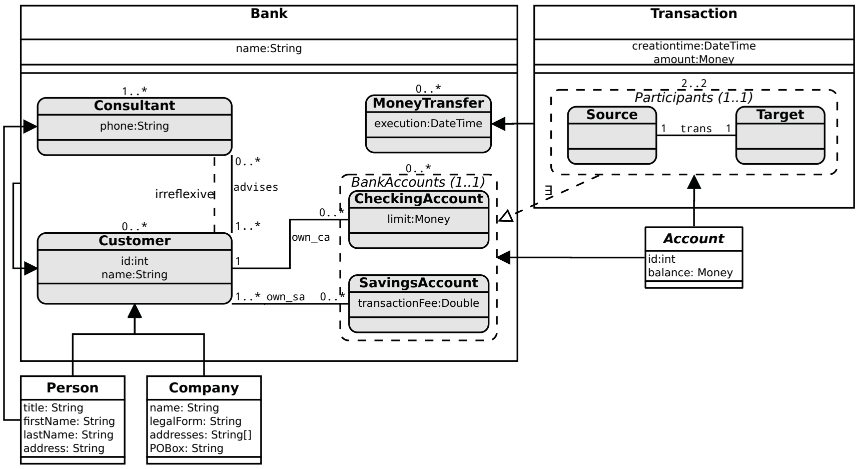


Fig. 3.25 Kühn [149]: example of a role-based structural model

related business processes) that led to the specific adaptation. Further, relationships between objects (data items) are not clearly stated. This is, of course, due to the fact that this model is used for data items, and relationships are done via key entries. Nevertheless, if used for general structural reference modeling, this approach needs to be extended to consider full-fledged relationships (and their adaptation).

Compartment Role Object Model (CROM)

The approach of Kühn [149] proposes the *Compartment Role Object Model* (CROM), a notation for role-based models, which is already introduced in the previous sections (cf. Sections 3.3.2). The model and its language combine all natures of roles, and the most role features are served [149, 244].

> "[CROM is] a framework for conceptual modeling that incorporates most of the features of roles into a coherent, graphical modeling language and provides a set-based formalization of roles." [149, p. 21]

The CROM language constitutes the basis for the developed BROS language in this thesis. CROM offers a set of model elements to describe a role-based system landscape. The three most important ones are:

Natural	Fixed domain entity with fields and operations
Role	Role entity that is fulfilled by naturals and changes their structure and behavior (as described in Section 3.3.1)
Compartment	Context entity that contains roles and relationships (as described in Section 3.18)

Figure 3.25 states an example CROM model with naturals, roles, and compartments in a banking domain. The used syntax of CROM is summarized in Figure 3.26, the full documentation of CROM can be found in Kühn [149, p. 99–137]. The concept of naturals, which can be regarded as objects, denotes the fixed entities within a domain. These naturals are then extended by roles at design time (at this point, this approach is similar to the adaptation mechanisms of Schütz and Schrefl [227]). Multiple adaptations could be made via using different compartments and roles fulfilled by the objects, while the compartment could be adopted to represent business processes.

Additionally, for CROM, there are already two formal models *formalCROM* [151] and *ConDL* [36], a textual notation *TRoML*[16], a reconciled database integration *RSQL* [134], and a graphical editor[17] available, which allows easier practical integration. In summary, CROM has the potential to be used for reference modeling since roles and their contexts can be added and removed at any time, and roles add structure and behavior to their players. This can be considered as an adaptation mechanism (see Figure 3.25) of a basic CROM "reference" model as a template. Nevertheless, CROM was never intended to be used for reference modeling. Various semantic definitions and functionality have to be adopted and re-specified in order to use CROM for the adaptation of enterprise reference models. Further, temporal descriptions of roles are missing, making relation with business processes more difficult. Additionally, the work of Kühn [149] introduces and compares a set of other role-based modeling and programming languages that, like CROM, could possibly be used as an adaptation approach as well. Nevertheless, they are all not directly purposed for reference model adaptation. Although they serve the needed basic functionality, further substantial adjustments need to be made in order to consider them as adaptation approaches.

[16] `https://github.com/Eden-06/TRoML`, acc. 04.2020

[17] See Appendix C.2 and `https://github.com/Eden-06/FRaMED-2.0`, acc. 04.2020

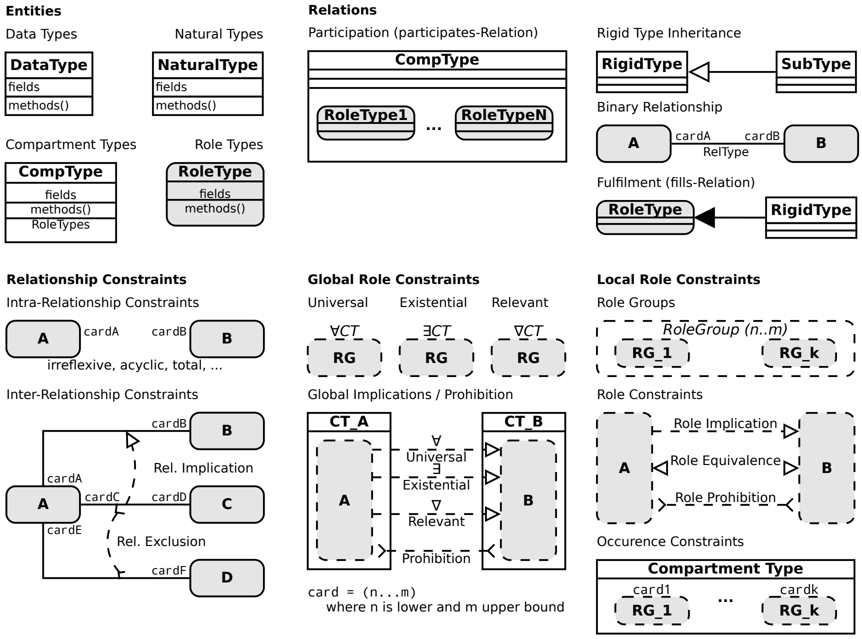


Fig. 3.26 Kühn [149]: the CROM syntax

3.4.2 Dynamic Adaptation

Since the thesis' artifact is considering the combination of structure and behavior to represent the behavioral aspect of business processes, several works have provided the foundational framework for temporal and behavioral natures in modeling. The main mechanisms of BROS are roles and context, thus, both need to be able to be specified in terms of time. This section provides a brief overview of the most relevant work regarding this thesis' subjects.

3.4.2.1 Temporal Behavior Modeling

Time representation has an extensive research community in the conceptual modeling area. Despite the fact that temporal modeling constructs (in conceptual and domain modeling) are hardly used in practice, they are often well-founded and defined.

Considering the theoretical foundations, the work of Allen [6] has influenced many other works by introducing a classification of sequence relationship occurrences and their formal definition (see Table 3.1). The resulting classes are commonly known as "Allen's operators" and further used for other research. Based on (or related to) these, a lot of other research was done (e.g., [8, 60, 73, 139, 266]). In particular, in

Table 3.1 Allen [6]: set of sequence operators

Relation	Symbol	Inverse	Pictoral Example

X before Y	<	>	`XXX`
			`    YYY`

X equal Y	=	=	`XXX` `YYY`

X meets Y	`m`	`mi`	`XXX`
			`   YYY`

X overlaps Y	`o`	`oi`	`XXX`
			` YYY`

X during Y	`d`	`di`	` XXX`
			`YYYYYYY`

X starts Y	`s`	`si`	`XXX` `YYYYYYY`

X finishes Y	`f`	`fi`	`    XXX`
			`YYYYYYY`

Guizzardi et al. [108], the operators are discussed and formalized as axioms. The work of Halpin [112] identifies the conceptual issues and drawbacks of the operators.

In practice, the work of Selic and Gérard [232] introduces a framework rather well-known in industry applications: the *Modeling and Analysis of Real Time and Embedded systems* (MARTE) UML profile for modeling time-dependent software systems. A UML profile allows the redefinition and extension of standard UML classes.

> "In simplest terms, a UML profile is a facility for augmenting with supplementary information. Typically, this is information that can not be expressed directly using standard UML." [232, p. 28]

This can be done in two different ways: (a) extending the UML language or (b) attach additional information to models [232]. The MARTE approach adds temporal information to UML class diagrams and entities. In general, the additional information is specified time values that limit the lifespan of entities or messages, or state specific points in time after something should happen. The MARTE profile is mainly used in practice to model critical systems that are in need of verification (e.g., in the aerospace domain). UML profiles have, despite their usefulness, the drawback that they do not serve temporal entities as a "first-class citizen" in models. Temporal entities will always be UML classes and use the common assumptions (e.g., use inheritance, types are static). However, the UML profile approach works on established industry UML models and can (theoretically) be added at any time.

In Cabot et al. [42], temporal information is introduced into the UML modeling language, more precisely into UML class diagrams. They use the same method as proposed by Selic and Gérard [232] by using a dedicated UML profile for temporal entities. For example, the class `Employee` could be of `«temporal»` type[18]

[18] Interestingly, this is done for many classes that actually are rather roles. Thus, instead of roles, temporal classes with `Frequency=intermittent` are used in these models.

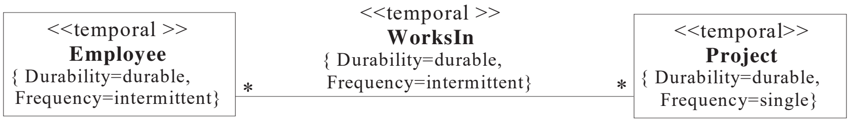
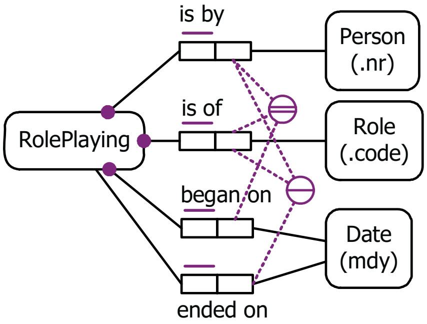


Fig. 3.27 Cabot et al. [42]: temporal classes in UML as UML profile

Fig. 3.28 Halpin [112]: roles associated with temporal facts



(see Figure 3.27). Temporal information is added as OCL constraints. Further, this approach allows lifespans of objects and model statements about an object's frequency. As this approach is based on UML profiles, it comes with similar advantages and benefits as the MARTE approach.

The *Object Role Modeling* (ORM), proposed by Halpin [112], comes with temporal statements as well. This work uses temporal rules on a conceptual level in order to reason over temporal statements. Temporal facts are added to roles that state the nature of the temporal fact. In particular, the role "play" relationship is extended by temporal facts (see Figure 3.28): if an object plays a role, the respective relationship can have temporal information `began on` associated with any specified `Date`. Temporal periods of entities are described as OCL time constraints. The temporal defined entities can then be of three different types: definitional (which is true by default), once-only, or repeatable [112]. Further, time is only represented as role "play" associated information; there is no object that represents temporal facts on its own.

3.4.2.2 Event and Scene Modeling

Behavioral modeling can, with the necessary abstraction, be split into a chain of "happenings" (e.g., events, actions, activities) and their "environment," in which they achieve a certain goal (e.g., process, behavior flow, task). The former can be regarded as events, the latter as their context. Since context is a more generic term for *any* context, this thesis uses the term *scene*[19] to state a behavioral (dynamic) context.

[19] See Section 4.4.3 for the concrete BROS scene's semantics.

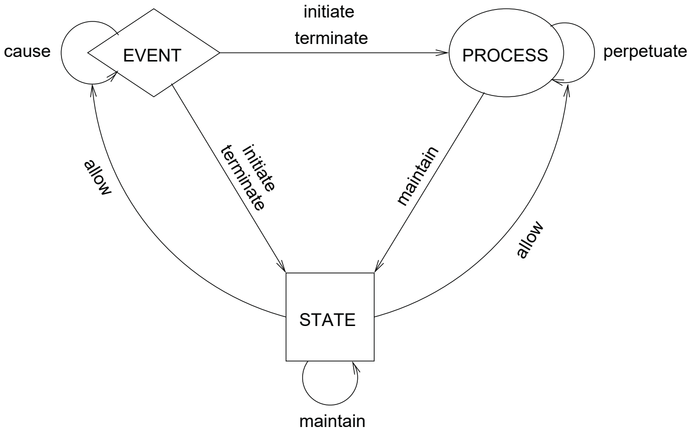


Fig. 3.29 Galton [91]: causal and causal-like temporal dependencies of event, state, and process

Since events and scenes are of special interest in this thesis, this section states a comprised overview of both concepts from literature.

Events

The event is a widespread concept in every modeling domain. Like roles, it is a common term used in natural language and describing use cases that depend on temporal requirements. However, there is no real common understanding of what an event is used to be and how it is defined. This differs from domain to domain (e.g., an event in EPC is conceptually different from and event in BPMN or user story events). The meaning and nature of events can be of tremendous variation among different approaches. Nevertheless, there is no possibility and requirement to uniform all event concepts possible, however, a strong meaning of an event is a prerequisite to defining event-based concepts in this thesis any further[20].

Different works cover the subject in different ways. A very extensive perspective on an event's nature is given in Galton [91]. Derived from a philosophical background of causality, the event is an occurrence that is caused by an object, another event, or a fact [91]. The event's occurrence leads to a specific state, which is part of a process. The interrelationship between the three concepts is explained in detail on an ontological basis. Further, this work introduces event types and the event itself as a particular version of that type:

> "Events are usually identified with reference to an event type, e.g., the statement 'John baked a cake yesterday' identifies a particular event as an instance of the type 'cake-bakings by John' (itself a subtype of the cake-bakings generally), pinpointing a particular instance as the one which took place yesterday. I take it that all this is relatively uncontentious." [91, p. 8]

[20] Please note, that the concrete BROS event semantic is described in Section 4.4.4.

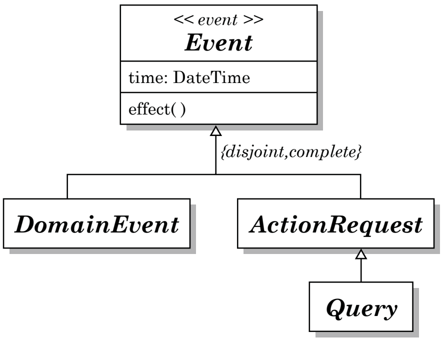


Fig. 3.30 Olivé and Raventós [190]: top-level taxonomy for event types (figure adapted from original)

The ontological foundation is further covered by Guarino and Guizzardi [104], which identifies the event as an ontological entity that

> "[. . .] is determined by a couple $< r, f >$, where r is a spatiotemporal region, and f is the event's *focus*, consisting of a collection of individual qualities, which we shall call *focal qualities*." [104, p. 246]

The focus is given by the used context (a scene) that is based on the used relationships [104]. The work of Kaneiwa et al. [139] considers events on their ontological level as well. This work distinguishes between natural events (that occur without any actors involved) and artificial events (actor-based occurrences). It introduces formalized relationships between events, state changes, and causes as well as semantic functionalities of events.

Regarding the conceptual and OO-based modeling, the work of Olivé and Raventós [190] analyzes the event concept and how it is used across the different model types. In order to introduce the event as a unique modeling element in (UML) OO models, a new UML type for a class is introduced (called "event"). The supertype of an event can be distinguished between domain events (caused by the environment) and action requests (caused by user interaction), illustrated in Figure 3.30. Further, event constraints and prerequisites are developed (based on OCL) to enable an easier transition towards the implementation of events. In Halpin [112], a set of defined events leads to assignments of roles to temporal information. Popović et al. [197] propose the event as a platform-independent model element that contains a source, defining business logic, an event handling level, and a type. The developed event concept describes its business logic in the textual notation *IIS*CFuncLang* [196], which is designed to be implemented, especially by relational databases [197].

Nevertheless, in practice, events are often only used implicitly (e.g., as textual statements) or "standard" classes like every other entity. Technologies allow the definition and handling of events for communication and reaction (e.g., via message bus *RabbitMQ*[21]). However, these are technical solutions and, therefore, inappropriate for top-level and conceptual modeling of the system's landscape. Approaches that state events or scenes as individual and unique model elements are rare.

[21] `https://www.rabbitmq.com/`

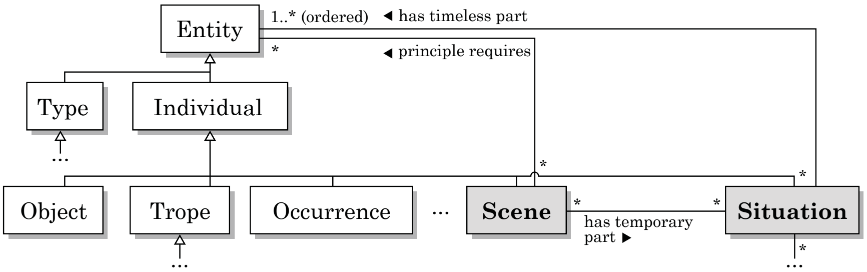


Fig. 3.31 Almeida et al. [8]: Ontological view of scenes and situations (figure adapted from original)

Scenes

Scenes are often considered to be a context of events (and roles) [8]. However, conceptual model research on scenes is not as popular as on events. The ontological basis is given by, e.g., Almeida et al. [8], where scenes are encapsulating events, which, in turn, lead to different situations of the scene. Thus, a process flow (represented as a scene) can be regarded as a chain of temporary situations that are driven by events. Both the scene and its situations are dedicated entities in the ontology (see Figure 3.31). The scene is also part of the work of Guarino and Guizzardi [104], where the ontological event is analyzed, too. There, the scene is considered to be

> "[. . .] whatever happens in a suitably restricted spatiotemporal region." [104, p. 245]

With that, two major constraints are applied to the ontological scene [104]:

1. The scene can not be an instant but is always a temporal duration
2. A scene is located in a convex region of spacetime: it occurs at a certain place during a specified time interval

Further, a scene contains all participating and involved participants during its spacetime [104]. The ontological scene is host to events that happen in its spacetime.

In practice, the scene is not well established yet (at least, not as an explicit concept). However, several models and languages support context elements or concepts as individual modeling elements (e.g., CROM [149]). This context can (partly) be regarded as a scene, if (and only if) this context fulfills the previously stated ontological constraints, that is, the scene (a) is annotated with specific time values (has a duration, start, and end), and (b) contains participants (e.g., roles).

Chapter 4
Business Role-Object Specification

"When modeling complex systems, it is of great value to be able to draw on both the solutions found by others, and the analytical techniques used to construct those solutions." [162, p. 50]

Abstract The construction of a proper software system requires a well-specified blueprint. However, such a detailed blueprint is only possible in combination with sufficient and adequate semantics and syntax, combined in a powerfully expressive modeling language that can hold many conceptual and technical statements required for the software system construction. Currently, the established modeling languages, suitable for software construction, lack the possibility for using the role paradigm of using roles for reference model adaptation, as described in Section 2.2.1. Especially the non-invasive modification of an established (reference) model towards a new feature, specified as a process model, is often not possible in current modeling languages. Thus, in this chapter, the *Business Role-Object Specification* (BROS) modeling language is introduced, which handles both the creation and adaptation of software systems regarding the role paradigm. The BROS language is a tool intended for conducting a role-based adaptation, especially of reference models, which are often standardized and prescribed. Without the requirement of an adaptation, BROS can also be used for the development of models (or systems) from scratch.

4.1 Language Intention and Design Goals

The BROS language was developed especially for the adaptation of structural reference models. Current adaptation mechanisms are often invasive regarding the underlying structural model. Changes to the reference model in an ad-hoc manner are the mainly chosen approach of how to adapt and include new features. To overcome the current issues with object-oriented reference modeling (see Chapter 2), the specialized BROS modeling language was designed. The implementation of the new role-based adaptation method is explained in the next Chapter 5. The construction phase of the BROS language was inspired by some of the concepts proposed by Voelter [267]:

1. "A language, domain-specific or not, consist of the following main ingredients. The *concrete syntax* defines the notation with which users can express programs. [. . .] The *abstract syntax* is a data structure that can hold the semantically relevant information

> expressed by a program. It is typically a tree or a graph. It does not contain any details about the notation [. . .].” [p. 27]

Such, the developed BROS language describes both the abstract and the concrete syntax (see Sections 4.4 and 4.5).

2. > “DSLs often start simple, based on an initially limited understanding of the domain, but then grow more and more sophisticated over time, a phenomenon Hudak notes in his ’96 paper [referring [129], ed. Note].” [p. 26]

   The BROS language establishes a limited, basic set of modeling elements that are necessary to model the most common and important concepts. It is, however, able to be extended with new modeling elements in the future.

3. > “A *descriptive model* represents an existing system. It abstracts away some aspects and emphasizes others. It is usually used for discussion, communication and analysis. A *prescriptive model* is one that can be used to (automatically) construct the target system. It must be much more rigorous, formal, complete and consistent.” [p. 31]

   Since BROS is used for software construction, it is categorized as a prescriptive model. Nevertheless, because of its ability to specify adaptations easily, it could be used for analytic purposes as well (see Section 3.1.1).

4. > “[DSLs are] formal, tool-processable representations of specific aspects of software systems. We then use interpretation or code generation to transform those representations into executable code expressed in general-purpose programming languages” [p. 32]

   It is, at least theoretically, possible to transform BROS models into source code. The ability to do so is a matter of the right tooling that often does not exist.

5. > “[Small languages] have few, but very powerful language concepts that are highly orthogonal, and hence, composable.” [p. 35]

   On the one hand, the BROS language has only a few modeling concepts that are, on the other hand, highly dynamic and powerful mechanisms for specific adaptations.

To demonstrate the explicit intention of the BROS language, several major design goals are formulated: principles that guided the creation process of the language and the design decisions of single model elements and language concepts.

Non-invasive Changes

The creation of an application model by adaptation can be decoupled as far as possible from the underlying reference model and without modifying its structures.

For creating and maintaining software, it is crucial to adhere to design templates (and reference models) that are established and verified by domain experts (see Section 3.2). Modification of such predefined structures (i.e., adding, removing, and modifying model elements) indeed may lead to a faster, cheaper, and easier adaptation compared to non-invasive methods and forcing a separation-of-concerns. However, there are two known issues with invasive methods: (a) the software maintenance will get impaired as every individual modification has do be documented and may affect a lot of different parts of the software (that may not be known beforehand), and (b) future evolution steps of the underlying template (the reference model) are difficult to reproduce since once standardized elements are modified to individual solutions. A non-invasive adaptation method, like BROS, benefits from improved maintenance and evolution capabilities by avoiding both issues.

Maximum of Expressiveness

The number of available model elements is sufficient to cover as many model statements as possible.

Limitation and restriction of modeling elements is a useful mechanism for error prevention. However, unrestricted access benefits from more possible statements. While a limitation reduces the workload of the modeler, unrestricted access to possibilities implies that the modeler is responsible for not incorporating any errors. In turn, the modeler receives a much larger repertoire of possible statements, which can be implemented with the language.[1]

> "[. . .] conceptual modeling languages should be highly-expressive, even at the cost of sacrificing computational efficiency and tractability." [105, p. 8]

Therefore, the BROS language is designed as an expressive language with many different concepts and only a few limitations. As far as the modeler chooses, both cases can be used fine-grained specifications or only the most important elements (e.g., if there is a risk of incorrect or inconsistent application). Please note that the increased expressiveness also has impact on the technical and logical feasibility of constructed models, dependent on the quality of the modeling process. Further, despite BROS' focus on conceptual expressiveness, the language design still adheres to common conventions of understandable and usable syntax design (e.g., [179]).

From Design to Implementation

It is possible to construct a model that describes abstract business concepts as well as the corresponding technical specification for implementation.

Closing the gap between architectural description and implementation specification is a huge advantage in software development. Iterating refinement of the same model from the abstract level to the technical level provides an overview, decision basis, and confidence to potentially more stakeholders. It is useful both for the business side and for the developer side to successively enhance and improve defined model statements in a single model (so that generic requirements become implementable constructs)

[1] This use case can be compared with the "pointer" mechanism from various programming languages. On the one hand, the use of pointers may be beneficial regarding efficiency and performance. On the other hand, the wrong use of pointers may lead to crucial and non-transparent errors.

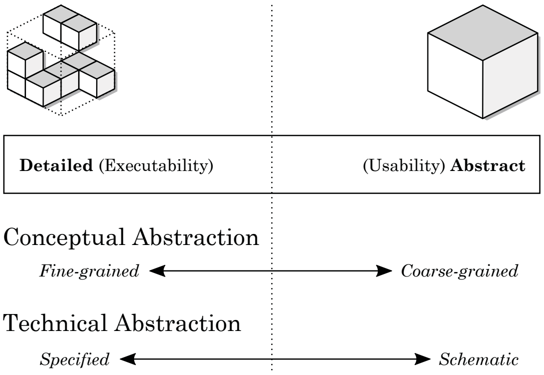


Fig. 4.1 Conceptual and technical abstraction (own illustration)

instead transforming different models into new ones at each level of the software development stack. Consequently, BROS allows the abstract definition of business requirements (e.g., business objects and tasks) as well as implementation-specific model statements (e.g., attributes and methods) within the technical range.

Further, BROS is intended to serve for conceptual abstraction as well. The possibility to model a software system in its basic parts and relations is needed and valuable for early discussions with all stakeholders. Thus, understandability is in the focus when designing the system. However, in the later phases of software development (e.g., when designing the system considering the more specific requirements), more detailed and fine-grained information is used to further define the structure and behavior of the system by using more and disassembled model concepts (that are technically refined in the later steps), shown in Figure 4.1.

Language Extensions and Features

The basic language is sufficient for describing models in the needed scope. For all other cases, the language can be extended with further language concepts.

In this thesis, it is not claimed to present an all-encompassing language that can handle all different kinds of issues. Instead, it is essential to support only a few (but powerful) modeling concepts that are necessary, according to point five from Voelter [268] stated above. The concepts currently introduced in the BROS language are sufficient to model the most common and necessary model statements. It is, however, possible and intended to extend the BROS language with *language extensions* that introduce new concepts for more domain-specific and appropriate adaptations. Such language extensions are similar to BPMN extensions and could consist of new modeling elements, entities, relationships, constraints, or related features. Thus, BROS is the foundation for new use cases and adaptations that might arise in the future while providing the necessary framework for new evolutions.

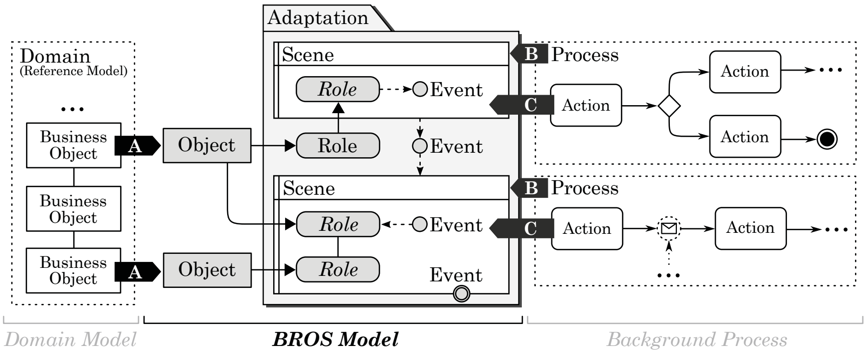


Fig. 4.2 BROS language concepts used for the adaptation approach: the domain model (as a reference model, left) and the background process (right) guiding the modeling of the BROS model as an application model (from [223])

4.2 Structural and Behavioral Information

The BROS language can be used to construct BROS models for software systems. In fact, there are two possible approaches to use the language: the "greenfield approach" and the "adaptation approach." While the former uses BROS to design models from scratch, the latter is used for reference model adaptation. In such, a predefined and standardized reference model is used as a template for an individual application model (see Chapter 2). Although the greenfield approach is beneficial in certain application areas, in this thesis, the focus is on the adaptation approach (see also Section 2.3). Nevertheless, the greenfield approach is still a feasible option.

For the adaptation approach, two components are required: the *domain model* (the structural information) and any *background process* (the behavior information), where the domain model should be a reference model. Figure 4.2 illustrates both sides. The final BROS model is then derived from both components and can be seen as the application model. The elements of the BROS model are explained later in Section 4.4 and 4.5. It is the intention of a BROS model to derive its content from both template sides (that is, structural and behavioral reference models). However, even without any template, the BROS model can be constructed from scratch.

4.2.1 Domain Reference Model

Several definitions are available for a domain model. For this thesis, the domain model is a term that generalizes any structural model that could be used for the adaptation approach to derive structural information about the (future) software system.

This definition includes two major points: (a) the representation is structural, and (b) the information about business knowledge is included. Both aspects are important for building BROS models with regard to domain models. On the one hand, the

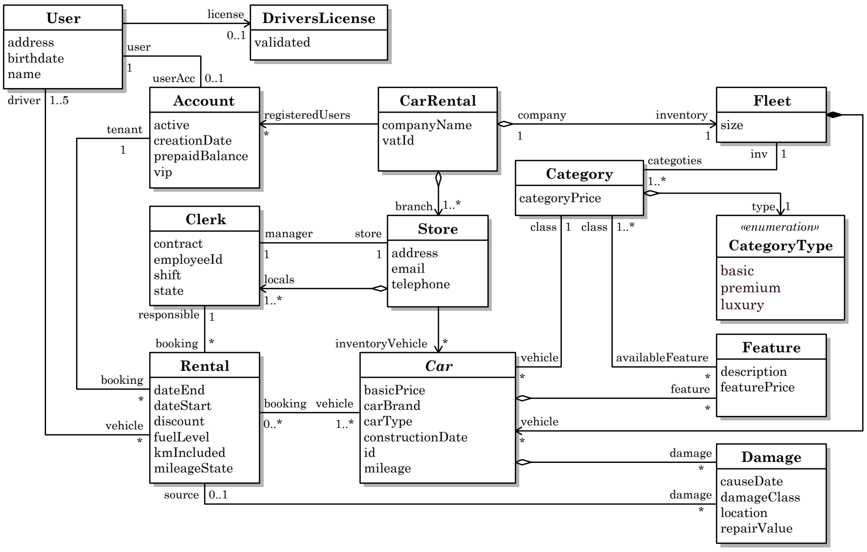


Fig. 4.3 An example structural reference model of a car rental domain

structure is needed for the actual software implementation, and on the other hand, the requirements need to be set as possible goals for the adaptation. A domain model can be used as a template for the BROS model design. Dependent on the degree of detail, domain model elements can be mapped towards BROS model elements.

Suppose the (structural) domain model is standardized and predefined (by, e.g., domain experts for the whole industry area), then the domain model is often used as a reference model (see Section 3.2), called a *domain reference model*. In this thesis, a reference model is a special type of domain model, including more technical depth, higher variability, and clearly defined relationships as well as (business) objects. This explicit specification of a domain's entities and relationships is useful for implementing a maintainable and extendable software system, as long as it does not get changed by its extensions. A reference model is the best possible template for a BROS model since BROS is developed to adapt such reference models in a flexible and non-invasive manner, to meet individual requirements. The given structure, especially the business objects, serve as a basis for the BROS objects (cf. Section 4.4 and 4.5), which then can be further adapted with roles and other BROS concepts (arrow "A" in Figure 4.2).

An example of a structural reference model is given in Figure 4.3. The domain of "car rental" is set by a reference model (type: UML class diagram), which serves as a common business knowledge base for similar "car rental management" software systems. Please note that this is only a small and fictional example of a reference model to demonstrate the purpose. Other "real" reference models[2][3] are much more complex and differ in their levels of detail and abstraction. The reference model in Figure 4.3 is technically detailed and consists of several business objects (like, e.g., `User`, Car, or `Damage`) as well as the relationships between them. The objective of this reference model is to identify the most important entities of the domain and to establish anchor points for a derived application model, but not to over-specify the domain in order to allow enough freedom for enterprise-specific adaptations. For example, from a software design perspective, it is not always suitable to implement a single business object called `Rental` that includes everything about the actual rent of a car. Instead, a company may decide for a different approach with various "booking packages" that are assigned to a car booking (e.g., weekend package, single-trip, extra holiday fees, unlimited included kilometers). In such a case, the enterprise must adapt the given reference model towards its specific business requirements and thus derives its own application model (see Chapter 5 for adaptation of reference models with the BROS language). The reference model, as part of the domain model side, serves as a template for structural information, i.e., the elements used for the BROS model. The definition of business objects and their interaction with each other is the main source of the BROS model semantics and syntax.

4.2.2 Background Process

New features of software systems are often initially defined by behavior specifications in terms of process models. The discrepancy between structurally defined systems and new incoming features as process models is an issue. But not only new features but also changed and evolved business processes can have an impact on the software structure as input. BROS solves this issue by including temporal information for its modeling language elements. The source of this temporal information can be either defined by the modeler (e.g., as part of the requirements specification) or derived from the upcoming process model or specification. For the latter case, the respective process model acts as a guideline for the modeled temporal information in the final BROS model and is called a *background process*. In contrast to a reference model, the background process is not necessarily a prerequisite for an adaptation or BROS model construction.

Definition 4.1 Background Process A business process, behavior specification, or (implemented) algorithm, defining the detailed execution of the actual program or system, which serves as a source of events in a Business Role-Object Specification model.

While the main structural elements are mainly derived from the domain model, the temporal information and execution flow is derived from the background process.

[2] `https://www.bian.org/`

[3] `https://www.ifla.org/publications/node/11412`

It is named that way for the reason that a BROS model is not intended to model a process flow but only takes the process as a point of reference to assign a point in time to the structural elements and their interactions, e.g., when a role is played (see arrow "C" in Figure 4.2). It is, in fact, the background to the temporal information at the structural side. Furthermore, since the process model is only intended to be the background, it does not matter of which type the background process is (e.g., BPMN, petri nets, event process chains (EPC), activity diagrams, informal text). As long as the background process is able to provide necessary information about the behavior of modeled structural elements, any type of process description can be used as the execution context (see arrow "B" in Figure 4.2). This universality is enhanced by a special structural model element called *event* (see Section 4.4.4), that denotes such points in time and could be based on a background process.

It is also possible to use multiple background processes for the same BROS model. However, it is useful to define and maintain reference labels (e.g., a key-value list or similar structures) for used elements from the individual background process and related structural elements in the BROS model. Any form of documentation of this transformation process helps for future adaptations. In general, the non-deterministic and liberal usage of a background process model leads to more responsibility but also more modeling freedom for the modeler.

Since the background process only acts as a guideline for temporal information, there is no strict transformation or mapping of the background process towards the BROS model (and this should not be the intention of a background process). Therefore, consistency checking is difficult. Nevertheless, in Section 6.1 and Section 6.2, some related rules and guidelines are stated to determine relevant mappings based on information stated by the background process. In addition, Appendix B.2 describes a tool-supported project related to ensuring consistency between a given background process and a BROS model. For that, a defined set of consistency rules are checked against a BROS model via model element comparison. As a result, warnings and errors based on found misconceptions are formulated and listed for the modeler.

Figure 4.4 shows an example of a background process, modeled as a BPMN diagram. The represented process is a trivial car rental process based on the two business objects of `Client` and `Car Rental`, who exchange messages and information to rent a car. This model specifies (illustratively) how to perform a possible new feature: to be able to book a camper for a long-term rental via a monthly token[4]. The idea behind this new feature is to rent a camper car only for whole months (as long-term rents) and renew the lease of the camper for the next month spontaneously. That way, the car rental can calculate the rentals of camper cars on a monthly basis (and make special offers to the client due to less maintenance). The lease token contains the monthly fees as well as all necessary insurances for the camper (e.g., baggage insurance for foreign countries). All used tokens are paid when returning the car.

In reality, business processes are much more complex and contain a variation of different modeling elements. An overview of the BPMN usage and its modeling possibilities is given in Section 3.1.3 as well as in Allweyer [7]. Additionally to that

[4] This BPMN background process is further used in the Chapter 5 for an example BROS adaptation.

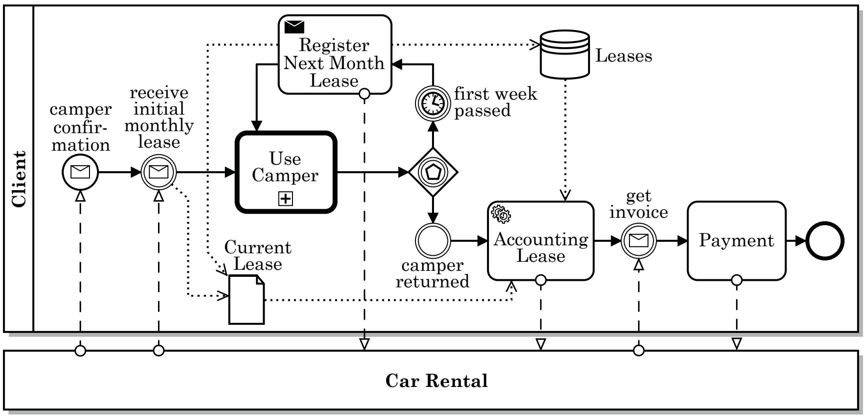


Fig. 4.4 An example process model of a long-term car rental process (in BPMN)

use case of a long-term rent, other features' BPMNs could be used for background processes as well: online booking, VIP program, check-in via app, etc.

4.3 Metamodel

The BROS language offers various modeling elements to express modeling statements as a BROS model. In general, such a BROS model is (in the first place) independent from any other requirement or model. However, in the process of reference modeling, the domain model and related background processes are used to determine the right choice of elements as well as their usage at design time when creating a BROS model.

The set of BROS model elements is summarized in a metamodel, a special type of model that denotes the nature of the (BROS model's) elements used. Metamodels are often considered to be models about models, which establish the modeling rules between all the elements in the actual model.

> "The prefix 'meta' is used before some operation *f* in order to denote that it was applied twice. Instead of stating '*f*–*f*', as in 'class–class' one states 'meta-*f*', e.g., 'meta–class'. For any further application of the operation, another 'meta' prefix is added to yield 'meta–meta-class', etc." [153, p. 377].

According to Kühne [153], there can be two metamodel instantiating types: ontological (instantiation of the element's meaning and content) and linguistic (instantiation of the element's form and shape). BROS metamodel instantiations are considered to be of the former kind. Metamodels are an important tool for software engineering since they are able to abstract the used models and guide the development of new ones simultaneously [96]. Especially, the metamodel notation *Meta Object Facility* (MOF) [186] is a popular approach to defining languages in syntax and semantics. It defines three levels of a model's instance, which are also used by BROS:

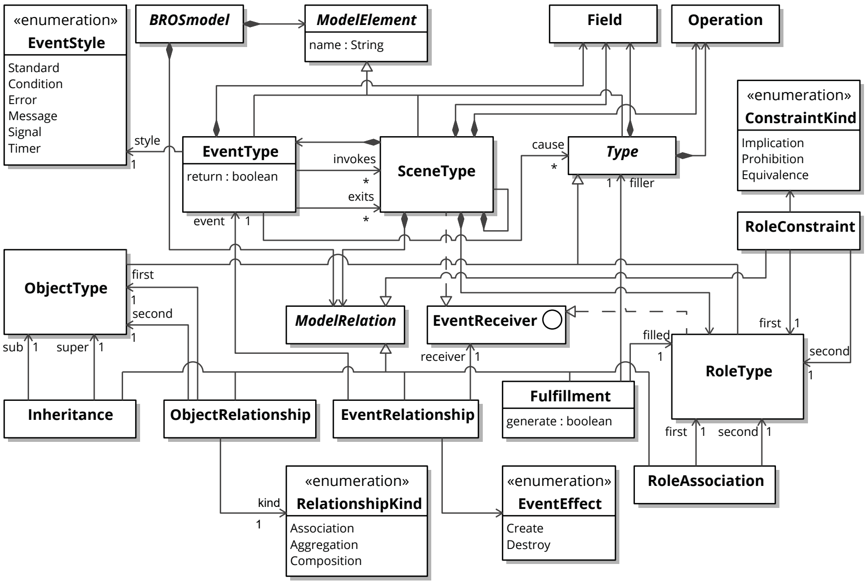


Fig. 4.5 The BROS modeling language metamodel (adapted from [223])

M2 the description (as entities and relationships) of the M1 model element types
M1 the actual representation of the concrete system
M0 runtime objects that are instances of M1's defined types

For BROS, the metamodel is in the stable state, that is, only those model elements are implemented, which are required for the most important concepts (called "core concepts") in order to construct valid BROS models. Nevertheless, model elements of other concepts (called "extended concepts") may not be represented in this metamodel. The BROS metamodel is shown in Figure 4.5. Its content (the individual BROS elements and their relations) is not explained on the basis of the metamodel itself but treated separately in the following Sections 4.4 and 4.5. Also, both sections mainly cover the semantics of the BROS language concepts; the details of style and graphical usage are further explained in Chapter 6.

4.4 Core Concepts

Core concepts in the BROS language are the major "things" that can be specified in a creation or adaptation process. They are connected via relationships and are representing structural nodes in the software system. Core concepts are tested and applied in use cases, which is why they are represented in the basic metamodel of BROS. In BROS, the following concepts can be considered as core concepts:

Table 4.1 Example domain models that can be used to derive BROS objects

Domain Model Type	Object Template Element
UML Class Diagram	Classes
Business Object Model	Business Objects
Entity-Relationship Model (ER)	Entities (only partially suitable)
UML Use Case Diagram	Actors
UML Component Diagram	Logical Components
Web Ontology Language (OWL)	Class
Prosa Text Requirements	Nouns (words in text)

- Objects
- Roles
- Scenes
- Events

These concepts might be combined with extended concepts (see Section 4.5), however, this could lead to inconsistent model statements.

Every core concept has its own syntax, semantics, inner elements, and relationships, which are described below. Please note that the BROS concepts are rather few but powerful mechanisms that may be used in many different ways to express a large set of model statements (with respect to the targeted language concept "few, but very powerful language concepts" described in Section 4.1).

4.4.1 Objects

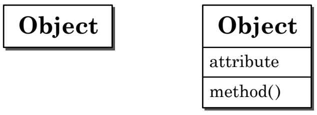
The *objects* are the central modeling elements within a BROS model. They carry the meaning and identity of the business objects. They come with their initial and individual behavior and state (that is, for implementation purposes, fields and operations in the metamodel).

Definition 4.2 Object The central modeling construct for representing business objects within a given business domain. The object has its own identity and is uniquely identifiable among all other objects. It contains all necessary information to represent the business object with its universally valid properties, behavior, and relationships within the system.

Semantics

If a domain model is used for the construction of the BROS model, the objects can be retrieved from the domain model. Dependent on the type of the given domain model, one could use various elements as objects. Some examples of which elements

Fig. 4.6 The object model element in short (left) and complete (right) view



from a domain model can be used as objects are given in Table 4.1. In fact, most structural model types offer some kind of entity element (e.g., in the form of actors, participants, node, package, etc.) and can, therefore, be used as templates.

If, in addition, the domain model is used as a reference model, the extraction of an object out of a reference model should not be mixed up with any duplication or creation of new entities. The reference model should be fixed and non-changeable[5]. The object, in case of reference model adaptation, functions as a representation of the underlying entity in the domain reference model. One can divide this situation in two possible ways:

1. the reference model is detailed enough to use its elements for objects, or
2. the reference model is under-specified and the objects need to be enhanced in a way that they can represent the given entity of the reference model.

For the first case (e.g., the reference model is a UML class diagram), the creation of BROS objects is an easy task since the objects can make use of all the given details, especially fields, operations, and relationships. However, for the second case (e.g., the reference model is a UML use case diagram), the objects need to be designed as appropriate representatives regarding the (later used) implementation details. In general, the more details a reference model has with respect to its entities, the better the objects can be derived from it. After all, a BROS model (which includes the objects) should ultimately have a certain technical complexity in order to be suitable for software design.

Please note that the object is simply the single object, while a *compound object* is the set of the object, including its relationships and, more importantly, its roles [159] (especially at runtime). There is no separate notation, however, it could be necessary to express several statements.

Syntax

As a graphical shape, the object is represented by a single rectangle (see Figure 4.6). The object could make use of its signature, i.e., fields and operations, in separated sections. In general, an object behaves like regular classes (like, e.g., in *Java*[6]).

> "The reason why [it is not called] class but object is simply due to stress that objects always represent a real business object and not any technical class concept [. . .]" [223, p. 249]

Two objects may be connected via relationships, which denote the interaction between two objects. For shortcuts, the object can be written in the short notation (name only).

[5] For that, BROS can be used for non-invasive adaptation (see Section 4.1).

[6] `https://www.w3schools.com/java/java_classes.asp`

This is often used for the first, non-detailed sketch of the system's components (i.e., focusing usability regarding Chapter 4.1 and Figure 4.1).

Inner Elements

Fields

As a meta-representative of commonly known "attributes," the *field* concept is part of the objects "signature" and used as state and structure description of an object. Like in the majority of programming languages, fields are typed and named variables that are assigned a value at runtime (e.g., `price`). The BROS language does not determine the exact way of how to state a field since BROS should be applicable for implementation (thus, it depends heavily on the chosen programming language). For, e.g., a UML-like notation, an attribute field may have default values, limitations, multiple structures, and more. Dependent on the chosen programming language, the fields (that is, typed and named variables) are written in the fields section of the object. The example in Figure 4.7 illustrates an object (in a complete view) with specified `name` and `age` of a `Customer` object, while `age` is assigned the default value of `21`, and the value limitations of 17 up to 90 (written as multiplicity `17..90`).

Operations

Usually known as "methods", the concept of *operations* is the basic expression for an object's behavior and a part of the object's signature as well. Operations denote the possible actions and activities that can be executed by an object, e.g., `rentCar()`. This follows the general notation of methods in various programming languages. Please note that the brackets after the name are an indicator that distinguishes an operation from a field. Nevertheless, just like fields, the BROS language does not enforce the notation of an operation, too, in order to handle multiple underlying programming languages with different notations of operations. However, return value type and typed and named values as a parameter are often part of the operation notation (e.g., `rent(car:String, duration:Time):int`). A graphical example of an object containing fields and operations is given in Figure 4.7 and follows the common UML modeling style of classes (see, e.g., [11, p. 49]).

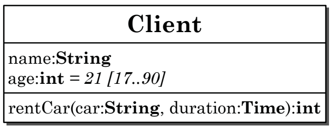


Fig. 4.7 The object with exemplified fields (attributes) and operations (methods) in common UML notation

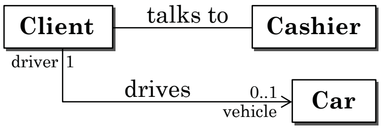


Fig. 4.8 Three objects with simple and detailed associations

Relationships

Association

As stated above, an object may use an *association* to interact with another model element. An association is graphically represented by a single connecting line and possible arrow heads at its ends. The association is the most general relationship between two model elements since it is simply stated "the usage and knowledge of each other." Thus, the interaction of two elements can be translated as "using method calls" or "pushing information" between each other. If the association is unidirectional, the communication is restricted towards one of the two sides: the access in the opposite heading direction of the arrow is not allowed or established. Further, besides its *name*, the association has *multiplicities* and *association ends*: the multiplicities are used to specify the number of certain entities participating in this relationship, and association ends (sometimes known as roles) name the association ends. For both, the notation follows the general UML notation. The details of these features are given in, e.g., Ambler [11]. Figure 4.8 illustrates two associations: the first one `talks to` as simple notation (only name and target) and bidirectional, and a second one `drives` with more details (additional association ends and multiplicities) and unidirectional. However, in general, the name of an association in BROS might not be unique, but the signature of the whole association statement should only be defined once.

Aggregation

Aggregations are relationships that represent the "is-part-of" something else, often another object. Usually, aggregations are used to state that two entities are not only known by each other but that one entity contains the other as part of itself (e.g., a `Team` might aggregate several `Members`). In fact, the containment is realized by a reference, such that multiple (runtime) entities may have the containing entity. The containing entity never "really belongs" to the owner but may also exist on its own. The aggregation is graphically represented by an association combined with an (empty) diamond at the end of the containing entity (as shown at the top in Figure 4.9, top). Further, it consists of association ends, multiplicities, and a name (like the association). The naming of an association is difficult, as there are limited possibilities that denote the containing nature (e.g., `has`, `includes`). Thus, association names are often omitted.

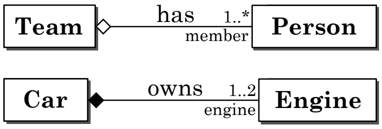


Fig. 4.9 An aggregation (top) and a composition (bottom)

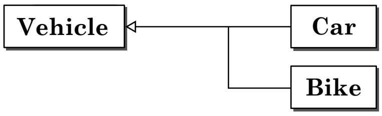


Fig. 4.10 Two objects inherit from another object via inheritance

Composition

Composition has a filled diamond as its arrow (see Figure 4.9, bottom). Although the composition is a "part-of" relationship as well, its intention is different and much stronger than an aggregation. The contained parts of the composition relationship are not only belonging to the containing entity but are its vital inner components (e.g., a `Car` might compose an `Engine`). If the containing entity is gone, all its parts are ending as well (or do not make sense as standalone objects, e.g., the `Fox` that composes the related `Paws`). There is a subtle borderline between aggregation and composition. In doubt, it is recommended to use an aggregation over composition since it is less restrictive (or leave out the aggregation [156]).

Inheritance

Generalization and specialization are made via *inheritance* relationships. This is one of the most important concepts of object-oriented modeling. If an object inherits from another object (it is "specialized"), it inherits all structure and behavior from its superior object (the inherited object) and is also (sub-)type of that superior object. Further, it could contain specialized functionality (structure or behavior) that is not present in the superior object. An example would be `Animal` as a superior object of the `Fox` object, which inherits all structure and behavior of `Animal` but specifies some of its structure and behavior as specialization. The inheritance arrow is drawn as an association with an (empty) triangle arrowhead in the direction of the superior object (see Figure 4.10). Further details and explanations can be found in the official UML documentation [187].

Fulfillment

As inheritance is essential for object-oriented modeling, *fulfillment* is the most important relationship concept for role-based modeling (thus, for BROS as well). It determines the relevant player that takes over the role at runtime.

Definition 4.3 Fulfillment The relationship that assigns a role to an object.

This definition correlates with the definition of a role (cf. Section 3.3.1). A role is always "fulfilled" by an object (or other roles under certain circumstances). A

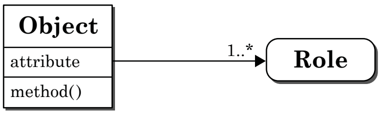


Fig. 4.11 An object fulfilling a role

distinction is made between design time and runtime: the former case is called “fulfillment,” whereas the latter case is called “plays.” This is used to express whether the role binding is used for design purposes (i.e., the role may be played by the fulfilling objects during its lifetime) or for runtime issues (i.e., the role is actually played by an object as a “player”).

The fulfillment, the ability to play a role at runtime, is the mechanism to adapt a (given) object in the favored manner. By fulfillment, the object can adapt its structure and behavior (that is, fields and operations) at runtime. An object may fulfill several roles simultaneously, as well the same role several times (each with its own fulfillment relationship). Figure 4.11 shows a fulfillment arrow from the `Object` towards the `Role`, i.e., the object gets adapted with the role. Considering the adaptation, the object overtakes the role’s inner elements and relationships. There are no names for fulfillment relationships (except maybe an internal ID), however, there could be a multiplicity to specify how often the object may play a role in the runtime application.

On the one hand, other elements can “see” and interact with the `Role` as it would be an object. On the other hand, the `Object` can represent itself with the `Role` as a mask and interface to others. In such a way, adaptation is achieved in a non-invasive way. In BROS, there is the additional constraint that other entities are not allowed to access the role’s underlying object directly since independence and transparency of the adaptation and the reference model should be kept. The access to an object in the adapted application model is only possible through roles that are fulfilled by the object (due to the representative policy of BROS).

In general, a fulfillment could be made from one role to another, called a *deep role*, which is explained in the role’s fulfillment section. However, this is often the exception (and has a different meaning), and usually, the fulfillment is only made between an object and a role. Thus, if not stated otherwise, one implies a fulfillment as always between an object and a role.

4.4.2 Roles

The *role* concept is the main modeling and adaptation mechanism of BROS. It serves as a characteristic placeholder, i.e., a representative of a business object (i.e., the objects) from the domain model side, and is used as an object’s participation for various behavior specifications (i.e., a process). The conceptual background of a role is already described in Section 3.3.1, thus, this section introduces the technical terms of using the role concept for adaptation with BROS. For that, the role definition from Section 3.3.1 is used.

Table 4.2 Example process models that can be used to derive BROS roles

Process Model Type	Role Template Element
BPMN	Swimlanes
UML Sequence Diagram	Lifelines
UML Communication Diagram	Objects, Parts
UML Use Case Diagram	Actors
Prosa Text Process Description	Subjects (words in text)

Semantics

The role concept is the major adaptation mechanism of an underlying reference model. Its proposed fields and operations are used to override the basic entities of the reference model for two purposes:

1. to reflect the reference model's static entities in new contexts, and
2. to execute individual behavior requirements from the business side.

Roles are the connection between the domain model and the background process since they are able to abstract and refine any given object in its usage and interaction with other entities (see also Section 2.3).

Thus, the role is fulfilled by objects (in rare cases also by other roles). Such, the role adapts and modifies the underlying object with its fields and operations. For that, the role has its own fields and operations specified in the same way as regular objects (see *fields* and *operations* of objects). The usage of a role is similar to the usage of an object, except for two differences:

1. a role is not given by the reference model but designed by the modeler (with respect to the background process), and
2. a role can not be used without fulfillment (thus, making its specification incomplete since it depends on the fulfilling object's specification and identity).

Although the role is dependent on the intention of the modeler, there are several hints in the background process that allow the introduction and specification of a BROS model's roles. Some examples are listed in Table 4.2. Please note that the examples are only hints and not given by any rule. It is the freedom of the modeler and the adaptation intentions to select sufficient roles. Further, it can be observed that some examples are similar to the examples of objects (in Table 4.1), e.g., actors of UML use case diagrams. This is due to the fact that current models do not make use of a role concept. Thus, the modeler should decide whether an entity is an object or a role, also based on the given reference domain model.

Both, roles and objects, may have a lifetime at runtime. However, in contrast to an object, a role, i.e., its fulfillment, is supposed to be limited time-wise and of temporal nature [239]. The exact point in time, when a role starts and ends, can be modeled within BROS (e.g., via *events* or *scenes*). A role that is affected by such temporal events (and is, therefore, limited in its fulfillment by the object) is considered to be a *dynamic role*. It can be assumed that every role is of temporal nature with respect to

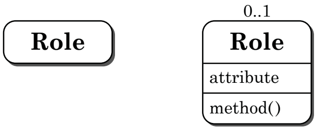


Fig. 4.12 The role model element in short (left) and complete (right) view

the idea and intention of the role concept (as a representative of an object). However, in BROS, the modeler is in charge of such specifications. If a role does not get a specific lifetime assigned but holds for the whole use case, it is called a *static role*. Static roles can be useful for simple state expressions or placeholders for objects within an adaptation (e.g., the static role `Cash` can be an appropriate adaptation of the object `Payment`).

Syntax

A role is represented by a rectangular shape with explicitly rounded corners (see Figure 4.12). It has a section for the *name*, the used fields, and operations. For shortcut representations, the role can be collapsed and only be shown with its name. The role can have several relationships, most importantly the incoming fulfillment of an object. Further, the role itself can have a *multiplicity*, written above the role, that states a specific amount of the role's instances in a certain context that can be valid simultaneously. Please note that this is different from the fulfillment multiplicity, which considers the role's instances per fulfillment relationship.

If a role is dependent on temporal effects (e.g., within a *scene* or affected by an *event*), the name is written in italics and used as a dynamic role. If, in contrast, the role name is in regular shape, it is not affected by temporal conditions and counts as a static role.

Inner Elements

Fields

A role consists of *fields* that denote the state of a role. However, it is not the role's state but the adaptation of the state of its fulfilling object. In general, the fields are similar to fields of objects and use the same syntax. From an external view, the field of a role is not distinguishable from the field of an object. This similarity is intended and due to the fact that a role should be used in a representative manner of the underlying object. When fulfilling a role, the fulfilling object takes over the role's fields in an additive manner and can make use of them. Further, the fields of a role can use *indexes* to denote the adaptation behavior of the fields compared to the original fields of the object.

Operations

The role's *operations* are similar to the object's operations for similarity purposes. The role's operations state the behavior of it. More precisely, the operations are stating extended and adapted behavior of the fulfilling object. This is used for behavior adaptation of the underlying reference model objects. Like fields, the operations are used additive towards the fulfilling object in that way that the object can execute the role's operations (so-called "dispatching"). The operations can make use of *indexes* as well if there is any need during the additive merging.

Indexes

By using a role, the fulfilling object gets adapted towards a specific use case. The role individualizes a generic object with its application. The roles do not have any individual identity but always using the proposed identity by the playing object. However, they can define individual state and behavior (that is, fields and operations in the metamodel). In contrast to the object's state and behavior (which is fixed all the time) the role operates additive: the role's state and behavior are added towards the fulfilling object, and the object can use the new role's fields and operations like they are its own.

For adaptation purposes, the role's inner elements can make use of *indexes* that show the difference and handling of the inner elements of the basic reference model object. In BROS, there are three special *indexes* (marked by `/`, `:`, and `#`) that can be annotated in front of a field or an operation. The indexes are used for cases in which a field or operation is not simply added towards the fulfilling object but handled differently. The special indexes are used in the following ways:

- `/` **Escape Index**: This index blocks the use of a specific field of the fulfilling element. Even if the field or the operation is available by the player, the usage of the same field or operation via the role is prohibited and returns `escaped` error. Adaptations utilizing the escape index is often intended to block the access to crucial functions of the player for, e.g., role-based access functionality.
- `:` **Conditional Index**: The conditional index is used to determine the existence of a player's field or operation and acting accordingly. The standard case (single colon) is used to overwrite a particular field or operation: if (and only if) there is an existing field or operation provided by the player, then overwrite it with the following field or operation. This index is often used when there are multiple objects available that fulfill the role. Since this is a rather simple case, brackets (`:[<condition>]{<action>}`) can be used to describe more complex relations with the base player[7], e.g., via logical expressions using Boolean operators or temporal logic. Further, it can be of use for unknown future players.
- `#` **Purge Index**: Instead of escaping every field and operation of a player, the purge index can be used. It is a single character stated at the beginning of a role's inner specification and prohibits the usage of all the player's once defined fields and

[7] This is not part of this thesis but introduced as an interface available for future research and application.

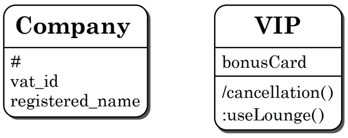


Fig. 4.13 Roles with a purge index (#, left) and regular indexing (right)

operations. This is intended to wipe the whole foundation and starting from scratch with the adaptation. However, this should not be used carelessly since this is contrary to the idea of adaptation (i.e., using the role as a representative of an object and not as a replacement of such). Using the purge index is similar to "modify one object to a completely different one", which is not recommended. When used, all other fields and operators do not use any other index (since the original object's signature is wiped).

The indexes are exemplified in Figure 4.13. The `Company` role uses the purge index to wipe the signature of any fulfilling object and replace it with its own signature (that is, `vat_id` and `registered_name`, which are stated without indexes themselves consequently). Any third party element can make use of this, however, the `Company` does not transfer any original field and operation as adaptation. Such, the `Company` completely redefines a fulfilling object into a "new one", which is not recommended for reference model adaptation. In contrast, the role `VIP` adapts the `Client` in a proper way. The `bonusCard` is simply added, the access of `cancellation()` is blocked (maybe due to cancellation restrictions of the contract), and `useLounge()` overrides existing similar operations, if existing (e.g., due to more lounge possibilities for VIP's than normal clients).

It is important to remark the fact that the index specified fields and operations are always valid for all possible fulfilling entities. Otherwise, different roles must be used. For the case that nothing is subject to change when using the role, the role specification is empty. Then, the role acts as a simple representation of the original fulfilling object. When using roles for adaptation, it is recommended to use a role for every object involved, which makes empty roles necessary.

Relationships

Association

The *association* of a role is similar to other associations of the UML and denotes the interaction of a role with another role. The role association is used to express the interaction (e.g., method call) of participating roles and is drawn like a simple line. The role association has a *name* and *multiplicities*, however, without association end labels. In general, it can be both unidirectional or bidirectional.

It is not allowed to use the role's association between roles and other entities since this is purely a role-based connection, and the adaptation should be as separated as possible. For adaptation purposes, it might be helpful to state the nature of the association between two roles (e.g., containing, pushing information, calls). Especially

Table 4.3 Role constraints in BROS

Name	Icon	Description (behavior at runtime)
Implication	—▷	playing role `A` implies the player to play role `B` as well
Equivalence	◁—▷	role `A` and role `B` can only be played simultaneously
Prohibition	├—┤	playing role `A` does not allow playing role `B` simultaneously

when two already connected objects getting adapted with two roles, a resulting role association can be more specified, dependent on the original relationship. There is no explicit aggregation between roles since, usually, the technical base relies on the reference model, which already includes aggregation and composition. Nevertheless, the containing nature can be used by a small indicator (filled or empty diamond) beside the association's name.

Fulfillment

The concept of the role *fulfillment* is a bit different than the fulfillment of an object towards a role (i.e., the object fulfillment). If a role is fulfilled by another role, then it is called a *deep role*. Even though deep roles are possible in several role-based modeling languages [151], deep roles are semantically difficult since there is no real definition of how a role can play another role[8].

In BROS, deep roles are allowed per definition. Thus, roles can use a fulfillment to play other roles. This decision was made due to the language design decision of the "maximum of expressiveness" and "modeler's freedom". However, it is not recommended to use deep roles as the semantics of a deep role is not well defined. In this thesis, if a role plays another role, it is defined as follows:

Definition 4.4 Role Fulfillment An object's instance is able to play a deep role without an own fulfillment towards the deep role if, and only if, the same object already fulfills another role that fulfills the deep role.

With this definition, the semantics of a deep role is adapted towards object instance shifting and "identity pass-through" at runtime rather than "allowing a role to play another role". Additionally, like the standard fulfillment, the role fulfillment has (besides its *name*) a *multiplicity* to state how often the target role can be played.

Role Constraints

Roles can use *role constraints* at design time to express certain regulations and dynamics between them. At runtime, the constraints are used to restrict or guide the behavior and interrelationships between the roles as part of the targeted adaptation.

In BROS, there are three major constraints that are established in current literature, especially introduced by Riehle and Gross [209], shown in Table 4.3 and in

[8] This is different to the statement that an object plays a deep role "through" another role.

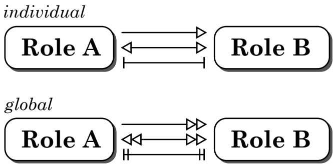


Fig. 4.14 Roles with the three constraints implication, equivalence, and prohibition on the individual (top) or global level (bottom)

Figure 4.14 as well. They are always bidirectional and propose modeling statements that are affecting the runtime and playing of roles by an entity. The constraints must (only) be enforced at runtime and not at design time, since, at design time, an object can definitely fulfill several roles that are restricted at runtime. Thus, constraints are a matter of runtime role relationships. Nevertheless, a constraint is established between runtime roles, not runtime objects. There is a slight difference considering the dependency of the player (an object's instance that plays the role at runtime). On the one hand, a constraint between roles can include the assumption that the player of both constrained roles is the same. The constraint is then called as *individual constraint*. On the other hand, it is possible to establish a constraint between roles independent of the fact which object's instance is actually playing these roles. Such independent constraints are then called *global constraints* because they are valid for the whole model context and every possible player. If nothing else is stated, a constraint is always on the individual level. For the global level, the constraint gets an additional arrowhead in its graphical representation (like shown in Figure 4.14, bottom) and the keyword "global" is mentioned before the constraint[9]. It is not intended to use multiplicities for constraints. For further specifications, e.g., a side note has to be used.

The *implication* is used to denote a relationship between two roles, in which a role needs another role being played. Like the nature of an implication, the implied role can be played without the implication role as well. However, if the implication role is played, the implied role has to be played as well. The player depends on the kind of constraint level: if an individual constraint is used, the constraint is restricted to the same player instance. On the global level (two arrowheads, see Figure 4.14), the constraint is independent of any player instance. Usually, the implication constraint is used to denote that one role is "kind of" another (like inheritance) or further specialization of other roles. For example, the `VIP` role could imply by the `Client` role, if the VIP requires to be a client in the first place. The client, however, may exist without the VIP.

The *equivalence* is used to state the similarity between two roles. A role that has an equivalence to another role can not be played without the other role being played as well. Thus, this is a bidirectional implication and enforce a stronger combination of both roles. This constraint can either be used as individual or global. Often, this constraint is not used for a specialization of another role (like implication) but to establish a connection to a role in another context or with another dimension of intent.

[9] The instance handling can also be partially represented as a constraint between the fulfillment arrows. However, arrows between arrows are not considered a good modeling style.

Table 4.4 Example process models that can be used to derive BROS scenes

Process Model Type	Scene Template Element
BPMN	Process
UML Sequence Diagram	the whole diagram
UML Use Case Diagram	Use Case
Event-driven Process Chain (EPC)	Process
Petri Net	the whole diagram
OO-based Source Code	a method or script
Prosa Text Process Description	the whole process description

For example, the role `VIP` may use the equivalence to another role `Bonus Card` to state that a VIP always needs to have a bonus card and vice versa.

The third constraint is the *prohibition*, which is the exact opposite of the equivalence. If two roles have a prohibition constraint between them, it is not possible to play both of them at the same time. One may use the individual prohibition or global prohibition to generalize the constraint. Usually, the prohibition is not used for all roles that are "not possible" to be played together but roles that are "not allowed" to be played together (while it would, technically, still be possible, though). For example, the role `Cashier` may not be allowed to be played with the role `VIP` simultaneously since employees might not be able to sign up for the bonus program of a VIP.

4.4.3 Scenes

Structural modeling often suffers from the lack of the abilities to represent temporal information (regarding the structure). Time frames, limits, or specific points in time are hardly possible in established structural languages, like, e.g., UML. Via workarounds (e.g., with UML profiles), this functionality can be used to extend the static and structural language. Nevertheless, this new "behavior specification" with temporal information must be synchronized with the goal of why someone might want to introduce temporal information into the structure: to represent and model the needed requirements.

In BROS, the concept of *scenes* is introduced. Scenes are the objectified entity that denotes a process, chain of actions, or simply a time frame with a defined start and endpoint in time. It is used to be a context for the introduced roles.

Definition 4.5 Scene An instantiated, temporary collaboration context of roles and events that are related to the same business logic part.

Semantics

A context can be both static and dynamic. Both states are already specified for roles (i.e., static and dynamic roles); it is equivalent for context: a *static context* states a collaboration for entities that are not dependent on time but will hold forever, whereas the scene as a *dynamic context* is a time-dependent collaboration. The role-based modeling language CROM introduces static context as *compartments* [149, 151].

A dynamic context is fundamentally different since the participants (the roles) are not universally valid but dependent on temporal information and lifetimes [239]. BROS introduces the *scene* as the dynamic context as defined by Almeida et al. [8]:

> "[. . .] a scene is not static, and involves a (temporal) succession of situations and occurrences involving the objects in the scene." [8, p. 29]

Although compartments can be used in BROS, they are not part of the core set of the language concepts but contained in the extended concepts (see Section 4.5.2). Instead, the scene is the major context mechanism and allows the temporal collaboration of roles based on business logic.

The scene is a context that is objectified at runtime. Context is, as understood in this thesis, meta-information that defines the nature of used roles and their relationships. Using the definition of context from Dey [64] (see Section 3.3.1) a scene is interpreted in such a way that

> "[. . .] a scene (and the model itself) is a particular *information* (the *context*) that characterizes the situation of objects (the *entities*) whereas the objects then are relevant for the general interaction within the scene, expressed as roles as context participants." [223, p. 249]

Thus, the scene acts as a surrounding, temporal collaboration that defines both the individual temporal time frame as well as the roles within that frame. The start and end of a scene are usually defined by events (init, return, and exit events, cf. Section 4.4.4). The roles are (per definition) the representatives of objects that interact with each other in that (and only that) specific context. In general, the scene is representing a complete temporal time frame. Thus, it can be derived and motivated by the behavioral specification of the system, i.e., process models. Some examples, how a scene can be derived from process models, are listed in Table 4.4. Since the scene is an entity, there are instances of the scene at runtime, just like any other entity concept. Multiple instances of the same scene can exist and interact with each other. The scene in BROS corresponds to the two properties, specified by Guarino and Guizzardi [104], that a scene should use as a foundational semantic:

> "The important facts are:
> 1. A scene can not be instantaneous: it always has a time duration bound to the intrinsic time granularity and temporal integration mechanisms of the perception system considered;
> 2. A scene is located in a convex region of spacetime. It occurs in a certain place, during a continuous interval of time." [104, p. 245]

Roles that are not used in such temporal collaborations (static roles) are used outside of scenes, making compartments less needed and scenes technically a refinement of the compartments.

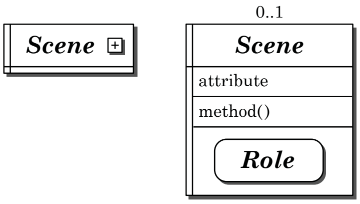


Fig. 4.15 The BROS scene in its collapsed state (left) and full representation (right)

The scene's identity is given by its behavioral nature (e.g, `Transaction`). In BROS, there are two ways on how a scene can be defined to specify certain behavior:

1. The *process-like* scene is used to specify the context as a process or chain of actions. The roles and events related to this kind of scene are usually participants in an exchange-like behavior, and the start and end is a defined action of the related process (e.g., `VIP Subscribing` or `VIP Cancellation`). Further, the background process (see Section 4.2.2) is tightly connected to the scene, and the content is transferable more easily (e.g., a BPMN diagram).
2. The *state-like* scene is different from a process-like scene in such a way that the identity is not based on a concrete and deterministic process (e.g., described as a background process) but frames the time of certain roles in the manner of states. Per definition, a scene has a start and endpoint in time, which holds true for the state-like scenes. However, they are not considered as a chain of actions but a specific state in the system that starts and ends with certain events (e.g., `VIP Membership`). The related background process has to be more abstract to be able to transfer its content to the BROS model and this kind of scene (e.g., a requirement specification).

Although some other works establish more fine-grained ontology terms regarding the scene (e.g., the scene could be composed of *situations* [8]), the BROS scene does not make use of such concepts (yet). However, due to the possibility of extending the BROS language, such new concepts are able to implement in the future, if needed.

Syntax

A scene is represented as a box with a crossed line at the left side, as shown in Figure 4.15. There is a *name* as an identifier, which is shown in both states, collapsed and fully represented. In the latter state, there is a section for fields and operations, the scene's own state, and behavior. Within its body, the scene contains its enclosed roles and events. Following the notation for dynamic roles, scenes (which are dynamic contexts) are written in italics (thus, making all inside roles dynamic as well). When collapsed, a small square with a plus sign is shown next to the scene's name. If needed, the scene can have a *multiplicity* on top to define the possible amount of runtime instances of the scene.

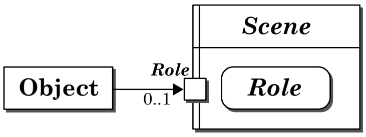


Fig. 4.16 An object fulfilling a role within a scene via port access

Inner Elements

Fields

For adaptation purposes, the scene can make use of *fields* to state its own state (e.g., `transaction id`). Since the scene is an entity itself, it has a runtime instance that gets values assigned for its defined fields. Technically, a scene is a grouping object that can be called for its fields (and operations). The used fields are kind of meta-data for the inner roles in that scene and can carry important information about the context of a current role instance. There is no exact place where one might extract the fields of a scene, neither in the domain model nor in the background process. Nevertheless, fields might represent certain business requirements or use cases that are stated (e.g., "Every transaction has a transaction id."). Figure 4.15 (right side) shows a scene with a field.

Operations

Unlike fields, the *operations* of a scene are rather rare since the scene acts as a process or behavior itself, making the scene's operations often redundant or even conceptually misplaced. However, due to the paradigm of the modeler's freedom, the scene is still able to execute behavior in terms of operations in case it is needed. For that, scene operations are defined (see Figure 4.15, right side). Possible examples for operations as a context behavior are aggregations (e.g., the number of active role instances) or summarize functionality (e.g., the state of a certain event). The execution of business logic should be avoided and done by the roles within the scene context.

Roles

The dynamic *roles* within a scene are considered to be the heart of a scene. The roles are an adaptation of established (business) objects and act as representatives. The idea of a scene as a process of (business) behavior requires actors that are able to execute the behavior. Roles are used as the participants in that scene, also stated by ALMEIDA ET AL. [8]

> "The scene is characterized (at a particular point in time) by a number of participants, which it inherits from the rigid embodiment that is manifested at that point in time." [8, p. 31]

However, in contrast to ALMEIDA ET AL. [8], the scene in BROS is characterized but not defined by its inner roles. Figure 4.16 shows an object fulfilling a role inside a scene. For this, it uses a *port* to state that the target is inside the scene, not the scene itself. The port is, therefore, a representative of a role to the outer world of the scene and objects may use it to fulfill the role. The port has a name that is equal to the target role. Nevertheless, one could draw an arrow through the border of a scene from the object to the role, but this is considered to be a bad modeling style (see Section 6.1).

The interaction of the inner roles is representing the behavior flow. Further, a scene can be determined by the current state of all its inner roles' instances, called *situation*. As a special field, the situation of a scene is a runtime state of a scene as a whole and changes with every role instance alteration.

> "Each situation in a scene is a temporary part of the scene, forming a unified whole in time." [8, p. 29]

In BROS, there is no further usage of situations yet. It is a construct for future language extensions and may be used for logical reasoning, e.g., prediction or calculation of states of whole scenes after a series of events (like state machines or petri nets).

Events

The set of *events* within a scene specifies the temporal behavior of the scene's roles (events are further explained in the next section). Every event that may occur in the scene context is considered to be in the scene's set of events. Events (or rather the event's instances) that belong to a particular scene are used for that specific context and do not have any effect if the scene does not exist at runtime. Nevertheless, the events might use ports to access the roles of nested scenes. Some events may be modeled in a special for starting or ending the scene (see Section 4.4.4).

Scenes and Compartments

It is possible to nest scenes in other scenes; they then function as a kind of subprocesses of the outer scene. There is technically no limit in the number and depth of nested scenes. However, many nested scenes should be avoided to maintain usability. Please note that a nested scene can only be is instantiated at runtime when the surrounding scene is instantiated as well. In no case, a scene is able to contain a compartment as a nested context.

Relationships

Scenes do not have special relationships. They are the context of roles and represent certain business behavior. However, scenes might use constraints at runtime (e.g., the scene `rent car` can only occur after the scene `login`). Scenes theoretically support constraints like roles (implication, equivalence, and prohibition). Nevertheless, often, scene constraints can be better implemented by using events.

Table 4.5 Example process models that can be used to derive BROS events

Process Model Type	Event Template Element
BPMN	Event, Split, Action, Flow, . . .
UML Sequence Diagram	Message
Event-driven Process Chain (EPC)	Input, Output, Event
Petri Net	Transition
Prosa Text Process Description	Verbs (words in text)

4.4.4 Events

When it comes to the specification of exact times, common structural modeling languages are often not sufficient. Due to their nature, it is not the task of a structural modeling language to represent such points in time. However, when using the language for adaptation, temporal specification is necessary. The *events* are, besides scenes, the most important concept in BROS to define temporal behavior. While scenes define the context boundary for roles that are participating in the behavior, events are the concrete points in time that define the foundational execution of the behavior, especially the lifetime of a role in that behavior context.

Definition 4.6 Event A contextual modeling construct for specifying the temporal behavior of roles. It occurs at a specific instant of time, with possible re-occurrences. An event may have causes to define the entities that may trigger the event and determines the object fulfillment of roles and start and end of scenes.

Semantics

Events can be considered as occurrences in the flow of behavior. Such occurrences may happen in many different expressions. It depends on the modeler, how to choose the right events for their BROS models and adaptations. Some examples of process models that can be used to derive BROS events are listed in Table 4.5.

> "Quite often, what appears at one granularity level as a process may, when described at a finer granularity, be seen as a sequence of events." [91, p. 10]

The breakdown of a process flow into several events may lead to an issue about BROS modeling of events regarding the actual background process. A BROS model is not supposed to be a process model, it can not and will not reflect the whole process. The events used are not for representing the process flow but for the specification of roles and scenes. Thus, events can be seen as interface points between the background process and the structure of the BROS model. As long as an event does not interfere with the lifetime of a structural element (a role or a scene), it is not relevant for the BROS model. Only the relevant events are considered to be part of BROS, used as interfaces to determine the state and behavior of roles [223]. E.g., if the BPMN event `sign` is required for the start of the role `Customer`, it can be used as a BROS event.

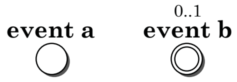


Fig. 4.17 The BROS event as two different types: (a) as regular event, (b) as return event

In general, BROS events are defined very vaguely since there are many different ways on how to determine the right BROS events. As long as an event can be described with a unique identifier (name) and an exact point in time, it is sufficient.

> "An event is a specification of something that may occur at a specific point in time when something of interest happens relative to the properties and behaviors being modeled, such as the change in value of a Property or the beginning of execution of an Activity." [187, p. 74]

The event is modeled on the design time level as an entity, which means that an event has an instance at runtime. In contrast to all other entities in BROS, the event trigger is global and "known" by all event's runtime instances in the model. Nevertheless, the effects of an event may be different, depending on the modeled context: e.g., the (design time) event `sign` happens once for the whole model, its instances at runtime are triggered, but the effects may be different (dependent on the context of the event). If the `sign` event is, e.g., only modeled within a specific scene, the runtime occurrence of the `sign` does nothing if the specific scene is not instantiated and valid as well.

For technical reasons, all used events in a BROS model are part of at least one *script*. The script is the collection of events that orchestrate the runtime instances of the BROS roles. Further, contexts like scenes can be considered to have their own sub-script of events, affecting the scene's inner roles. However, like situations, the scripts are not used further in BROS and are only important for future language extensions and technical implementations.

Syntax

As shown in Figure 4.17, the event is represented as an exact circle within the BROS model. It has a *name* attached and may be placed anywhere it belongs to. Further, the event may be specified with a *style*, can have a *cause*, and several *create* and *destroy* relationships towards other entities. Contrary to the regular event, a return event (see Paragraph *Relationships – Destroy*) is represented with a double borderline and placed on the edge of the particular scene. Additionally, events can make use of *multiplicities* to state the possible amount of occurrences within the context.

Inner Elements

Style

The *style* of an event is specifying the nature of the respective events. Table 4.6 shows all different kinds of styles available for events. The style icons are drawn within the

Table 4.6 Event styles in BROS

Event Type	Icon	Description
Standard	○	anything, not specified (often a manual user action)
Message	✉	triggered by sending or receiving a message
Timer	⏲	triggered by a specific time that has passed
Condition	▤	triggered by a condition that is true or false
Error	ϟ	triggered by an (undefined or unknown) error that (might) arise
Signal	△	triggered by a state change that is used to determine the next step

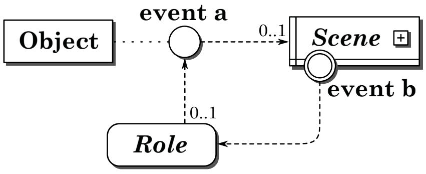


Fig. 4.18 Two BROS events, which orchestrate the behavior: create a scene and destroy a role (a) as well as end the scene and create a role (b)

event circle and derived by BPMN [184]. However, event styles are neither mandatory nor functional. They serve the readability and usability but do not constrain the usage and application of events. Further, it is possible to extend the list of possible styles in future language extensions.

Events are used to state which kind of event is expected (that is why it is called style and not type). Typically, the event styles of a BROS model emerge from the used background process and indicate the process flow conditions that might arise. E.g., the event `sign` could use the signal style if the background process indicates a decision split in the BPMN flow. However, it is still possible to use the message style to indicate that the `sign` event is, for example, a retrieved contract.

Spec

Besides its name and its style, the event has not really more options to be represented in its graphical form. Nevertheless, an event can be complex in its constraints and values. Since the event is only vaguely defined, it does not offer specific fields or operations for the specification. Instead, the BROS language uses an event *spec* to define its specification. The spec can contain different information, e.g., formal or informal descriptions of its occurrence or a link to a BPMN model element. Thus, the form of the spec depends on the chosen background process and the nature of the event. Graphically, it is not represented due to its mutable content and to avoid the resulting cluttering of the model. If necessary, the spec can be written in two ways: (a) as a simple comment box (in a UML style), or (b) as an extern defined listing next to the BROS model. In the latter case, an asterisk ($*$) icon next to the event circle should indicate an externally defined spec.

Relationships

Cause

An event can make use of a *cause* to define its source of triggering. Usually, an event happens globally, and it is undefined who is responsible for the trigger. An event without any cause can, theoretically, be triggered by any other entity in the system. In contrast, when the cause is defined, only the respective entities specified in the cause are able to trigger the event. For example, if the event `sign` does not have any cause, every role and object is (theoretically) able to trigger it. Otherwise, if only the role `Client` is specified in the cause, the event can only be triggered by the `Client` role.

With this mechanism, it is able to define two events that act differently, depending on the specified cause. Another `sign` event may act in other ways when the role `Consultant` is specified in its cause and triggers it. Graphically, the cause is either represented by a dotted line from the event towards the cause entities (like shown in Figure 4.18), or specified in the spec (if there are too many different causes).

Create

The *create* relationship concept is exclusive for events (as well as the *destroy*). It is a relationship that denotes the start of a runtime instance of either a role or a scene and can be used (only) between an event towards a role or an event towards a scene.

> “Occurrences (e.g., events, processes) are also in the realm of the entities that are considered to be in a relation to other entities (objects and other occurrences)” [8, p. 30]

Figure 4.18 shows such a create relationship from `event a` and the `Scene` and another one from `event b` to `Role`. Events use the create (and destroy) relationship towards entities, i.e., roles and scenes, to determine their lifetime. The create relationship is responsible for specifying the beginning of an instance of a role or scene (that is, the start of its lifetime).

The create relationship is used to determine what happens in the BROS model after the event occurred. It may start a whole scene (then called a *init event*), or a single role that is going to be played. This playable role (or scene) is then called *valid* as long as the lifetime is not destroyed by a destroy relationship. Thus, any other entity is (only) able to interact with a valid role or inside a valid scene. However, for consistency reasons, events within a scene should not cross the borders to the outer context, only towards ”more inner” nested contexts (i.e., other scenes that are contained within the scene). This has to be done using a port; otherwise, the event would create (or destroy) the complete nested context instead of an element within the nested context.

Further, when an event creates a scene, all the *init roles* are instantiated as well. Such init roles are roles that require at least one instance in the scene, usually represented by its multiplicity (e.g., 1..*). They exist from the beginning and are created simultaneously with the scene instance (like the init roles would be directly addressed by the init event’s create relationship). However, for simplicity, if there is *no* multiplicity annotated to a role[10] within a scene, this particular role is considered to be an init role if (and only if) there is no dedicated event creating the role during the scene. It is assumed that the role is there from the beginning.

Graphically, the create arrow is a dashed line with a small, narrow, and filled arrowhead towards the role or scene. This direction dependency is unique since the opposite direction would result in the destroy relationship, a conceptually completely different relationship. The create relationship can have a *multiplicity* to state how many instances are affected.

Destroy, Return Events, and Exit Events

The *destroy* relationship is conceptually just the complementary case of create. Instead of starting a role's or scene's instance at runtime, the destroy ends the lifetime of such an instance. It has the same constraints and abilities as the create relationship.

Graphically, the destroy relationship is drawn like the create arrow but in the opposite direction (see Figure 4.18). It might be confusing since it could be read as "the role leads to the event." Nevertheless, the behavior in BROS models is event-driven, thus, the reading of behavior is focused on events, like "the event consumes the role." As well as the create, the destroy can have *multiplicities* as well.

In BROS, there is a special type of event: the *return event*, which specifies the end of a scene (as destroy replacement; e.g., `event B` in Figure 4.18). While a regular event is used for everything, the return event is initiating the end of the whole scene by the same scene itself. It can be compared with a return statement in a method of any programming language, where the return statement initiates the end of the method and delivers one (or more) values back to the more outer context. The return event, triggered from inside like any regular event, ends the scene and everything within. This correlates with the definition of a scene that has a start and an end, which are represented as two (or more) events. Thus, return events are stating the endpoint(s) of a scene. Thereby, return events do not make use of destroy relationships (from inside the scene) but can create other roles and scenes. However, there is a difference between a return event of the scene and an event that destroys the scene from the outside. In the former case, it is the scene that finishes itself; in the latter case, the scene gets interrupted from an event triggered in the outside context (called an *exit event* for the scene). This distinction is made to realize processes that end themselves and processes that are relying on external conditions to end. For both cases, it depends on the way the event is intended. In general, it is recommended to strive for return events instead of interrupting destroy events from the outside. Graphically, a return event is modeled with a double circular line at the edge of the scene (`event b` in Figure 4.18).

[10] Please note that, in such a case, the missing `1..*` leads to the possibility of destroying this role in that scene.

4.5 Extended Concepts

Extended concepts are features of the BROS language that are not validated enough but can be useful, though. Using extended concepts might lead to an inconsistent model state or even impossible model statements, however, when used with care, they can improve specific model statements significantly. BROS uses the following extended concepts:

- Package
- Compartment
- Port
- Group
- Token

Please note that all these concepts are *not* represented as model elements in the stable metamodel (see Section 4.3). The extended model element set needs further investigation to be included in the stable metamodel.

Further, the set of extended concepts can be used for future language extensions, if needed. Therefore, this section only covers the basic functionality and intention of these concepts and is not going into much detail like the core concepts.

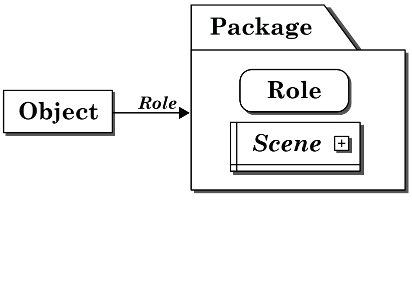
4.5.1 Packages

Packages are well known in modeling languages, often as simple encapsulation concepts or as a container for the whole model. In BROS, a package is the interpretation of an encapsulated adaptation of a reference model. Thus, every BROS model element that is part of a reference model adaptation is contained in such a package.

Semantics

The BROS package is simply a container for roles, scenes, and events that are part of an adaptation. As stated previously, BROS is not dedicated to pure adaptation purposes; one could design a software model without any intention for adaptation. Nevertheless, the original idea and goal of BROS was the easy and non-invasive

Fig. 4.19 A BROS package that contains all other elements used for reference model adaptation



adaptation of standard reference models. Thus, the package contains these elements, that are used for adaptation of the underlying reference model. In fact, the reference model is represented by (and expressed with) objects. The adaptation method is then to add the fulfillment arrows towards roles that execute the adaptation (see also Chapter 5). If it is assumed that the reference model is given as a set of (fixed) objects, the adaptation (i.e., roles, scenes, and events) are best encapsulated by the package due to separation of concerns. Another adaptation of the same reference model (i.e., a fixed set of objects) can be made with a different package.

Syntax

A package follows the regular representation from UML [187]. It is a box with a *name* label on top (see Figure 4.19). Usually, when using a package for adaptation, there are no packages within other packages. However, it is still possible.

Inner Elements

A package can contain everything that is offered by the BROS language and needed for the adaptation, except objects. Further, a package does not have any fields nor operations. Also, there is no port used when an outside object fulfills inner roles or scenes since the package can not be fulfilled itself (unlike, e.g., the compartment).

Relationships

The simplicity of a package as a pure encapsulation entity results in the absence of relationships of a package. There are no constraints or other relationships defined for a package. The only relationship that is close to the package is the fulfillment of outer objects to inner elements.

4.5.2 Compartments

Compartments are a concept mainly used in CROM [151] for a role’s context. Since the BROS language is inspired by CROM, this compartment concept as a construct for static context is maintained, as well as serving compatibility with CROM.

Semantics

The compartment is a kind of precursor of the BROS scene: it is intended to be a context for roles and objectified as well. Like a scene, the compartment has instances at runtime and is considered as an entity. The semantics of a compartment is very similar to a scene, with the following exceptions:

- The compartment is considered to be static since it has not any temporal information about the inner elements. It has no start or end; it simply represents an instantiated context for roles. Thus, the roles within a compartment are static roles (if not combined with an event).
- The compartment can be fulfilled (as it would be a role) and can fulfill other roles (see Figure 4.20)
- A compartment is used for encapsulating roles that belong together concerning the roles' structural coherence. The scene encapsulates roles that belong together with regard to their interaction in behavior or a process.

It is allowed that scenes are placed within compartments and vice versa. To avoid inconsistent model statements, it is not recommended to nest compartments and scenes in each other and separate both concepts. Further information about compartments can be found in Kühn et al. [151].

Syntax

The compartment is written like a scene but without the vertical line on the left side. It has separate sections for its *fields* and *operations*. It may be confused with a regular object; the only difference is the set of inner roles and the ports.

Inner Elements

Besides its *name*, the compartment can have a *multiplicity*, has *fields* (as a state), *operations* (as behavior), and a set of *roles* that are the inner elements of a compartment. Further, a compartment may contain *nested scenes* and *compartments*.

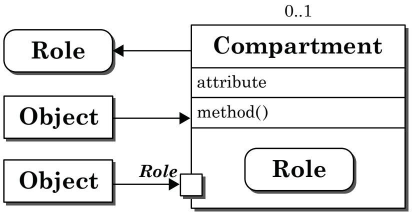


Fig. 4.20 A compartment in BROS for static context of roles

Relationships

Since the compartment can be considered as a role, it has the same relationships as a role: *fulfillment*, *associations*, and *constraints*. Each relationship comes with their specified syntax, especially the *multiplicities*.

4.5.3 Ports

The *port* concept is an extended feature and not stated in the metamodel in Section 4.3. Ports are not used for basic modeling of structure but may help to specify certain business requirements.

Semantics

The port is intended as an element to access inner entities within a context (e.g., scene). The port is already known in UML [187] as a gate for inner components. In BROS, the port is primary for accessing the inner roles in a scene. It has the name of the target role attached, and outside entities are allowed to use the port when they interact with the role. Currently, with the writing of this thesis, the port is not part of the core concept set and has no more functionality than easier readability (as a replacement for crossing the scene's borderlines).

The port, as a part of the extended concept set, can be used for more detailed modeling statements. The most important additional task of an (extended) port is to control the instances that get access. Nevertheless, other relationships (like conditions) can use the port as well. The port can specify a condition to restrict the allowed inner access, called *port conditions*. Some examples of port conditions are

- the absolute number of runtime accesses
- a condition of a specific *field* value at runtime
- logical expressions of more complex statements (e.g., Boolean expressions)

Like the spec of an event, the conditions of a port are rather vague to be flexible for future language extensions. With the extended port version, it is possible to model certain "slots" for, e.g., an inner role of a scene. For example, a role can have two ports `S[:2]` and `S'[<condition>:*]`, where the first one `S` allows two instances to play a role without condition, every other role has to use the second port `S'` with some condition applied. Another example would be a port `P[age>18:*]` which only allows runtime access for object instances with the field `age` and its value greater than `18`. Such a more fine-grained specification of behavior (especially plays) is possible, however, the subject of future language extensions.

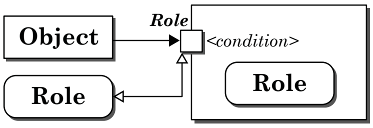


Fig. 4.21 A port to access inner elements of a context

Syntax

A port is drawn as a simple, small square at the edge of the border of a context. Besides its *name*, that is usually the target role from the inside, it can have a *condition* applied. Like the spec for events, this port condition is either attached to the port or used in an external spec (please note to use an asterisk ($*$) to indicate such an external specification).

Inner Elements

Besides its name and condition, the port does not have any further inner elements. For future extensions, it might be useful to define *fields* and *operations* for a port. This is, however, not yet implemented.

Relationships

The port does not offer any relationships besides the pass-through of other relationships. The port is always connected to a certain context, thus, this connection can be regarded as a relationship towards a context. Since the port is always drawn on the edge of such a context, there is no graphical representation of the relationship.

4.5.4 Groups

A role *group* is a convenience construct to encapsulate roles that share similar semantics, especially the cardinality of the roles.

Semantics

The role group is, in the first place, a construct to encapsulate multiple roles and over which certain conditions are applied. These conditions are usually group multiplicities that denote a single, shared cardinality range for several roles. Further, Boolean operators (e.g., or, and) might be specified between the grouped roles. These conditions of a group's roles distinguish it from a package. The role group and its semantic is mainly derived from CROM [151].

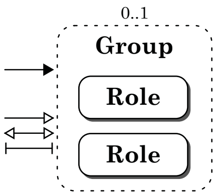


Fig. 4.22 A group to encapsulate several roles

> "[. . .] role groups are a novel construct to impose a cardinality constraint on the players of a set of roles inspired by the *cardinality operator* [Van Hentenryck and Deville, 1990] [referring [263], ed. Note] [. . .]" [149, p. 105]

The role group establishes a pool of roles where the group multiplicity (i.e., cardinality in the definition) affects all included roles in order to specify model statements between them (e.g., "Two of these three roles have to be played."). It is intended to state constraints between several roles. Instead of examining each role individually and for itself, the group can be used to exploit semantic similarities (often, the shared pool of a multiplicity). The most used model statement, where a group is used for, is the *n-out-of-m* pattern, where the group multiplicity is used to restrict the playing of the inner role. However, various other statements are still possible with the possibilities of a role group.

Syntax

The role group is represented by a dash-lined box with rounded corners (see Figure 4.22). Its *name* is in the upper part, all belonging roles are inside the box. There might be a *multiplicity* right above the top of the role group.

Inner Elements

The inner elements of a group are roles (besides its *name* and *multiplicity*). Besides roles, there is no possibility for other concepts to be grouped together (at least, it is not defined yet). It is imaginable to use groups for events, however, events are triggered by environment conditions and other roles and, therefore, differ from the mechanism of playing a role from a pool.

Relationships

Aside from the multiplicity pool, the group can be used to group multiple relationships towards each role and use the group as a relationship end instead. Thus, a fulfillment arrow can address the whole group instead of each role individually (see Figure 4.22). The same holds for constraints concerning the inner roles. Which concrete role is fulfilled depends on the business logic. However, the given group constraints (i.e., the multiplicity or any Boolean operator) might limit the set of possible target roles. Further, it is technically and graphically complicated to assign roles from different contexts into a single role group. In such a case, bilateral constraints are often more appropriate.

4.5.5 Token

The *token* concept is different from the other core and extended concepts proposed. While the other concepts are considered to be design time entities, tokens are a runtime concept for identity management during the execution phase.

Semantics

Tokens can be seen as an alias of different identities. When a role plays a role (at runtime), the identity of the role is the same as the object's identity. However, when using contexts, it might be unclear which identity fulfills certain roles. Especially when constraints and deep roles are used, it can be difficult to model which identity is used for certain roles.

For that, tokens are used for different identities when fulfilling a role. This equals a unique ID for an object's runtime identity. Tokens can be used whenever an identity is important (e.g., association, constraints). Usually, the token is an identity label for a certain other concept, e.g., a role, or a port. However, it can also be used for relationships, e.g., the association. Although labels in angle brackets (< and >) are a reserved syntax for tokens, tokens are considered as identity naming labels and are not fixed in their usage or expression. The token concept is part of the extended set since the definition and usage of a token are vague. There are two major types of tokens:

- tokens for identities that already exist, and
- tokens that get created when used.

For the former case, the modeler can use a token set for a fulfillment to create several "slots" for certain identities. For example, an `Object` may use the token set `<1,3>` to bind two of its identities to the slots labeled `1` and `3` when fulfilling a `Role` (via the port). This is represented in Figure 4.23. Then, when using the role any further, the modeler can specify exactly these two fulfilling identities (labeled with `1` and `3`). For example, the identity `1` is used in an association with `Role b`, where the identity of `Role b` must be the fulfilling identity of `2` from the `Object`. Tokens can be used whenever an explicit (role fulfilling) identity is a prerequisite for a certain action, not only for associations and but also for other relationships like create and destroy or further fulfillments of nested scene's roles.

The second type of tokens are create tokens – the only difference in their meaning from regular tokens is the referenced identity's nature. As its name implies, create

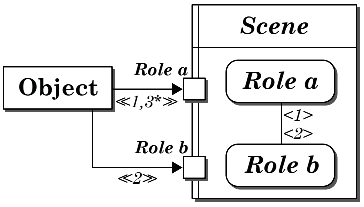


Fig. 4.23 Tokens used for identity management in BROS

tokens are not assumed to be already existing (i.e., the regular case is simply undefined about the identity nature), the create token states the creation of a new identity for the respective object explicitly. This can be useful to denote a certain participant's identity that gets created while a process is executed. Such a create token was used by Schön et al. [223], named as "create fulfillment", to state that a `tasty pizza` role identity gets created during the process as a new `pizza` object identity. This is, however, semantically equivalent to identity create tokens.

In general, tokens often only are advantageous in combination with ports for accessing contexts (e.g., a scene or compartment) to describe the explicit interaction of two or more identities (via role association) or restrict some fulfillments. Especially, when objects' identities are going to be used in subsequent or nested scenes, tokens can be helpful to determine the identity of forward-shifted role fulfillments from one into another context. In theory, tokens are allowed for static roles as well, however, it is not recommended due to minor usability and a possibly higher chance for invalid statements.

Just like every other extended concept, the tokens are not used as a core concept since it needs further validation and might lead to inconsistent model states. However, the token concept can be used for more precise model statements, while it is recommended to use it only for smaller model parts. Further, it allows a future language extension to make use of such tokens if needed.

Syntax

Graphically, tokens are written like labels in angle brackets for every other concept, as shown in Figure 4.23. They are numeric and attached to the respective element. E.g., for an association, tokens are written at the end of the arrow; for ports, the token is written beside the name. A distinction is made between the *assignment* and *usage* of token labels: the former case is represented with two brackets on each side (ref. Figure 4.23, the fulfillment between `Object` and the ports) and states the newly bound identities (assignment). In the latter case (the usage) is represented with single brackets to state that an already assigned identity is used. Further, the create token is represented by an asterisk ($*$) in the label (ref. Figure 4.23, the fulfillment with ≪`1,3*`≫). The create token is only meaningful for the assignment, the usage of create tokens is not defined.

Chapter 5
Role-based Adaptation of Reference Models

"No matter how technically awesome a particular system is, if it is not able to meet the needs of the business (both functional and operational needs) the software is of no use." [118, p. 178]

Abstract The adaptation of reference models is crucial when building enterprise-specific software systems according to specific standards. The right choice of design decisions and activities during the adaptation process is, therefore, an essential part of a proper and suitable result. There are plenty of modeling languages to define structural models regarding a reference model, most commonly the UML. An adaptation is in need of a modeling language that offers the highest degree of transparency, traceability, and non-invasiveness. The BROS language was designed for that purpose and offers, besides those properties, a strong and powerful set of modeling concepts to define such adaptations. Nevertheless, a language alone does not make an application model by adaptation. Rather, a well-defined adaptation method is needed as a rule set and guideline for the right design decisions when adapting a reference model. Thus, in this chapter, the BROS adaptation method is proposed, tailored to adapt an object-oriented reference model with the help of the BROS language towards an enterprise-specific (role-based) application model. For that, several phases are introduced, and an instruction set is proposed to perform the adaptation.

5.1 Reference Model Adaptation with BROS

The adaptation, as a subject in-between evolution and maintenance, is an essential part of creating and developing enterprise-specific software systems. On the one hand, due to the individual requirements of an enterprise, there is hardly a standard solution that is sufficient for multiple enterprises. On the other hand, large enterprises and their software systems are in need of standardization [90]. They are often required to adhere to legal or functional constraints. Not only external factors are the drivers of adaptation, but also internal development and new requirements are often the reason for further development in the sense of adaptation. In general, the adaptation has to bridge the individual requirements of an enterprise and the standardized circumstances of critical software systems (see Chapter 2).

> "The evolution of systems is driven by a number of external forces, especially continuous flow of requirements for the implementation of new functionality and new technology, resulting in a considerable rate of change and at the same time the need to continuously enhance the efficiency of the system." [181, p. 23]

In this thesis, the adaptation of systems with the BROS language (see Chapter 4) is in focus to solve the issue and to bridge the gap. Using the BROS adaptation method is significantly different from the usual ad-hoc adaption that is carried out conventionally. Instead of adapting the reference model itself, adaptation logic is shifted to a new level by roles and thus abstracted from the original model. BROS was developed for easy reference model adaptation, especially in the object-oriented (OO) models area. The roles paradigm, which is the main driver for the adaptation with BROS, leads to a non-invasive and extensive application model construction. The result of the reference model adaptation via BROS meets the enterprise-specific business requirements. By separation of the adaptation from the underlying reference model, the adaptation is both structured and traceable. This is immensely helpful when the reference model changes itself (see Chapter 5.3).

The expressiveness of the application model of a performed adaptation varies with the required granularity that is used for the adaptation itself. In general, the pure application of roles as adaptation activity is already sufficient to count as a BROS adaptation. In contrast, if the adaptation makes use of the full stack of adaptation activities (which includes many more model elements), the complexity and expressiveness increase. It depends on the modeler to set the adaptation requirements of the application model granularity.

5.1.1 Project Structure

The adaptation itself is only a part of an adaptation project. For proper documentation and maintenance of the adaptation, the "physical" components of the model are more than the adaptation as a BROS model. Especially when making multiple adaptations on the same project, it is essential to keep information and decisions about the various

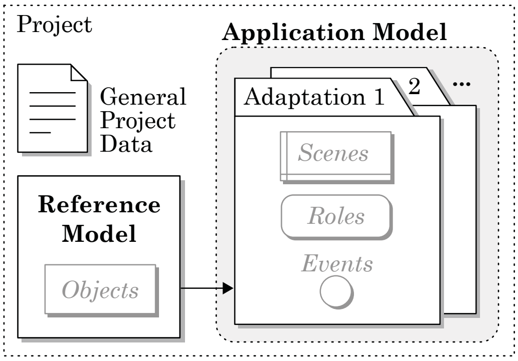


Fig. 5.1 The structure of the adaptation process, including the reference model, application model, the individual adaptations (possibly as packages), and the additionally relevant data

adaptations closely together. Such an ideal project consists of four components: the reference model (RM), the application model in a certain state (AM_n), the individual adaptations within the application model ($a_1 \dots a_n$), and the other information called *general project data* (GPD) which also include the background processes. Figure 5.1 visualizes the four components in a schematic overview.

Reference Model

To perform an adaptation with BROS, the reference model (as the domain model, cf. Section 4.2.1) in use is the foundation of the adaptation, and it can not be done without it[1]. It is assumed that the reference model has the right level of granularity in its OO-based structure. There is no need for a background process (see Section 4.2.2) for a successful adaptation since BROS and the adaptation are aiming for structural models. Nevertheless, a background process is beneficial when designing the application model via adaptation (especially the application of temporal concepts, e.g., events, is in need of a detailed background process). Figure 5.1 proposes an ideal project structure that includes the reference model as an essential part.

In theory, an adaptation implementation process can make use of multiple reference models. This can be useful when a single reference model is not sufficient to cover the whole domain model side. For that, multiple reference models serve as templates, ideally for different, separate, and non-overlapping parts. However, if the reference models overlap in any way, inconsistencies must be considered and solved beforehand. Multiple reference models count as a single one (with regard to the adaptation process). Nevertheless, the respective linking between the several individual reference models and the resulting application model adaptations must be traceable at any time, also for future adaptations.

Application Model

The application model is at the heart of the adaptation implementation; it is the model the whole adaptation is made for (see Section 2.3). Retrieving the application model from a reference model is common sense, however, how the application model looks like is very different. First of all, it has to be on the same technical layer, i.e., an application model in the technical OO-layer requires a reference model in the OO-layer as well.

Besides the layout and technical layer of an application model, the content and "shape" of the application model can be different as well. For example, an application model (AM) derived from a standardized UML reference model (RM) is probably the whole reference model with some changes (thus, $RM \in AM$). There is, in fact, no optimal method to abstract the application model in such a way that the reference model is not a part of the application model anymore. Nevertheless, this is not

[1] It is still possible to construct BROS models from scratch, without a reference model and adaptation intentions.

necessary if the adaptation is made consistent and non-invasive (like the BROS adaptation method). According to Figure 5.1, the application model derived from a BROS adaptation is slightly different in its terminology and "shape": instead of using the reference model as an internal part of the application model (with changes), the application model consists of several adaptations (individual changes) that reference towards the respective reference model. Thus, the definition of the application model can be expressed as

$$AM_n = \{a_1 \ldots a_n\}^{RM} \tag{5.1}$$

with $a_1 \ldots a_n$ as individually made adaptations that reference to the underlying reference model. In general, an application model derived via a BROS adaptation can not be stated without the underlying reference model. However, such a reference model is only for *reference* and not an internal part of the application model itself. The individual adaptations are considered to be adaptations made independent from each other, encapsulated, and structured in a package structure (if possible).

The BROS adaptation of a reference model is always dependent on the information and details of the reference model (granularity and domain completeness). If the reference model is a set of business objects, it is easier to adapt it towards the intended structure as the given objects can be composed more freely. In contrast, when adapting a reference model in, e.g., a UML style (with a lot of technical details and established and complete relationships), the adaptation is bound on the established degree of details: objects and their relationships are constrained by the given (detailed) relationships. Thus, they are limited in their adaptability. This is reflected by the single instructions and may limit the possibilities of adaptations with regard to new or desired object-role constellations. In general, a valid and stable adaptation should and can only refine the reference model, not generalize or modify it.

Adaptation

The adaptation is considered as a single activity to derive the application model from the reference model. However, the adaptation is not the application model itself. Every past, ongoing, and future adaptation implementation is part of the application model and extends it further to the final state[2]. This is also represented in Figure 5.1. While it is not necessary to include more than one adaptation within the project, the possibility enables certain adaptation practices, like constructing the final application model in parallel or in distributed ways.

The implementation process of the adaptation is described in the next Section 5.2. Each individual adaptation is made in a BROS adaptation method implementation

[2] On the condition that every adaptation is following the same goals and intentions for the application model (and are not completely different ones).

process (executing the instructions with basic steps). The encapsulation of an adaptation (or rather its results) in packages is of advantage for traceability and separation. Since packages (see Section 4.5.1) are not part of the BROS language core element set but an extended model element, it is not necessary but recommended to use packages.

For its application, the individual adaptations have to be based on a given reference model. However, it is also possible that an already existing application model (AM_n) is extended by a new adaptation to result in the next version or state of that application model (AM_{n+1}):

$$AM_{n+1} = \{a_1 \ldots a_n\}^{RM} \cap a_{n+1}^{RM} \tag{5.2}$$

Please note that the new adaptation a_{n+1} is <u>not</u> based on the previous adaptations but still on the template reference model. If it is the case that a previously made adaptation is not correct anymore (overhauled, errors, etc.), it has to be removed from the AM set, corrected, and added again. This non-stacking behavior of adaptations ensures a very separated and discrete process of adapting a reference model. Due to the nature of role-based adaptation, overlapping adaptations are possible: an RM entity can play several roles at runtime for different contexts. However, overlapping adaptation areas can lead to problems, especially when applied to the same application domain (e.g., two types of the same context `Rental`). Thus, it is recommended to model the individual adaptations as separately as possible.

General Project Data (GPD)

Everything that is not necessarily part of the application model but enriches the implementation with more detailed information belongs to the *general project data* (GPD). Especially the used background processes are an essential part of the GPD. With such information, future adaptations can be made with more precision since every meta information helps to keep track of the adaptation process and the requirements of future adaptations.

There is no exact layout of the GPD. However, it should consist of at least the following values:

- The background process(es)
- Authors and stakeholders
- Identification data (e.g., date, time, id, version)
- Statements, decisions, and goals

To improve the utilization of the GPD, the respective file should be structured systematically (e.g., as *JSON* or *XML* format). This allows easy access and supports readability for humans and machines. It is recommended to include a section for each adaptation implemented since such information is helpful for transparency. In contrast to Figure 5.1, the GPD can be separated and distributed (e.g., when using more complex database schema for storing the respective information). In general, the versioning of the different updated GPD files is recommended.

5.1.2 Variants

The adaptation implementation of the BROS method can be considered as a predefined list of several steps that need to be considered (that is, phases with instructions). In general, there are mainly two variants on how to adapt a given reference model with BROS, leading to a slightly different adaptation implementation, especially concerning the roles:

1. the progressive adaptation, and
2. the regressive adaptation.

The former is a variant often dedicated to architectural thinking regarding construction and adaptation, while the latter is the alternative, i.e., procedural thinking. Since both are using the same reference model as input and produce the same application model type as an output[3], they are considered to be *adaptation method variants*. The BROS *adaptation method* is still the adaptation of an OO-based reference model via the BROS language. The final adaptation method is then always executed in five phases (see Section 5.1.3 and Section 5.1.4), which is valid for both variants). The implementation of the different variants is explained in the relevant phase in Section 5.2.3.

5.1.2.1 Progressive Adaptation

The *progressive* method variant is based on the view of the structural side of the given reference model, using roles as "representatives" of objects.

Definition 5.1 Progressive Adaptation The Business Role-Object Specification adaptation method of a reference model towards an application model by using the given objects as a guideline for the adaptation design decisions.

With a well-defined OO-based reference model, the progressive adaptation is made by adapting the given reference model objects towards the needed role concepts. It is some kind of "forward-thinking" and has the benefit that there are clearly defined responsibilities of which object fulfills which role in a certain context. For a progressive adaptation, the perspective is driven by the existence of already defined objects in the reference model. Thus, the adaptation idea could be "Which object can participate in the context and how?" For example, a progressive adaptation for the context `Car Rental` could lead to the design decision to adapt the existing reference model object `Car` towards a `Camper` if it is needed.

For context and entity definition, the progressive variant is easier to implement since the chosen reference model determines the actual facts and, thus, making adaptations decisions more reliable[4]. The adaptation can be carried out with the

[3] However, the content of the application models can vary.

[4] For example, it is easier to adapt the `Person` object with a `Customer` role if it is known what structure and behavior come with the `Person` object.

certainty that the reference model supports it with its elements. It might be obvious, but if one chooses to adapt the `Car` towards the `Camper`, it is quite sure that the `Car` is existing for adaptation (in contrast to the regressive adaptation method variant). However, due to the restriction and focus on the given reference model, the enterprise's requests and possibilities are of secondary importance.

A progressive BROS adaptation is probably the more straightforward and common variant since it is based on the traditional paradigm of inheritance: the top-down approach of deriving sub-objects from each other. To think about the possible sub-classes of already existing classes is similar to this progressive approach. The progressive adaptation leads to the fact that the resulting adaptation and its contexts are not designed independently from the reference model; there is a strong connection between both. The existing objects lead to the possibilities in the adaptation.

5.1.2.2 Regressive Adaptation

The *regressive* adaptation is a slightly different approach to the progressive one and aims for the target adaptation, mainly utilizing roles as "participants" [54] in contexts.

Definition 5.2 Regressive Adaptation The Business Role-Object Specification adaptation method of a reference model towards an application model by using the targeted (behavior) environment and context as a guideline for the adaptation design decisions.

Instead of thinking in classes, sub-classes, and which object can be adapted to participate in a context, the regressive adaptation method variant is driven by "backward-thinking." The target adaptation comes first, the related objects for the adaptation second. Especially when designing the context, the backward-thinking approach can lead to more independent and specified structures. One can design the application model as intended without focusing on the (limited) set of the reference model. Therefore, it has more similarities with an aspect-oriented than an object-oriented paradigm. The main adaptation idea can be summed up with "Which object can be used for this adaptation in this context?" There is a slight difference on how to select objects from the reference model: while the progressive adaptation selects the given objects first and the adaptation target (for a context) afterward, the regressive adaptation selects the target adaptation (that is needed for a context) first and the objects in question second. For example, for the same context `Car Rental`, one specifies that a `Caravan` is needed. The reference model objects are getting scanned, and the `Car` could be chosen appropriately. Such, instead of "taking" the reference model object first, the needed adaptation for the context is in focus.

When defining the contexts and entities for an adaptation, the regressive variant is independent of given circumstances from the reference model. Since the application model should represent the specific enterprise, this variant leads to an application model more tailored for the individual needs. The layout and the chosen contexts of the model are inspired by the design of the target structure (the real enterprise), instead of the given boundaries of the reference model (like in the progressive variant). For example, if the enterprise definitely needs to model a `Caravan`, with the regressive

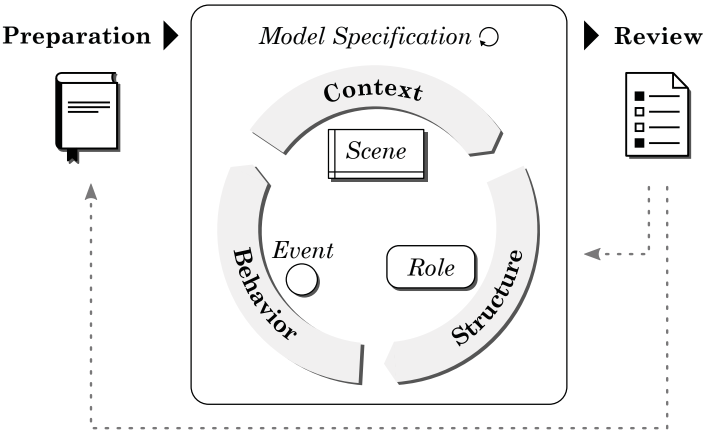


Fig. 5.2 The BROS adaptation method phases

adaptation method variant, the `Caravan` will be modeled (and possibly the `Car` from the reference model will be found afterward).

The regressive adaptation is probably less common than the progressive version, since the aspect- and the context-focused view is not that popular. Further, the conception of the adaptation can be more difficult as the final adaptation is defined first, and the reference model comes second. This may lead to inconsistencies. However, the substantial benefit of the regressive method is the independence of the underlying reference model. The target adaptation can be designed nearly without any restriction, and the focus is on the context and the adaptations (of objects) that are intended. The reference model objects that are getting adapted towards the goal are chosen afterward.

5.1.3 Adaptation Phases

The adaptation via BROS is specified as a methodical approach. It consists of five separated phases, each phase uses the concepts of the BROS language to fulfill the whole adaptation process:

1. **Preparation**, the phase of starting the adaptation, defining its meta data and setting the adaptation in relationship with the reference model (see Section 5.2.1)
2. **Context Specification**, the phase where all the needed contexts are defined (see Section 5.2.2). For this, the interaction with the reference model and background process is essential
3. **Structure Specification**, the generation of roles and other structure for the adaptation, building the bridge between the reference model and the behavior (see Section 5.2.3)

4. **Behavior Specification**, the follow-up phase after introducing roles to the contexts. In this phase, the temporal effects and constraints are introduced to orchestrate the roles according to the background behavior (see Section 5.2.4)
5. **Review**, as the final step to complete the adaptation process and control the design decisions (see Section 5.2.5)

Figure 5.2 shows the phases in relation to each other. Each phase has its own intention to contribute to the adaptation process. However, the phases are not expressed as a set of steps themselves. Instead, the whole adaptation process with BROS contains an *instruction set* to describe the single steps (see next Section 5.1.4).

In general, the implementation of each phase depends on the applied use case. While the preparation and review phases are only implemented once, the main phases (context, structure, and behavior specification) are iterative processes. Dependent on what and how the adaptation is executed, the specification of structure, behavior, and context can be processed in different order and intensity.

5.1.4 Instruction Set

As introduced in the previous Section 5.1.3, the adaptation with BROS is divided into several *instructions*, which are related to several phases. An overview of all available instructions is given in Table 5.1, which are explained further in the following sections.

These instructions are used to describe what is happening in the execution of the adaptation. Thus, instead of describing the whole phase, the description is given by completing the instructions per phase. Further, it is possible to change between instructions and, eventually, skip certain instructions that are not necessary. This always depends on the individual needs of the adaptation implementation process and which action is needed to successfully implement the adaptation. Furthermore, some instructions depend on whether there is an existing application model (which is to be extended) or if the model is to be completely built from scratch. An instruction can be described as a set divided into the following categories:

- **Objective** as the description of the objectives of the instruction,
- **Input** as the needed prerequisites and things that need to be present for starting the instruction's execution,
- **Output** as a description of the instruction's result,
- **Activities** is a list of *steps* that are recommended to fulfill the instruction's objectives, and
- **Example** for an easier understanding of the instruction.

Since the adaptation process is a methodological approach, the single categories are described in a non-formal manner. For each instruction, there is an *Example* category, which shows the instruction in practice for a car rental enterprise called "Road Trip Car". This is purely for a schematic illustration of the instruction and its applicability concerning the whole adaptation process and not a real-world industrial use case.

Table 5.1 Complete BROS adaptation method instruction set

Name	Abbrv.	Description
Preparation		
Adaptation Objective Definition	p.AOD	Define the goals, boundaries, and domain of the adaptation and gather all related information.
Reference Model Selection	p.RMS	*(mandatory)* Select one (or more) appropriate reference model(s) as a template for the application model or an existing application model to be extended.
Background Process Selection	p.BPS	Select one (or more) appropriate background process(es) as behavior guidance for the application model.
Context Specification		
Package	c.PAC	Specify and/or extend the adaptation packages.
Dynamic Context	c.DYC	Specify a dynamic context (scene).
Static Context	c.STC	Specify a static context (compartment).
Structure Specification		
Role	s.ROL	Specify a role.
Association	s.ASC	Specify an association between two roles.
Constraint	s.CON	Specify a constraint between two roles.
Group	s.GRP	Specify a group for multiple roles.
Port	s.POR	Specify a port for a role within a context.
Behavior Specification		
Event	b.EVE	Specify an event or return event.
Token	b.TOK	Specify a token for a model element.
Review		
Verification Check	r.VER	Verify the model syntax according to the BROS language and method rules and guidelines.
Validation Check	r.VAL	Validate the model semantics according to the adaptation goals and statements.
Feedback Loop	r.FBL	Utilize a feedback loop.
Documentation	r.DOC	Document the adaptation process.

The separation of the single steps into instructions also benefits from the idea of an adaptation DSL (which is, however, not part of this thesis but future work, see appendix Section C.1). Further, the set of instructions can easily be extended or tailored: instead of changing the whole adaptation phases, the instructions are independent enough to support individual requirements or future developments (e.g., new or changing instructions). Documentation of the adaptation implementation is much easier when assigning and following small and precise instructions instead of describing the whole phase on its own. The instructions are considered to be independent of each other to enable combination and repetition of instructions. Together with the iterative phase design, the whole model can be adapted towards an individual application model, driven by the needs of the enterprise. It is, as a result of this, important to note that the content of the instructions is representing an ideal process for guidelining the adaptation. Since every adaptation is different from each other (that is why it results in individual application models), the adaptation process can not be enforced by fixing the instruction executions. Thus, the listed steps within the instructions are recommendations to succeed in achieving the instructions' objectives.

5.2 Adaptation Implementation

The implementation of an adaptation is done in five phases, as written in Section 5.1.3. Each phase consists of several instructions that are used to construct an enterprise-specific application model (see Section 5.1.4). The *implementation* of the adaptation is an execution order of the instructions of each phase.

5.2.1 Preparation

The preparation phase consists of instructions that are related to the start of an adaptation. The instructions in the preparation are rather trivial, however, they are of an advantage when executing the adaptation in practice. An overview of the preparation instructions is given in Table 5.2. In contrast to the other phases, the preparation depends on whether or not there is an existing application model that can be used further. It is a lot easier to extend an existing application model than building a new one. This concludes that the instructions of the preparation phase have an additional category "On Extension." It describes the differences when following the instructions of the preparation phase when using an already adapted application model (e.g., implementing an additional adaptation or when returning from the review phase due to a failed validation).

Table 5.2 Preparation instructions

Name	Abbrv.	Description
Adaptation Objective Definition	p.AOD	Define the goals, boundaries, and domain of the adaptation and gather all related information.
Reference Model Selection	p.RMS	*(mandatory)* Select one (or more) appropriate reference model(s) as a template for the application model or an existing (role-based) application model to be extended.
Background Process Selection	p.BPS	Select one (or more) appropriate background process(es) as behavior guidance for the application model.

Adaptation Objective Definition / p.AOD

Define the goals, boundaries, and domain of the adaptation and gather all related information.

Objective The exact definition of the adaptation in an informal manner is useful for the general usability of adapted reference models. A labeled and specified goal and capability set of the adaptation can be used to tailor and streamline the expected future application model. This can be used if, e.g., not the whole enterprise should be represented in the future application model but only parts of it. Furthermore, the collection of all relevant information about the targeted application model is of an immense advantage when following the adaptation method. However, this instruction is mainly to support the instructions *p.RMS* and *p.BPS*, why this instruction *p.AOD* should be done first.

Input

- *(optional)* Related information about the targeted application model and its intentions (e.g., requirements, use cases, user interface mock-ups, standards, neighborhood systems and their interfaces, business processes, glossaries)

Output A structured set of adaptation facts (goals, boundaries, etc.) and meta data for the GPD.

Activities

1. Set the *organizational context* for the adaptation implementation, e.g.:
   - What should be changed by using and adapting a reference model?
   - How would the roll-out of the model be executed (after the implementation of the adaptation)?
   - Who is responsible? Who are the stakeholders?
   - What is expected? (e.g., a complete system, a module, a back-end service)[5]
   - What is out of scope?
2. *(optional)* Formulate the main *goals* and *sub-goals* of the future application model
3. *(optional)* Specify the *adaptation statements*:
   a. Identify the missing issues between the reference model, the background processes, and the goals
   b. Open a statement for every fact that needs to be adapted in the reference model
   c. Use state-like descriptions to describe how the application model solve the issues (like use cases)[6]; include it into the adaptation statements

[5] For this step, general activities for initiating a software project should be considered (e.g., as described in [90], [141], or [181]).

[6] For this, it is recommended to follow the guidelines of requirements engineering, analysis, and agile development, e.g., as stated in [65] or [165] (especially use cases and user stories).

   d. Review the statements whether they cover the expectations and whether they can be used for later acceptance tests
   e. Check and (if necessary) include new implied statements that emerge from the former ones
   f. Organize the statements in hierarchical form

4. Specify the adaptation's *boundary*:

   a. Identify the targeted domain coverage of the adaptation (e.g., components, parts, functionality)
   b. Formulate parts of the domain that are explicitly included
   c. Formulate parts of the domain that are explicitly excluded

5. Specification of (at least) the following values and adding them to the GPD

   - General project data (date, author, name, version, . . .)
   - Adaptation process values
     - name/id
     - version
     - author
     - date
     - adaptation goal description
     - the adaptation statements
     - the boundary (together with their meta data, e.g., names, versions, content)

6. Add the GPD to the project

On Extension When an already existing application model is to be extended, the GPD should be investigated. The facts stated in the GPD are valid for the whole application model, which includes every single adaptation (package). Ideally, the previously defined goals and boundaries remain unchanged if they are available. Since every adaptation is a new part of the application model, a new adaptation is not interfering with the already existing adaptations. If necessary, review the facts in the GPD and update the values and statements. However, versioning of the GPD is recommended since the other adaptations might use the old GPD version.

Example After feedback with all relevant stakeholders, the future application model of the "Road Trip Car" enterprise should represent the long-term rental part of the enterprise's domain. Thus, the rental part is going to be constructed as an application model for a software system construction later on. All organizational information is considered and structured. After reviewing the requirements and several use case diagrams, the enterprise's software architect and business process manager identified the (rather independent) domain parts "long-term rental," "long-term maintenance," and "long-term reservation," since they are the individual sub-domains that are affected by the adaptation[7]. Next, the stakeholders decide to implement the "long-term rental" boundary only (and the others later on), since it is the most important

[7] Please note that in real-world scenarios, the boundaries can be more precise, more complex, or smaller (e.g., "sales and logistics", "credit card payment" or "middle management employment").

one. Every boundary would get handled as its own adaptation process in the future (while sharing the same GPD). As for the adaptation statements, e.g., “The long-term car has to be a camper.” and “The contract for long-term rentals is billed on a monthly base via tokens.” are added to the GPD.

Reference Model Selection / p.RMS

(mandatory) Select one (or more) appropriate reference model(s) as a template for the application model or an existing application model to be extended.

Objective The reference model is acting as a base layer for the later adaptation. It is often standardized and maintained externally and out of the hand of the enterprise’s software architect. The right choice of a reference model is essential for the success of the adaptation project. There can be more than one reference model involved, e.g., different models for various sub-domains. The aim is to find a suitable reference model (as described in Section 4.2.1).

The choice of possible reference models may be restricted due to standards or company-specific guidelines. Thus, this instruction *p.RMS* is dependent on the availability of choice. If there is a requirement of using a fixed reference model, this instruction can be skipped. Otherwise, this instruction will help to choose between several reference model options.

Input

- A set of possible reference models appropriate for the domain

Output One or more reference models with the right granularity and completeness.

Activities

1. Collect all possible domain models that
   - are considered as reference models, and
   - is relevant to the chosen boundary
2. Choose one or more reference models that are possibly suitable (dependent on whether the reference model choice is restricted, e.g., by a standard)
3. Make sure that every chosen reference model
   - serves the goals and boundary defined in instruction *p.AOD* (if possible),
   - has the right granularity with respect to its technical details, and
   - is complete with regard to its specified domain.
4. If necessary, limit the chosen reference model set (e.g., “choose one”)[8]
5. Add the meta data of the reference model to the GPD (e.g., name, id, version, date, author, source, the model itself); this may require to execute the *p.AOD* instruction

[8] An adaptation can also be made with multiple reference models (cf. Section 5.1.1)

On Extension If handling an already existing application model, including one or more adaptations, the set of possible reference models is probably limited to the reference models defined in the GPD. In theory, new reference models can be chosen, or existing ones can be discarded while updating the GPD file, if they do not conflict with the existing adaptations. In general, if the boundary of a new adaptation overlaps with an existing adaptation's boundary, the reference model from the GPD is a good starting point. If necessary, review the chosen reference models and update them accordingly.

Example The car rental enterprise "Road Trip Car" has selected a generic car rental reference model that is suitable to be used for the desired adaptation defined in the *p.AOD* instruction (in this example, the UML model in Fig. 4.3 is used as the reference model). There are no other reference models needed since the found reference model is suitable and can be used for the defined boundary in *p.AOD* ("long-term rental"). Moreover, this model is already used by other groups in the enterprise, which makes this reference model a good choice. In its original template state, the reference model does not cover any part of long-term rentals, so it becomes the object of an adaptation. The reference model gets added to the reference model set within the GDP file.

Background Process Selection / p.BPS

Select one (or more) appropriate background process(es) as behavior guidance for the application model.

Objective Similar to instruction *p.RMS*, the *p.BPS* instruction is used for the right adaptation implementation in the later execution phase. As described in Section 4.2.2, the background process guides the behavior-aware part of the adaptation, in contrast to the structural side considering the domain model. With the inclusion of processes, the contexts emerge their identity and meaning. Additionally, multiple background processes can be used. However, the background process is not mandatory for the adaptation.

It might be possible that, like in *p.RMS*, the background process is fixed due to a certain requirement for the adaptation (e.g., if the whole adaptation is only executed to implement a specifically new process). This can lead to a situation where this *p.BPS* instruction can be skipped or restricted.

Input

- A set of possible business processes, execution flows, or other related behavior descriptions

Output One (or more) background process(es) used for adaptation guidance.

Activities

1. Collect all possible process descriptions (in any form) that are relevant to the goals and intentions defined in *p.ATS*

2. Remove *syntactically* unsuitable process descriptions (e.g., wrong/non-useful type, inconsistent, incomplete)
3. Remove *semantically* unsuitable process descriptions (e.g., unimportant, other scope, too ambiguous or vague)
4. *(optional)* Remove process descriptions that are not covered by the boundaries
5. *(optional)* Add process descriptions that are not in the boundaries but still needed to include for the future application model
6. Formulate an explicit background process per process description identified
7. Prioritize the identified background processes into the category levels

   M (mandatory) processes that are necessary to include when implementing the adaptation

   O (optional) processes that can be used for temporal modeling but can be omitted

   I (informative) processes that are acting as hints for other processes but are not implemented further

8. Add the meta data of the background process(es) to the GPD (e.g., name, id, version, date, author, source, the model itself); this may require to execute the *p.AOD* instruction

On Extension Like selecting a reference model in *p.RMS*, the selection of relevant business processes as background processes should be based on an existing GPD, if possible. Any existing application model can benefit from the case that its already defined background processes are reused to strengthen their meaning and intention. Maintenance is more comfortable with a smaller amount of background processes within the GPD. Although adaptations are rather independent of each other, the GPD is shared. Thus, background processes (and reference models) can be used by every other present or future adaptation. If necessary, the GDP set of available background processes can be updated to meet the expectations. In general, it should be avoided to modify existing processes in the GPD without reviewing the related adaptations.

Example The car rental enterprise collects all its available business processes, which are described either in BPMN or as simple requirement texts (in this example, the enterprise uses (among others) the process shown in Figure 4.4). Next, the processes that are not suitable are discarded, e.g., the process on how to employ new employees. Other processes, like various payment method processes, are indeed crucial to the adaptation and could be added regardless of whether they are part of the boundary or not. The resulting process descriptions are considered to be within the boundary “long-term rental”. Finally, the stakeholders prioritize them into the respective categories (i.e., the example process gets the *M* priority). All background processes are added to the GDP.

Table 5.3 Context specification instructions

Name	Abbrv.	Description
Package	c.PAC	Specify and/or extend the adaptation packages.
Dynamic Context	c.DYC	Specify a dynamic context (scene).
Static Context	c.STC	Specify a static context (compartment).

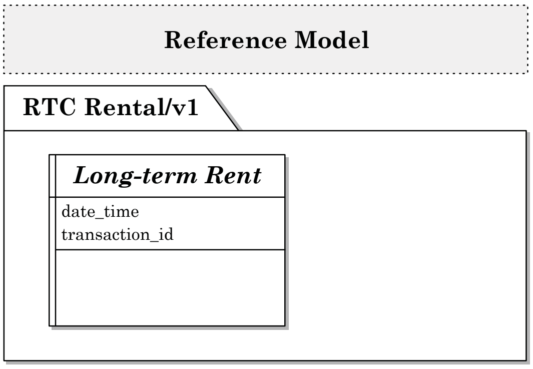


Fig. 5.3 The "Road Trip Car" (intermediate) application model example after implementing the context specification (including the *c.PAC*, *c.DYC*, and *c.STC* adaptation instructions); the reference model refers to the model in Figure 4.3

5.2.2 Context Specification

The context specification phase is required to set the environment for the adaptation's concepts. Especially to introduce roles, the contexts are needed in order to establish the meaning of the target application model's adaptations. An overview of the context phase's instructions is given in Table 5.3 It is recommended to start with *c.PAC*, as a package is most useful to group the individual adaptations within the project (cf. Section 5.1.1). Thus, it is often not needed to implement *c.PAC* more than once, if not necessary. However, if context elements are missing, it is possible to return to the context phase at any time (due to the iterative design of the model specification part, cf. Figure 5.2).

Package / c.PAC

Specify and/or extend the adaptation packages.

Objective The package concept (see Section 4.5.1), as a part of the extended BROS language concept set, is not required for modeling a BROS model. However, it is recommended to add a package for every individual adaptation implemented. Packages are an ideal construct to encapsulate all modifications that emerge with the adaptation. If no other adaptation is implemented (yet), the sole package represents the full application model, and is, therefore, of importance.

Input –

Output A boundary-specific package.

Activities

1. Generate a package (see Section 4.5.1) for the boundary
2. Specify the package with a name
3. *(optional)* Update the GPD according to the new package information

Example The "Road Trip Car" enterprise has no other adaptations within the application model, yet. Thus, the package shall represent the whole application model for now. The modeler creates the package according to the BROS language standard and simply names it `RTC Rental/v1` (cf. Figure 5.3). This package will contain all the future model elements that are going to be modeled with the adaptation implementation.

Dynamic Context / c.DYC

Specify a dynamic context (scene).

Objective A dynamic context is, within BROS, known as a scene (see Section 4.4.3). The scene defines a temporal frame in which actors (i.e., objects) are represented with a special structure, state, and behavior. To introduce a behavior-aware aspect into BROS, scenes are used in its models to define the context of roles. The context definition of scenes is not necessarily based on the reference model or background processes but reflect the adaptation expectations of the modeler and the enterprise.

Input

- *(optional)* Set of background processes
- *(optional)* Behavior and/or state description resources (e.g., processes, use cases, settings, user actions, tasks, activities)
- *(optional)* Adaptation statements from the GPD
- *(optional)* An existing scene

Output A specified scene as a dynamic context.

Activities

1. *(optional)* Identify a new scene (according to the adaptation statements)

   a. Determine a possible dynamic context:
      - Review the background processes for a possible dynamic context:
        - The whole background process
        - Related activities or sub-processes
        - Time-dependent states
        - Dependent and/or constrained relationships
      - Determine a state-like or process-like scene with the help of the adaptation statements, the boundary, and/or the behavior descriptions

   b. Check the following necessary conditions for the context:

   i. The context can be named (has an identity)
   ii. The context has a start and an end point in time (is temporal)
   iii. The context is not atomic (is not an event and may include participants)
   iv. The context can be differentiated from others (is unique)

   c. Check whether the identified context is part of the adaptations boundary and needed for the adaptation; if not, discard the context
   d. Include it as a scene into the application model (see Section 4.4.3)

2. Specify the scene
   - *(optional)* Discard the scene, if necessary
   - *(optional)* Evaluate whether the scene should rather be an event; if necessary, model the scene as an event (cf. Section 4.4.4 and the *b.EVE* instruction)
   - Set the scene name as identifier
   - *(optional)* Model the scene in its complete view: set the fields (e.g., `transactionID`, `runtime`) and operations[9] (e.g., `execute()`, `cancel()`) according to the syntax (cf. Section 4.4.3) and the adaptations statements
   - Set the scene in hierarchical relation with other already included compartments or scenes:
     – Side-by-side: the scene is on the same level as another context
     – Surrounding: the scene is parent another context
     – Within: the scene is a sub-context of another context

Example The modeler reviews the set of background processes that are assigned to the adaptation, starting with the priority *M*. It is searched for background processes that are aligned with the adaptations statements from the GPD (e.g., the "Pay Fees" background process). Further, the requirement of a long-term rental behavior is used as a dynamic context on its own ("Long-term Rent") since it is the new feature of the adaptation and considered as a temporal context for roles. It is a dynamic context regarding the rules from step *1b*. The context is then converted into the scene `Long-term Rent` (as a state-like scene). The model, including the scene, is represented in Figure 5.3. As an example of a process-like scene, a scene `Payment` could be implemented additionally. Another possible alternative would be a larger scene `Rental` that includes all possible processes of renting a car, including more (business) objects.

Static Context / c.STC

Specify a static context (compartment).

Objective The static context is the most used understanding of a context since it has no restrictions in terms of time. A static context can easily be known as a container or containing entities. In BROS, static contexts are known as compartments (see Section 4.5.2), based on the work of Kühn et al. [151]. Although it is not

[9] Please note that operations for scenes are possible but not recommended.

necessary to use compartments within the BROS adaptation (compartments are not even a part of the core concept set), it might be useful to cluster and separate several structural areas that affect the roles of entities. For example, the `Library` as a static context (i.e., compartment) can be used to define the role of a `Reader` for a `Person` (whereas both roles can be dynamic via events, nevertheless). The context definition of compartments is, like scenes, not necessarily based on the reference model or background processes but reflect the adaptation expectations of the modeler and the enterprise.

Input

- *(optional)* The reference model(s)
- *(optional)* Structure description resources (e.g., other diagrams and models, use cases, settings, goals)
- *(optional)* Adaptation statements from the GPD
- *(optional)* An existing compartment

Output A specified compartment as a static context.

Activities

1. *(optional)* Identify a new compartment (according to the adaptation statements)
   a. Determine a possible dynamic context:
      - Review the structure descriptions for a possible static context:
        – A whole structural model
        – Dividable parts of models or sub-models (e.g., packages, components)
        – Time independent states
        – Containing relationships and compositions
        – Interfaces, abstract classes
        – Real-world containing entities (e.g., buildings, teams, machines)
        – Patterns, frameworks, or other themed groups
      - Determine an invariant and stable compartment with the help of the adaptation statements, the boundary, and/or the structure descriptions
   b. Check the following necessary conditions for the context:
      i. The context can be named (has an identity)
      ii. The context is ubiquitous[10] (is not temporal)
      iii. The context is not atomic (is neither an object nor a role, and may include participants)
      iv. The context can be differentiated from others (is unique)
   c. Check whether the identified context is part of the adaptations boundary and needed for the adaptation; if not, discard the context
   d. Include it as compartment into the application model (see Section 4.5.2)
2. Specify the compartment
   - *(optional)* Discard the compartment, if necessary

[10] Of course everything has a start and an end; however, make sure that *this* context is omnipresent within the application and independent from any temporal effects.

- *(optional)* Evaluate whether the compartment should rather be a role; if necessary, model the compartment as a role (cf. Section 4.4.2 and the *s.ROL* instruction)
- Set the compartment name as identifier
- *(optional)* Model the compartment in its complete view: set the fields (e.g., `containerID`, `maxSlots`) and operations (e.g., `calculate()`, `openSlot()`) according to the syntax (cf. Section 4.5.2) and the adaptation statements
- Place the compartment in hierarchical relation with other already included compartments or scenes:
  - Side-by-side: the compartment is on the same level as another context
  - Surrounding: the compartment is the parent of another context
  - Within: the compartment is a sub-context of another compartment; please note that a compartment can not be placed inside a scene

Example The modeler reviews all information that is related to the structure. Especially the reference model's abilities, in combination with the enterprise's hierarchy models, are in focus. Although the modeler found several use cases for compartments (e.g., buildings, cash desks, consulting room, branches), it is decided not to include them, yet. The adaptation should be tackling the "long-term rental" boundary, and there is no necessity to use compartments in the modeler's interpretation of the adaptation statements in the GPD. Thus, the application model in Figure 5.3 remains unchanged. An alternative would be using a compartment for the rental building or the online software platform where the client could rent the car.

5.2.3 Structure Specification

The phase of the structure specification is mainly for introducing roles into the adaptation and the BROS model. The structure of a software system is the main aspect and goal of a BROS model: it represents the structure's foundational entities and their relationships in a context-dependent and behavior-aware manner. Table 5.4 lists the structure related instructions and their intentions. With the help of the structure-related instructions, the model gains its "core essence": the role-based structure that is needed to adapt the reference model. Ideally, the context specification was done before. However, since the structure specification is part of the iterative process, one can alternate between the specification phases or using roles without the context (not recommended).

In this phase, there is a difference between a progressive and regressive adaptation method variant (see Section 5.1.2), mainly in instruction *s.ROL* when creating a new role. The respective differences are highlighted with either the *[Regressive]* or *[Progressive]* keyword. Both variants lead to valid adaptations (however, some uses cases are easier or more difficult with either one). If unsure, choose the progressive method variant as it might be more understandable and often produces fewer errors.

Table 5.4 Structure specification instructions

Name	Abbrv.	Description
Role	s.ROL	Specify a role.
Association	s.ASC	Specify an association between two roles.
Constraint	s.CON	Specify a constraint between two roles.
Group	s.GRP	Specify a group for multiple roles.
Port	s.POR	Specify a port for a role within a context.

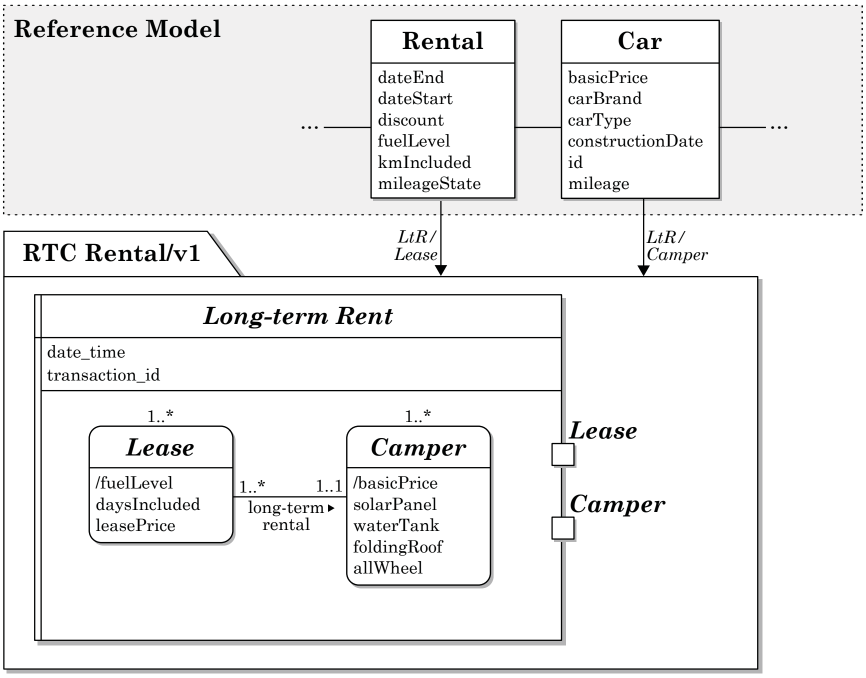


Fig. 5.4 The "Road Trip Car" (intermediate) application model example after implementing the structure specification (including the *s.ROL*, *s.ASC*, *s.CON*, *s.GRP*, and *s.POR* adaptation instructions); the reference model refers to the model in Figure 4.3 and only represents the important objects of it

Role / p.ROL

Specify a role.

Objective Roles are the most substantial part of BROS. They cover the structural part and are affected by the behavior-aware part as well. They serve as a bridge between the structural and behavioral side, between the domain (reference) model and the background processes. Introduced roles are acting as representatives and participants for an adapted object. The right choice, placement, and refinement of a role are essential for a powerful application model. Thus, the subsequent instructions are heavily dependent on the appropriate execution of this *s.OBJ* instruction. This instruction introduces a role (or modifies an existing one) for a selected object of the reference model to adapt it towards the enterprise's individual version.

Input

- Reference model
- *(optional)* Background process(es)
- *(optional)* Adaptation statements from the GPD
- *(optional)* An existing role

Output A specified role as a structural element of the application model.

Activities

1. *(optional)* Identify a new role
    a. Determine the role's identity, context, and fulfillment

        [Progressive]

        i. Review the reference model's available objects
        ii. Determine an object (e.g., *Person*) that is needed for the adaptation statements and the application model's boundary
        iii. Create an (empty) role as representative (cf. Section 4.4.2)
        iv. Locate the context in which the role should act as a representative of the object (e.g., `Payment`); add the role to the context in the application model
        v. Create an object fulfillment relationship between the fulfilling object and the new role (cf. Section 4.4.1)
            - Use ports for "entering a context" (cf. Section 4.5.3); if the role is placed in nested contexts, use multiple ports for access
            - Evaluate whether any other (existing) role can be used for a role fulfillment (creating a deep role, cf. Section 4.4.2); if necessary, use a role fulfillment instead of the object fulfillment
        vi. *(optional)* If the fulfillment relationship crosses a context border, a port can be used to access the nested role (see Section 4.5.3)

        [Regressive]

        i. Review the available contexts (scenes/compartments) within the application model; in this case, the whole model (possibly modeled as a package) counts as a (static) context, too
        ii. Select a context (e.g., `Payment`)
        iii. Determine a participant (an actor or a similar involved entity (e.g., *client*, *caravan*) that is not already existing in the selected context; if possible, use the background process(es) for help when identifying the context's acting entities
        iv. Create an (empty) role for the participant (cf. Section 4.4.2) and locate it in the context
        v. Choose for either object fulfillment or role fulfillment

- Determine an object from the reference model that is suitable to fulfill the role (e.g., a context-related or similar object) with regard to the adaptation statements[11]
- *(optional)* If the role can be fulfilled by another role from the application model (as a deep role) then evaluate if a role fulfillment is more useful than the object's fulfillment (cf. Section 4.4.2); if necessary, use a role fulfillment instead of the object fulfillment

vi. Set the fulfillment relationship (cf. Section 4.4.1 and Section 4.4.2) between the fulfilling entity and the role

vii. *(optional)* If the fulfillment relationship crosses a context border, a port can be used to access the nested role (see Section 4.5.3)

b. Check the following necessary conditions for the role:
- The role has no identity of its own
- The role is atomic (is not a container)
- The role emerges its meaning via the context
- The role is unique within its context

c. Check whether the identified role is part of the adaptation boundary and needed for the adaptation (with regard to the adaptation statements); if not, discard the role

2. Specify the role

a. *(optional)* Discard the role, if necessary

b. *(optional)* Evaluate whether the role should rather be a compartment; if necessary, model the role as compartment (cf. Section 4.5.2 and the *c.STC* instruction)

c. Set the role's name as identifier

d. Set the fulfillments for the role; make sure that the role is fulfilled by at least one suitable entity[12] (reference model object or application model role)

e. *(optional)* Discard not required fulfillments towards the role, if necessary

f. *(optional)* Model the role in its complete view:
- Investigate the fulfilling entity's fields and methods
- Set the fields (e.g., `id`, `engine`) and operations (e.g., `pay()`, `drive()`) for the role in order to perform its task within its context and according to the adaptation statements
- Set the role's indexes for fields and operations (cf. Section 4.4.2) with respect to the fulfilling entity

g. *(optional)* Set the role's multiplicity for the current context

Example The modeler investigates all information given by the reference model and the background processes. The enterprise's development department follows the progressive approach, thus, the modeler uses the reference model in combination with

[11] It is not recommended to choose *any* object but a relevant one. To lower the chances for inconsistent models, the relevant object should have the same intention as the role (e.g., a `Person` object for the `Client` role). Furthermore, object relationships should be noticed as well since they are still valid.

[12] For this, please follow the fulfillment steps from "Identify a new role."

the contexts defined in the application model. Two objects from the reference model are identified that are needed to adapt to the long-term rent use case: `Car` and `Rental`. E.g., the object `User` is not necessary since this object does not need to participate in the defined boundary[13]. Thus, both objects use roles as their representatives in the application model's context. According to the adaptation statements, the respective roles `Camper` and `Lease` are added into the scene `Long-term Rent` (cf. Figure 5.4). No other roles are used for now. Both roles are getting specified in their complete view regarding their fields and operations.

Association / s.ASC

Specify an association between two roles.

Objective Role associations are the central relationships between the two roles and state the interaction between them. The interaction of roles can be regarded as a special relationship between two objects because they are richer constructs and have fewer constraints as, e.g., UML associations. Instead of communicating with each other directly, both play roles in a particular context to interact in a very specialized (and adapted) way. This instruction introduces or modifies a role association (see Section 4.4.2) as an interaction between two given roles and is similar to the *s.RCO* instruction.

Input

- Two existing roles `A` and `B`
- *(optional)* Adaptation statements
- *(optional)* An existing role association

Output A specified role association element of the application model.

Activities

1. *(optional)* Identify a new role association
    a. *(optional)* Investigate the requirements within the GPD (e.g., the adaptation statements) for any defined association
    b. Determine the conceptual relationships between both roles
        - Are both roles on the same contextual level? Do they have to?
        - Who are the possible players at runtime?
        - Can both roles interact with each other?
        - Are there any inconsistencies?
        - What kind is the interaction based on? (knowledge, access, call, send, containment, . . .)

[13] Of course, the long-term rent also requires a person who rents the car. However, as a part of the "long-term rent" adaptation, the driver or user does not play any specific role but only the car (that has to be a camper) and the long-term contract (acting as a lease token). In contrast, if the adaptation boundary or context would include the activity of driving the car while long-term renting it, the driver would make sense to be included.

c. Check whether the association is similar to an existing relationship of the reference model (if so, the new association has to be a more strict refinement of the existing one)
d. Include the association into the application model between the roles `A` and `B` (cf. Section 4.4.2)

2. Specify the role association

a. *(optional)* Discard the association, if necessary
b. Check the refinement validity of the association:
   - the association is either a refinement of an existing relationship within the reference model, or
   - the relationship is new (there is no similar relationship within the reference model)
c. Set the following association properties (cf. Section 4.4.2):
   - A name as identifier, possibly with a reading direction indicator
   - One or two multiplicities and the end(s) of the association
   - *(optional)* A direction arrow (limiting the interaction direction)

Example The modeler investigates the two roles (`Camper` and `Lease`) again and found the interaction between them, according to the reference model. The role association is intended to state the exchange between the rented car and the related contract. Thus, the `Camper` role knows the related `Lease` role and vice versa. Next, the modeler models the multiplicities: within the scene are multiple (`1..*`) roles of `Lease` and `Camper`. However, via association, each `Camper` (`1..1`) can be assigned to multiple (`1..*`) instances of `Lease` tokens. There is now one remaining discrepancy: both roles can occur many times (`1..*`) within the context. However, the association only allows one assigned camper per lease token. But this is not an inconsistency since the multiplicity of the association ("How many campers are registered for the single lease token?", `1..1`) is different from the role's own multiplicity in the context ("How many campers may be long-term rented?", `1..*`). The new association is not intended to replace the existing relationship (in the reference model) but creates a new one (a relationship for long-term rentals instead of normal ones). Thus, there are no limitations in using the new association. After inserting the association into the application model, there are no more associations needed. The example's application model with the respective association is shown in Figure 5.4.

Constraint / s.CON

Specify a constraint between two roles.

Objective Role constraints are a basic feature to increase expressivity. The available role constraints (see Section 4.4.2) are used to set and restrict the fulfillment states between two roles. In that way, roles can be described in more detail. This instruction introduces a constraint between two roles and results in a role implication, equivalence, or prohibition. Please note that constraints are able to cross the borders of scenes and compartments as well as can make use of tokens (see *b.TOK* instruction and Section 4.5.5) to further specify their expressivity regarding individual constraints.

Input

- Two existing roles `A` and `B`
- *(optional)* Adaptation statements
- *(optional)* An existing role constraint

Output A specified role constraint element of the application model.

Activities

1. *(optional)* Identify a new role constraint
   a. *(optional)* Investigate the requirements within the GPD (e.g., the adaptation statements) for any defined constraints
   b. Determine the conceptual relationships between both roles
      - Are both roles on the same contextual level? Do they have to?
      - Who are the possible players at runtime?
      - Can both roles be played simultaneously? (necessity)
      - Are there any inconsistencies when constraining both roles?
      - What is the runtime meaning of the interrelationship of both roles?
   c. Include the constraint into the application model between the roles `A` and `B` (cf. Section 4.4.2)
2. Specify the role constraint
   a. *(optional)* Discard the constraint, if necessary
   b. Set one of the following role constraint types (cf. Section 4.4.2):
      - An *implication* when role `A` implies that role `B` is played as well
      - An *equivalence* when role `A` can only be played if (and only if) role `B` is played as well
      - A *prohibition* when role `A` can not be played if role `B` is played as well
   c. *(optional)* Set the constraint level (cf. Section 4.4.2) for the constraint
      - Use the *individual* level if both roles are only constrained when they are played by the same entity (this is the standard case)
      - Use the *global* level if the fulfilling entity is irrelevant concerning the constraint

Example The modeler investigates the available roles (`Camper` and `Lease`) for available and necessary constraints. However, there are no constraints required since there is no need for constraining the play of these roles. Thus, the application model remains unchanged (cf. Figure 5.4).

If, for example, another role like `Caravan` would take place in the scene as well, then a role prohibition constraint between the `Camper` role and the `Caravan` role would make sense: an object like `Car` could not play both roles at the same time, however, it can have two fulfillments to either role `Camper` and `Caravan` to be able to play one of the roles within the scene.

Group / s.GRP

Specify a group for multiple roles.

Objective Role groups, as part of the BROS extended model concept set (see Section 4.5.4), are used to cluster roles in a non-specific container. Thus, making groups different from scenes and compartments, the group can be used to multiple aggregate roles to, e.g., behave as a single one. Usually, role groups are neither dependent on the reference model nor the background process. This introduction identifies a set of roles that can be grouped, introduces the group, and specifies it. Nevertheless, role groups are not necessary for a valid BROS adaptation and may lead to inconsistent model states. It is recommended to use role groups only if there is a technical reason.

Input

- A set of roles
- *(optional)* An existing role group

Output A specified role group for a set of roles.

Activities

1. *(optional)* Identify a new role group
   a. Determine the possible common nature of the roles:
      - shared number of allowed instances, or
      - similar behavior (according to the relationships)

   b. *(optional)* Check whether the group can act as if it is a single role (validate the group properties for all roles within the group)
   c. Add a role group to the application model around the related roles (cf. Section 4.5.4)
2. Specify the role group
   a. Set a name as identifier
   b. Set the multiplicity for the group (cf. Section 4.5.4)
   c. *(optional)* Transform the incoming and outgoing relationships of individual roles within the group into aggregated relationships from and towards the group, if necessary (cf. Section 4.5.4)

Example The modeler has the two roles `Camper` and `Lease` in the model. Nevertheless, a role group would neither increase the “value” of the adaptation nor introduce any convenience regarding aggregated group relationships. Thus, there is no reason to include a role group for this model. The model remains unchanged (see Figure 5.4).

If, for example, a role `Caravan` would exist, a group for the roles `Camper` and `Caravan` can be used to unify them in their behavior regarding the associations.

Port / s.POR

Specify a port for a role within a context.

Objective The port concept is a part of the extended concept set of BROS (see Section 4.5.3). Ports are conceptual elements, representing a role and its accessibility. Usually, ports are used to access nested elements in more inner contexts (as introduced in instruction *s.ROL*). Therefore, this (rather trivial) instruction introduces ports as additional elements in the model, regardless of whether ports are already introduced by *s.ROL* or not. Additional ports can then be used to describe the structural relationships in more detail, especially the role fulfillment abilities of objects.

Input

- A role
- *(optional)* An existing port for the role

Output A specified port for a context within the application model.

Activities

1. *(optional)* Identify a new port for a role
   a. Identify the role's context and its existing ports
   b. Check the role's requirements by reviewing the adaptation statements
   c. If a new port is necessary, model the port on the contexts' edge (cf. Section 4.5.3)
2. Specify the port
   a. *(optional)* Discard the port, if necessary
   b. Set the ports name by using the role's name
   c. *(optional)* Set the requirements and constraints for the port (cf. Section 4.5.3)
   d. *(optional)* Set the port restrictions ("slots") according to the requirements (cf. Section 4.5.3, e.g., `[:2]`, `[(age>21):1]`)

Example The modeler checks the role requirements and available contexts. For the actual context of `Long-term Rent`, it is not necessary to introduce further ports since every role has its port already defined. Further, there are no more requirements for fulfillments for the roles. Thus, the application model remains as is (see Figure 5.4).

5.2.4 Behavior Specification

The phase of the behavior specification is not necessarily located after the structure specification phase, however, it is usually executed subsequently. While the structure specification allows for static constraints and elements, the behavior specification is used to implement temporal and runtime behavior. Especially events are in focus since they are the central concept for temporal behavior (of roles and scenes). Thus,

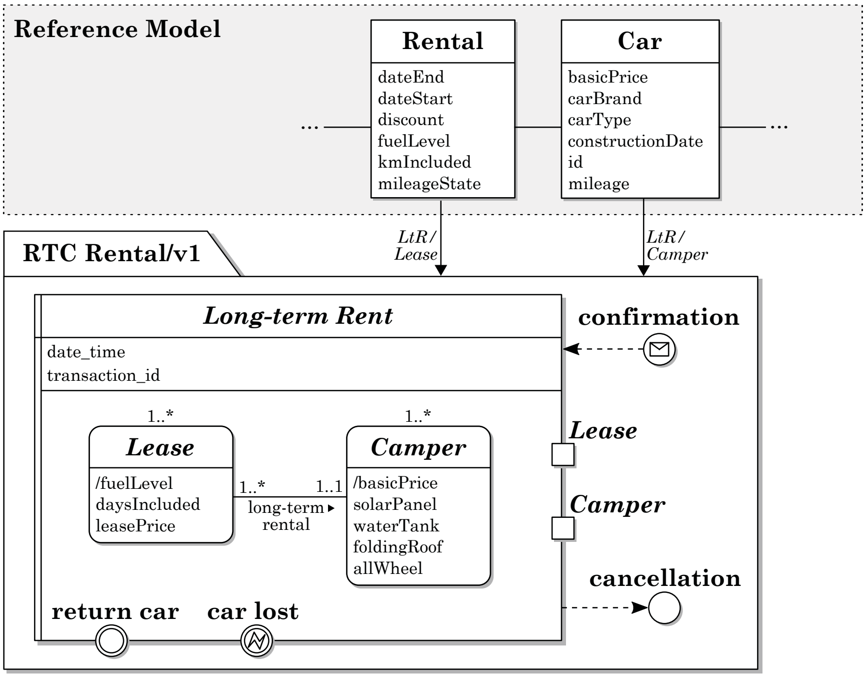


Fig. 5.5 The "Road Trip Car" application model example after implementing the behavior specification (including the *b.EVE* and *b.TOK* adaptation instructions); the reference model refers to the model in Figure 4.3 and only represents the important objects of it

the available background processes are of importance when modeling the behavior of the adaptation. In combination with scenes, which are dynamic concepts as well, the behavior specification can represent a business process and specify its effects on roles (and objects) within the model. Particularly when using various scenes, the behavior specification can orchestrate the temporal interconnection between them (e.g., the end of the first scene generates the start of the next scene). Table 5.5 summarizes the available instructions for the behavior specification phase. Often, the structure specification phase needs to be done first.

Table 5.5 Behavior specification instructions

Name	Abbrv.	Description
Event	b.EVE	Specify an event or return event.
Token	b.TOK	Specify a token for a model element.

Event / b.EVE

Specify an event or return event.

Objective The events are the main modeling construct for adding temporal behavior into the model. Although a BROS model can be modeled for adaptation purposes without any events, they are concepts of significance. A scene can not be modeled without an init event and return (or exit) event. Furthermore, roles can be improved in their meaning when coupled with temporal limitations regarding their playtime. This instruction states and specifies an event for a role or scene. Please note that including events as part of the adaptation is driven by existing roles and scenes but not the available processes. Events do not map the process flow but state the impacts of it on roles' and events' lifetime.

Input

- A role or a scene
- *(optional)* The background process set
- *(optional)* Adaptation statements
- *(optional)* An existing event

Output An event for a role or scene as an element of the application model.

Activities

1. *(optional)* Identify a new event for a role or a scene
   - Selecting a role
     a. Review the available background processes and/or adaptation statements for possible limitations of the role's fulfillment; start with the highest-ranked background processes
     b. Evaluate whether the role is affected by a surrounding scene's init, return or exit events; if this is the case, events of a role might be redundant
     c. *(optional)* Add an event for the role (cf. Section 4.4.4)
        i. Use a create (init event) or destroy (return/exit event) arrow as the event's effect
        ii. Place the event in the context that is responsible for triggering the event
        iii. Make sure that a dynamic role has at least one create and one destroy event; there can not be an event with only a create or destroy event
   - Selecting a scene
     a. Review the available background processes and/or adaptation statements for possible start and ending events of the scene; start with the highest-ranked background processes
     b. Make sure the scene has
        – at least one init event, and
        – at least one return or exit event
     c. *(optional)* Add an event for the start or end of the scene (cf. Section 4.4.4)

    i. Use a create (init event) or destroy (return/exit event) arrow as the event's effect
    ii. Place the event in the context that is responsible for triggering the event
- Check the following necessary conditions for the event:
  - The event represents a single point in time
  - The event is atomic (is not a container)
  - The event is on a global, model-wide timeline
  - The event is unique within its context
- Check whether the identified event is part of the adaptation boundary and needed for the adaptation (with regard to the adaptation statements); if not, discard the event

2. Specify the event

   a. *(optional)* Evaluate whether the event should rather be a scene; if necessary, model the event as a scene (cf. Section 4.4.3 and the *c.DYC* instruction)
   b. *(optional)* Discard the event if necessary
   c. Set the name of the event as identifier
   d. Set necessary create and/or destroy relationships towards roles and scenes that are affected by the event (cf. Section 4.4.4)
   e. *(optional)* Set the event's spec (cf. Section 4.4.4)
   f. *(optional)* Set the cause of the event towards an available role (cf. Section 4.4.4)
   g. Specify the event's style by the nature of its occurrence (cf. Section 4.4.4 and Table 4.6)

Example The enterprise's modeler goes through all scenes and roles available within the model, yet. It is identified that the scene `Long-term Rent` misses an init and return (or exit) event. Thus, while investigating the background processes with priority *M*, the event `confirmation` is modeled as the start of the rental (with respect to the process in Figure 4.4). As the possible endings, the return events `return car` and `car lost` are modeled as points in time in which the context ends itself (that is, it is completed and terminated in its activities and roles). As an external ending event, `cancellation` is used to interrupt the contract[14]. Since both available roles are within the scene, they do not need additional events for start and end (they do not have different start and endpoints in time compared with the scene). Finally, event styles (cf. Section 4.4.4 and Table 4.6) are assigned to the existing and new events: (a) the `confirmation` event is a message style, since it is based on a BPMN message event, (b) the `return car` and `cancellation` are standard style as they are not further specified in the background processes, and (c) the `car lost` event is assigned with the error style as it is not modeled in any process flow but is a "real error exception." The events' specs (stating more details) are added internally. The model in Figure 5.5 represents the application model, including the newly introduced events.

[14] Please note that not all events are represented in the example background process from Figure 4.4. Events are usually generated from a number of different background processes.

Token / b.TOK

Specify a token for a model element.

Objective The token concept is a very different concept and, compared to other BROS concepts, hardly defined in its details. In general, the token is used to identify an object's runtime identity (see Section 4.5.5). There are semantics and syntax to create, assign, and to use a token; all three token types should be used with care when modeling with BROS, especially when adapting a reference model. With the help of tokens, one can state that a certain object's identity is used for, e.g., a port. The assigned (or created) tokens can then be used within a scene or compartment to limit the participants for, e.g., an association. This instruction introduces a token element for the application model. It is assumed that the modeler already knows which element is going to be defined by a token (due to the uncertainty of the token's nature in the concrete application model). It is not recommended to execute this instruction without any good reasons, e.g., to solve temporal inconsistencies.

Input –

Output A token for a specified model element within the application model.

Activities

1. *(optional)* Identify a new token
   a. Select an application model element that needs to be constrained by a token (i.e., a model element, that gains its validity when using a specific identify, like a port access, an association end, a fulfillment)
   b. Place a token into the application model
   c. Evaluate whether the token is used, assigned, or created (cf. Section 4.5.5)
      - *Use token*: use an identity provided by another token; make sure that the token's identity is made available in this context (e.g., via a port)
      - *Assign token*: use the token to state a newly used identity (but do not create an identity); usually, this token is used in combination with a fulfillment arrow since this concept binds an object identity to a role
      - *Create Token*: like *assign*, however, the identity is created during the execution
2. Specify the token
   a. *(optional)* Delete the token, if necessary
   b. Set the token label as identifier
   c. *(optional)* Set the token's type (e.g., <1>, ≪1≫, or ≪1*≫; cf. Section 4.5.5)
   d. Check the model's integrity, especially within the token's context

Example Due to the small model size, there is no need for tokens. Further, to avoid unnecessary inconsistencies, all adaptations are made independently from the specific fulfilling object's identity. The model remains as is, without tokens (cf. Figure 5.5).

Table 5.6 Review instructions

Name	Abbrv.	Description
Verification Check	r.VER	Verify the model syntax according to the BROS language and method rules and guidelines.
Validation Check	r.VAL	Validate the model semantics according to the adaptation goals and statements.
Feedback Loop	r.FBL	Utilize a feedback loop.
Documentation	r.DOC	Document the adaptation process.

5.2.5 Review

The phase of the review is the counterpart to the preparation: the once defined statements and goals are checked for success. Further, the designed model is checked via verification and validation for its technical and conceptional correctness. Further issues in this phase regard the reporting and documentation of the implemented adaptation process. The summary of the review's instructions is given in Table 5.6. The review phase is, as the preparation phase, dependent on the given "environment" of the adaptation, e.g., the enterprise's modeling culture, the necessary requirements, or the purpose of the model. In a professional adaptation environment, this phase might be implemented in more detail (compared to, e.g., a more innovation-like experimental setup).

In this section, the "Example" category is rather an explanation than a real implementation, as the application model (see Figure 5.5) is built according to the standards in mind. However, some issues are called and improved to demonstrate the individual review instructions.

Verification Check / r.VER

Verify the model syntax according to the BROS language and adaptation method rules and guidelines.

Objective The designed model has to be syntactically correct in order to be maintainable and useful. Especially when using the final application model for more than documentation purposes. This verification instruction is used to check the generated model against the expected syntax defined by BROS. As a result, the model is successfully checked for compliance with the BROS language standard.

Input

- The current adaptation of the application model
- The BROS language standard

Output The *verified* label for the current adaptation

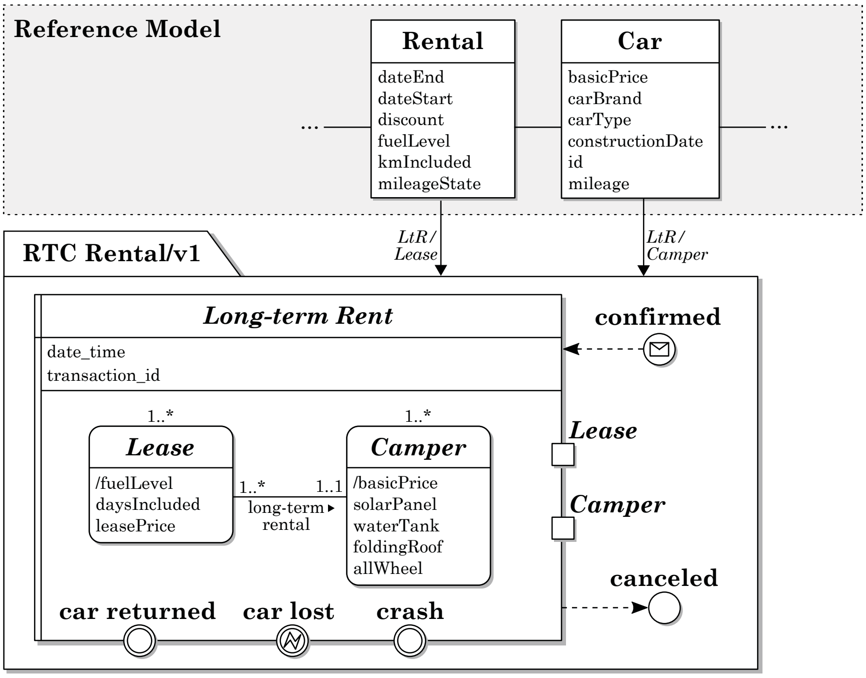


Fig. 5.6 The "Road Trip Car" application model example after implementing the review phase (including the *r.VER*, *b.VAL*, *r.FBL*, and *r.DOC* adaptation instructions); the reference model refers to the model in Figure 4.3 and only represents the important objects of it

Activities

1. Check the model's completeness
   - All necessary reference models involved
   - A set of used background processes
   - The actual adaptation as a set of used BROS language model elements (possibly within a package)
   - Other adaptations and external model parts used
   - *(optional)* The GPD data
2. *(optional)* Investigate the style guidelines (see Section 6.1) of BROS and check for truth
3. Investigate the verification rules (see Section 6.2.1) of BROS and check for truth
4. If every check is passed successfully, mark the model adaptation with the *verified* label in the GPD
5. *(optional)* If the checks have failed, return to the *model specification* phases: refine the context (cf. Section 5.2.2), structure (cf. Section 5.2.3), and behavior (cf. Section 5.2.4) specification to pass the verification checks

Example After the model finished the specification phases, the review phase is used to determine the accuracy and correctness with respect to semantic and syntax. The *r.VER* is used to check the final application model's syntax and style. After passing all checks stated in the *style guide* (see Section 6.1) and the *verification rules* (see Section 6.2.1), some corrections are made (e.g., the exit event `confirmation` is named into `confirmed`, and `cancellation` named into `canceled`). The new and corrected model version is stated in Figure 5.6.

Validation Check / r.VAL

Validate the model semantics according to the adaptation goals and statements.

Objective The validation instruction is a follow-up process for checking the model's consistency. In contrast to the *r.VER* instruction, this *r.VAL* instruction reviews the semantics of the model on a conceptual scale. Implementing this instruction should lead to a consistent model that is semantically tailored and meets the individual goals and needs.

Input

- The current adaptation of the application model
- The used reference model(s)
- The adaptation statements, boundary, and goals

Output The *validated* label for the current adaptation

Activities

1. Check whether all concept specifications are done (context, structure, and behavior specification with related instructions)
2. Investigate the validation rules (see Section 6.2.2) of BROS and check for truth
3. If every check is passed successfully, mark the model adaptation with the *validated* label in the GPD
4. *(optional)* If the checks have failed, return to the *preparation* phase (cf. Section 5.2.1): refine the goals, boundary, and adaptation statements to pass the validation checks

Example The "Road Trip Car" enterprise uses the *r.VAL* instruction to check the model for semantic mismatches with the help of the *validation rules* (see Section 6.2.2). An issue based on the background processes was detected: a return event `crash` is needed. Thus, the event is modeled[15] into the `Long-term Rent` scene (see Figure 5.6). Due to the small model size and the conscious adherence to the rules during the modeling process, the model seems to be free from any further misconceptions regarding the semantic correctness (e.g., the reference model's interpretation is

[15] Please note how this event is not modeled with an error style but a standard event. This is because the crash is modeled in a background process and not unexpected regarding the business processes.

maintained, the adaptation is as small as possible, contexts are well-placed, and events have their coupled point in background processes).

Feedback Loop / r.FBL

Utilize a feedback loop.

Objective A feedback loop is a tool for tracking and reporting misconceptions of the reference model to its owner (see Section 5.3). A feedback loop ensures that occurring issues are routed back to the responsible persons that maintain the reference model. However, the feedback loop works best for internal and enterprise-specific reference models as the chances to change the reference model are higher than changing an industry-standard reference model. This instruction makes sure that the fundamental misconceptions found are reported back to the reference model owner.

Input –

Output A list of reported reference model misconceptions via the feedback loop.

Activities

1. Identify possible misconceptions (with respect to the reference model, see Section 5.3) during the adaptation, e.g.,
   - Missing objects
   - Missing relationships (e.g., associations, aggregations, inheritances)
   - Unnecessary and/or inadequate abstractions
   - Insufficient, inconsistent, and/or not logical statements
2. Choose the reporting channel of the respective reference model (e.g., email protocol, user survey feedback, direct messaging) to report detailed information about the issues and misconceptions

Example The modeler has, while adapting the reference model, no issues with the reference model itself. If any misconception or missing feature of the reference model would be encountered, the stakeholders know the feedback loop channel to directly address the owner (a public ticket system for model maintenance). If the reference model would be in-house, the channel would possibly be shorter and done via direct communication (and documentation).

Documentation / r.DOC

Document the adaptation process.

Objective The process of adaptation consists of a lot of assumptions and decisions made by the modeler, often with a good reason. Nevertheless, documentation of the work is a priority task that does not have to be neglected [63, 81, 140]. When adapting a reference model with BROS, the GPD is a central repository for the project's information and meta-information. The documentation instruction takes care of the task to document all necessary information related to the adaptation made.

Input The current adaptation within the application model

Output A set of structured information for the GPD as the adaptation's documentation

Activities

1. Gather the necessary information about the current adaptation
2. Structure the information data (if possible, in compliance with the already existing documentation
3. Add the documentation (or a reference) to the GPD

Example As the last step, the whole adaptation is documented appropriately within the GPD. All missing information is added and updated. The modeler decides to include the model element's description in the GPD as well since the model is rather small. Each element gets information assigned to explain its intentions, uses, and technical specification.

5.3 Feedback Loop

The feedback loop is one crucial mechanism when implementing the BROS adaptation method. The BROS adaptation method has a significant drawback: due to its non-invasiveness and additive nature, it is not possible to solve misconceptions and non-adaptable parts of the reference model in every case. The feedback loop is used as a tool to provide feedback to the reference model owner since the BROS adaptation method is restricted in some adaptation cases (see Figure 5.7).

For example, the BROS method does not allow to introduce new business objects but only adapt to existing ones. Thus, either there is an adequate reference model object that can be adapted with a role, or the reference model is not sufficient enough for the current use case. The standard case would be to change the reference model or add a new one to the current project. If, however, the reference model can not be changed to a more suitable one (e.g., it is set by an enterprise's policy), there must be a possibility to "contact" the reference model owner to request a new version. The

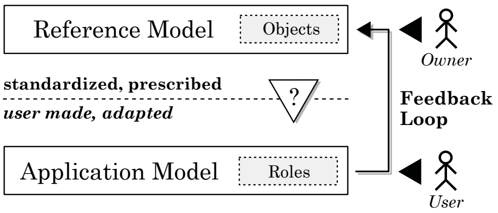


Fig. 5.7 The feedback loop as instrument for a user's requests and feedback for the reference model owner

request[16] can be made by, e.g., simple messages or established update mechanisms (e.g., a shared and internal *GIT*[17] repository).

The range of the requests done via the feedback loop can be diverse: a new object, other fields or operations of objects, new or changed associations, or simply a rename of a given entity. While using the reference model as a basis for the individual adaptation, the misconceptions may lead to unsolvable adaptation states (e.g., if the car rental reference model from Figure 4.3 would miss the needed `Contract` object but summarizes the accounting via the user's `Account` object). Instead of changing the reference model, the feedback loop should be used to report a request for a change.

Particularly for an internal reference model, it is essential to provide an easy-to-use but detailed feedback loop channel that can be used by the modelers and developers for improving the possibly following versions of the reference model. Only with feedback from the users' side, the reference model owner can act and react to their needs and improve the next published version of the reference model as an evolved reference model. Even if the case is a reject of the request done via the feedback loop, there is (and should be) a good reason provided to decline the request, which then can be used by the modeler to model the use case in other ways.

5.4 Best Practices and Limitations

The BROS adaptation method comprises a set of instructions divided into five phases. The objective of the BROS method is to adapt a given (OO) reference model towards an individual, enterprise-specific application model with the help of roles.

The BROS adaptation method is most useful for the early software development stack. In combination with requirements engineering and software architecture, the BROS adaptation method can help identify the individual needs of the enterprise. The combination of structure and behavior within the adaptation can bridge the gap between stakeholders, especially software architects, engineers, and business analysts.

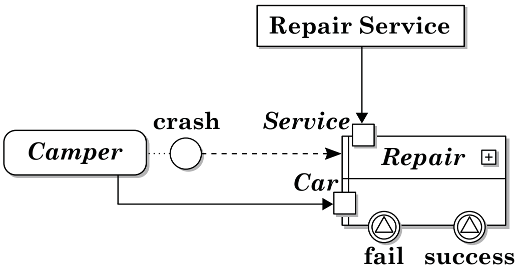


Fig. 5.8 Another (external) scene as a possible extension to the example "Road Trip Car" application model

[16] It is, nevertheless, more likely to request a change of a reference model when it is not standardized by an external group but from the internal side of the enterprise itself (since the internal reference model is intended to be used by the enterprise's IT members).

[17] `https://git-scm.com/`

By applying roles to predefined reference model objects, the reference model is left untapped, and the application model is still a valid representation of the individual needs (structure and processes). It works best with smaller parts of the main reference model since the four main model elements (objects, roles, scenes, events) can be too much when applying them to the whole model at once. For that, the BROS adaptation method allows adapting single parts of the reference model (the *boundary*, see instruction *p.AOD*). For example, the example application model from the instructions (see Figure 5.5) can possibly be extended with the BROS construction stated in Figure 5.8 to include a general repair service utilized to repair a car with a `Camper` role together with a specialized `Service` role treatment. Such partly adaptations can be used to extend the application model incrementally, mainly driven by the requirements.

The loose coupling of the adaptation also supports easier maintenance when a reference model evolution occurs. The whole adaptation is connected via the role fulfillment concept, which makes it easy to set up new (or change existing) role fulfillments after a reference model evolution.

The BROS adaptation method has two main design paradigms:

- the iterative design, and
- the modular design.

The former one is used as five phases in which the middle part (context, structure, and behavior specification) is not a linear process but an iterative design that allows remodeling and reacting on new decisions. The latter paradigm is used to be more flexible for future extensions: the instruction-based design supports the possibility to introduce new (or modify existing) instructions in order to evolve the BROS adaptation method appropriately.

The adaptation method described above (the instruction-based user manual) is intended to work as a guideline. Furthermore, the adaptation method is, after all, a user manual to help to adapt the reference model in a sorted and systematic manner. Is can not be formalized or automated since it depends on design decisions that need to be taken professionally. Also, the unified manual can not cover all possible adaptations of every reference model. It is assumed that the reference model has the right OO-based granularity and is complete within its domain. Although the feedback loop can help to overcome these issues, the instructions can not solve them right away.

Chapter 6
Modeling Practice

"Modeling guidelines will also save you time by limiting the number of stylistic choices you face, allowing you to focus on your actual job: to develop software." – [11, p. 1]

Abstract A valuable model is often more than a technically explicit representation of a domain. A good model takes advantage of a reviewed (that is, validated and verified) and readable layout. These "secondary" properties are often neglected and may lead to non-used models, although they might be syntactically complete. The BROS language and the related adaptation method are specified in Chapter 4 and Chapter 5. However, a good BROS model consists of more than the sum of its elements needed for adaptation. Although the previous chapters introduce the language and the adaptation steps, the "how to model" can still be improved: choosing the right model style. Further, the resulting BROS model needs to be checked against various adaptation verification and validation rules in order to review the correctness of the modeled domain. This chapter proposes a set of sections for BROS modeling practice to increase the model's value.

6.1 Style Guide

There is much literature on the subject of model style guidelines (e.g., [11], [242]). Since the BROS language is based on UML [187] and CROM [151], the recommended styles should be in relation to the general modeling style of UML models.

The BROS style guide differs from the BROS model validation (syntax check): while the style guide consists of recommendations only, the validation check is a set of rules that must be adhered to. In contrast, the rules stated in the style guide are guidelines: they are optional and can be applied and adapted individually.

> "A style guide is not universal: each project manager should be able to customize his/her set of rules according to specific needs." [123, p. 290]

By following the set of the style guidelines, the modeler has indeed fewer choices and decision points to build the model [123]:

> "A style guide defines a set of rules that any model must conform to. Style guides reduce the number of acceptable models and force developers to make models owning the wished properties, which results in smaller sets of licit interpretations" [123, p. 294]

However, style guidelines are also a helpful resource to generate simple, clear, and beautiful models. This has a direct effect on other stakeholders [179] as the model will probably be used and understood more often.

> "Models depicted with a common notation and that follow effective style guidelines are easier to understand and to maintain. These models will improve communication internally [. . .] and externally [. . .]." [11, p. 1]

As an overview, Hindawi et al. [123] explicitly state the following areas in which the style guidelines can be classified: *methodology*, *common methodology*, *consistency*, *modeling style*, *completeness*, *good practice*, *conventions*, *architecture style*, *refinement*, and *specification gap*. However, to reduce complexity and improve the applicability, the style guidelines proposed by this thesis are only divided into three different natures:

- the *general modeling* styles,
- the *individual element styles*, and
- styles for a *model's adaptation*.

As the style guidelines are "semantically meaningless" and only increase the overall syntactical appearance, they are, in this thesis, purely optional but recommended. In general, the style guidelines are often trivial and easy to implement. However, if the guidelines are implemented in a complete and integrated way, they will give a sound overall picture of the model.

In particular, the individual element limitations and restrictions (set by the BROS language and adaptation method) are not considered in the ruleset: those constraints are not optional and already included in the BROS concept and method description in Chapter 4 and Chapter 5 (e.g., "a role has a name" or "a scene always has a start and endpoint in time as events"). Instead, the guidelines are additional statements that lead the development of a BROS model as a secondary resource and are not included in the BROS specification itself.

In this thesis, the guidelines are represented as a basic, limited set of the most important guidelines[1] (and, thus, more abstract and general) to maintain the right balance of details, complexity, and extent. Since applications of BROS are still rare in the industry, the style guide (i.e., the individual guidelines) is based on experiences of previously modeled use cases as well as a recommendation from literature. Therefore, missing or incomplete guidelines may occur but may be added to the set in the future.

[1] Please note that the set can be subject to future extensions.

6.1.1 General BROS Model Style

Style guidelines for general BROS modeling are affecting the whole model and the structure of the model itself. They contain the categories of the model's *readability*, *simplicity*, *naming*, and *general guidelines* [11].

Due to the close connection of the BROS language and UML, several of the UML guidelines from Ambler [11] are used as a reference and partly adapted for the general BROS modeling style guidelines (those are marked with the [11] reference). Nevertheless, other works propose similar style guidelines as well (e.g., [155]).

Readability

The readability of a model is the quality of understanding the model in the way it is represented and illustrated. The model should be easy to read and should "please the eye" in order to grasp the content quickly and easily. The related guidelines help to increase the readability of a BROS model by defining the layout of various model elements and structures.

Guideline 1 *The model is in a rectangular shape, has enough space between the single elements, and the elements are distributed symmetrically.*

Guideline 2 *The elements of the model are aligned with each other in terms of location and size. [11]*

Guideline 3 *Crossing, non-horizontal, and non-vertical lines/relationships are to be avoided (constraints are optional). [11]*

Guideline 4 *Crossing relationships are indicated with arcs on their crossing (preferentially horizontal); crossing labels should be avoided. [11]*

Guideline 5 *Text labels share the same font, are placed horizontally, and must only be used in a few (~ 2-3) different sizes.*

Guideline 6 *Names of elements are positioned in the middle of their frame, inner elements of entities are left-aligned.*

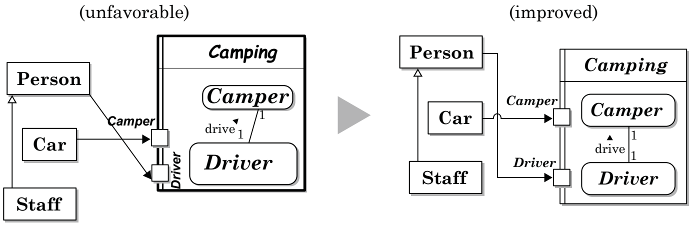


Fig. 6.1 Model readability improvement example in an unfavored (left) and improved (right) way

Guideline 7 *Frames and symbols that do not add any value to the model are to be avoided. [11]*

Guideline 8 *Size and width of lines and borders are the same in the whole model.*

Simplicity

The simplicity quality is meant to increase the understanding of a model by its used semantics[2]. Simplification of the used semantics (and its related syntax) of the model results in reduced complexity (which, in turn, increases comprehensibility). The guidelines listed here are meant to help ensure that the greatest possible simplicity is combined with the necessary complexity.

Guideline 9 *The number of elements used (and their element types) is reduced to a necessary minimum.*

Guideline 10 *Well-known structures (e.g., patterns) are used before new inventions.*

Guideline 11 *Elements are only represented in full-view when needed (otherwise collapsed).*

Guideline 12 *Large models are divided into possible smaller ones. [11]*

Guideline 13 *Only required domain content is represented in the model.*

Guideline 14 *Detailed specifications of elements (e.g., arrowheads, multiplicities) are only used when they are definitely true and necessary. [11]*

Guideline 15 *Scaffolding elements and inner elements (e.g., getters, setters, keys, technical operations, and fields) are not shown, if not needed. [11]*

Guideline 16 *Actors and pure data items are not to be modeled differently than a normal role (or object).*

Guideline 17 *Universal and "almighty" elements should be avoided and divided into multiple elements with different concerns.*

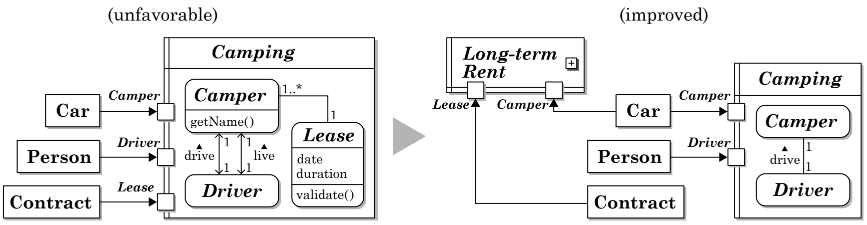


Fig. 6.2 Model simplicity improvement example in an unfavored (left) and improved (right) way

[2] In contrast to readability, where it is increased by the model's presentation.

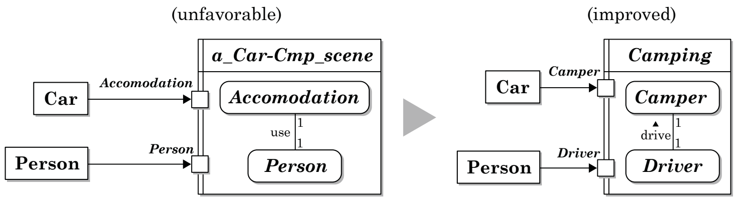


Fig. 6.3 Model naming improvement example in an unfavored (left) and improved (right) way

Guideline 18 *Generally assumed or obvious specifications are omitted. [11]*

Guideline 19 *Implicit relationships are not modeled. [11]*

Guideline 20 *The visibility of fields and operations is not shown.*

Naming and Terminology

The area of naming is tackling the right choice of labels and terms for the model's element names. It does not cover the right choice of fonts but the correct terminology to identify individual elements within the model. The following few rules take care of a thorough naming of model elements.

Guideline 21 *Elements are named in a concrete and precise way related to their purpose and task. [11]*

Guideline 22 *The available domain terminology is used before the own concept names. [11]*

Guideline 23 *Element names are done in a consistent way (e.g., naming conventions, tense). [11]*

Guideline 24 *If using time and temporal specification values, the unit measurement (e.g., $t = milliseconds$) and/or the date layout (e.g., $mm : hh\ dd : mm : yyyy \pm t$) is stated accordingly; this holds for other units as well*

General Directives

The last category is called general directives. It contains guidelines that do not belong to the other categories but still affect the whole model and its style. The guidelines of the general directive category are more generic and abstract, however, they can often be implemented easily.

Guideline 25 *Unnecessary colors are to be avoided (besides shades of gray).*

Guideline 26 *Entities have a small, gray shadow and a bold font.*

Guideline 27 *Temporal elements are aligned and related from left to right.*

Guideline 28 *Unknown parts are indicated with a question mark (?). [11]*

Guideline 29 *Incomplete parts are indicated with an ellipsis (. . .). [11]*

Guideline 30 *Notes (attached to elements) are used to further describe the model's content. [11]*

Guideline 31 *The general UML guidelines (class diagram) are used for BROS modeling, unless otherwise specified in other guidelines.*

Guideline 32 *The model's appearance is subordinated to its content. [11]*

Guideline 33 *User-made extensions and changes of the BROS language and method specification have to be documented and delivered together with the model.*

6.1.2 Individual BROS Language Element Style

Besides guidelines affecting the whole BROS model, there are guidelines for a consistent and aligned implementation of single BROS language elements within the model. In contrast to the BROS model guidelines, the categories of the BROS language element guidelines are divided by the main elements *objects*, *roles*, *contexts* (that is, scenes and compartments), *events*, and *relationships*. The degree of extent, detail, and granularity differ from one category to another, as the different elements are more flexible than others. Still, the BROS element guidelines are not considered as rules: they are only optional and recommended (if applicable and to the extent that it can be implemented).

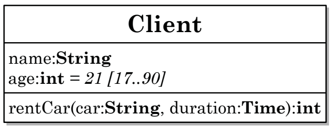
Objects

The object elements (see Section 4.4.1) are the constant and static model elements, often provided by the reference model (respective the domain side). There are only a few adaptation-independent guidelines available for objects since they are relatively fixed in their syntax and usage (for object adaptation guidelines see Section 6.1.3).

Guideline 34 *Object names are always derived by the domain's business objects (or the given reference model, if possible).*

Guideline 35 *Objects that do not participate in the model as a fulfilling entity should not be shown.*

Guideline 36 *The relationship details between objects are minimized.*



Roles

An application model's roles (see Section 4.4.2) represent the objects of the domain (reference) model side. In contrast to objects, roles are not limited to any template and can be designed freely, especially when using roles in an adaptation-independent use case (that is, if the BROS application model is designed from scratch without template). The general guidelines for roles in such a case are of a very basic nature.

Guideline 37 *Roles use singular nouns as names.*

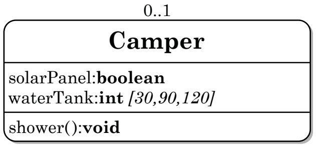
Guideline 38 *Roles are preferentially placed in contexts.*

Guideline 39 *Similar roles are grouped with a role group.*

Guideline 40 *The use of deep roles is minimized.*

Guideline 41 *Roles are named after the related task, not the respective item type or job position. [11]*

Guideline 42 *For each role within a context, at least one port is used; a port's label is positioned outside.*



Contexts (Scenes and Compartments)

The model's contexts, namely scenes and compartments, are used to encapsulate roles and give them their meaning (see Section 4.4.3 and Section 4.5.2). They act as containers in the model structure and should follow the simple guidelines below.

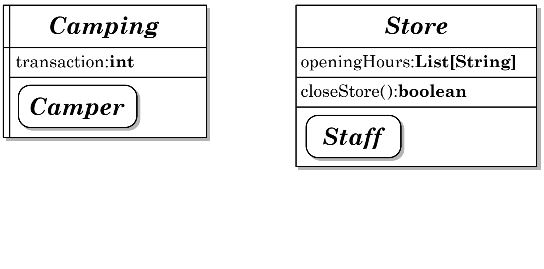
Guideline 43 *Names of process-like scenes use a verb in the present participle tense (possibly with a noun); state-like scenes and compartments use a single noun.*

Guideline 44 *Scenes (and compartments) are only used when they contain at least one role (otherwise, the scene could be an event itself).*

Guideline 45 *Operations of scenes are to be avoided.*

Guideline 46 *Scenes and compartments are only nested when definitely necessary.*

Guideline 47 *Only the horizontal header separation line crosses the vertical-left line of the scene's shape.*



Events

Events are the central concept for representing time in a BROS model. (see Section 4.4.4). Thus, the right placement and layout of events are of importance. However, events are the simplest elements (regarding their shape and general specification) and do not require many guidelines for an optimal event layout experience.

Guideline 48 *Event names use a verb in either simple past or present tense, possibly with an associated noun.*

Guideline 49 *Events are placed in the most outer context possible. (with respect to their effect).*

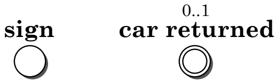
Guideline 50 *Event effects (create/destroy) that cross the boundary of their context are minimized.*

Guideline 51 *Return events are positioned at the bottom or right border of scenes and do not have any destroy-relationships.*

Guideline 52 *Exit events are to be avoided.*

Guideline 53 *Event causes are only indicated when necessary.*

Guideline 54 *Event labels are positioned above (or below if necessary) of the events' shape; return events labels are placed within the scene.*



Relationships

Relationships in BROS establish the business logic between roles. The term includes every "line" between two entities (e.g., associations, constraints, create/destroy). For a clear and understandable model, the suitable placement of relationships is pursued.

Guideline 55 *Associations always carry a label.*

Guideline 56 *Generic names for relationships are to be avoided (e.g., has, uses, contains), and strong, meaningful verbs are used instead.*

Guideline 57 *Associations crossing the context borders are minimized.*

Guideline 58 *Role groups are preferred over constraints.*

Guideline 59 *Constraints are preferred over tokens.*

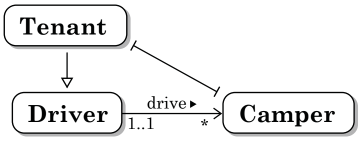
Guideline 60 *Direction arrows of relationship labels (◄ and ►) are only used if necessary and for reading purposes (not as relationship navigation).*

Guideline 61 *The use of constraints is minimized.*

Guideline 62 *No directly attached relationships end at the rounded corner of a role.*

Guideline 63 *Relationships are not too close together and easy to follow.*

Guideline 64 *The relationships always attach precisely to the ending elements (constraints are optional).*



6.1.3 BROS Adaptation Method Style

The last guidelines group is those which affect the adaptation of a reference model due to a BROS adaptation method implementation (see Chapter 5). The guidelines for adaptation are neither applied per se when modeling nor always applicable (as they depend on the scope and extent of the adaptation). The adaptation guidelines help with finding the right path of a reference model adaptation. Often, they are additionally stated in the process of the BROS adaptation method. The guidelines are again purely optional and can be applied within the adaptation process, if possible. Nevertheless, for the adaptation process implementation (in particular for the review phase, see Section 5.6), the validation and verification rules are more prominent (see Section 6.2).

Guideline 65 *An adaptation is encapsulated in a package.*

Guideline 66 *Used content from other adaptations is annotated accordingly.*

Guideline 67 *Primarily used objects are positioned prominently around the model.*

Guideline 68 *Complete re-adaptations (e.g., via a purge index) are to be avoided.*

Guideline 69 *Events should have their style according to the related background process activity.*

Guideline 70 *Roles are representing the object as a non-technical, real-world entity.*

Guideline 71 *A role may have the same name as the object (but still only occurs once in a context).*

Guideline 72 *The represented objects are labeled by their reference model source and version.*

Guideline 73 *Role indexes are sorted before un-indexed fields and operations.*

Guideline 74 *Similar return events, which are not needed any further, should be combined.*

Guideline 75 *If unsure, details are left out of the application model adaptation.*

6.2 Adaptation Consistency

The consistency of models is a different subject than the style guidelines. Instead of optional rules that are semantically meaningless and only increase the overall appearance, the consistency rules are vital for a valid BROS model. This section introduces validation and verification rules for the BROS adaptation method. The consistency checks are part of the adaptation process and done after the model specification phases within the *r.VER* and *r.VAL* instructions (see Section 5.6). In this thesis, the model's consistency is divided into two classes: verification rules and validation rules. While the former considers the syntactical correctness, the latter takes care of semantic consistency. For both, the adaptation of a reference model is in focus (especially the application of the BROS adaptation method, cf. Chapter 5). Nevertheless, the rules can be used partially[3] to check a BROS model that is constructed from scratch.

As well as the style guide, checking rules for validation and verification of the model are subject to change: future releases can introduce new rules and checks. Thus, the current set of verification and validation rules are a necessary foundation. Validation of a UML-based model can be done extensively and is a research field of its own (e.g., [29, 74, 261]). Additionally, the validation of a model is often dependent on the current use case, that is why it is challenging to state general purpose rules for validation. However, in this thesis, a basic set of rules is stated as an orientation for modeling with BROS and checking the result. The validation and verification of BROS are not in the focus of development yet, and multiple tasks have to be done (especially the formalization of the BROS language). Nevertheless, this section can be used to check the pursued syntax and semantics of the BROS model on a conceptual basis. The established rules are fewer but more substantial and significant than the guidelines, which is why they are positioned more prominently and kept generic.

6.2.1 Verification Rules

The verification rules are statements that make sure the constructed BROS model meets its syntax definitions. For every rule follows an explanation with some examples. The (yet) missing formal language of BROS does not allow rules as a formal statement, however, on a conceptual level, the rules can be used to check the BROS model after an adaptation (e.g., within the adaptation instruction *r.VER*). The rules do not affect the "content" and semantics of the BROS model but only the correct application of the individual model elements regarding their specifications.

Rule 1 *Every single model element meets its syntax definition.*

[3] Some rules are useful in combination with adaptation purposes only.

In Chapter 4, the individual model elements are explained, including a section for their syntax. In particular, this rule applies whenever an element is placed within a model: the correct shape, inner structure, and (possible) interrelationships have to be clear and precisely defined according to the BROS language specification. This includes, but is not limited to, the following checks:

- Shape and style of entities
- Line art of relationships and their arrowheads
- Placement of elements
- Connection of entities with relationships
- Necessary specification details of elements

Rule 2 *The domain (reference) model is structurally unmodified.*

The constraint of non-invasive and traceable adaptation steps (see Chapter 2) is the highest value of the BROS adaptation method as it utilizes the dynamic nature of roles, scenes, and events to define the adaptation in additive ways. The adaptation can only fulfill these properties when not interfering with the predefined, original reference model's structures. Therefore, the reference model as a template is a critical part when implementing the BROS adaptation method to retrieve the own enterprise's application model. The best adaptation can not succeed if the reference model is not suitable enough for the adaptation use case. A good choice of a reference model is needed (see instruction *p.RMS* from Section 5.2.1). Only if the reference model is unmodified during the adaptation process, the adaptation can benefit from being non-invasive. This counts for every part of the reference model (e.g., business objects, relationships, inner elements, multiplicities). Also, in case of an occurring reference model evolution (cf. Section 3.2.3), this paradigm is even more of an advantage as the non-modified reference model can be changed without tracing any modified parts of it.

Rule 3 *No other objects were used except those given by the reference model.*

This verification rule extends a step beyond the previous rule (do not modify the reference model). Not only must the reference model have structural integrity, but it also must not merely be extended by new fundamental concepts (that is, objects). In order to maintain the easy decoupling of the adaptation from the reference model, the adaptation has to be based on the concepts given by the reference model (again, also for reacting on reference model evolution). Therefore, the BROS adaptation method does not include any instruction that includes further objects into the model besides those that are derived from the reference model itself. A new object would indeed simplify the adaptation of the model to the requirements of the enterprise, but the new object is not aligned with the reference model. This discrepancy may cause the application model to diverge from the reference model, or further adaptations may rely too much on the new object, making it difficult to remove the new object from the model (or to align it with the reference model again). As a result, new objects are only allowed in BROS greenfield approaches where models are build from scratch.

Rule 4 *All roles have their fulfillments.*

This rule is rather basic and should make sure that every role has its fulfillment from an object (or another role). Independently of whether the roles are connected by arrows or using ports, static or dynamic, whether triggered events ever create them or not: every role needs at least one object that provides its identity via fulfillment (in the case of deep roles, the identity is passed by the fulfilling roles). Especially when using sub-scenes and nested contexts, the inner roles still have to be played by objects from the reference model.

Rule 5 *All dynamic elements are specified with events.*

A dynamic element of BROS, mainly the role and scene entities, is defined by its dependency on time. Time in BROS is, in general, represented by events. Thus, when a dynamic role or a scene is used, there have to be events that define the lifetime of these. For scenes, there are the init events to start the scene and return or exit events to end it. For usual dynamic roles, there are simply events stating a creation or destruction effect[4] towards the role. As a result, all dynamic elements in a BROS model are either in need of a start and end (modeled as events) or have to be modeled as static elements.

Rule 6 *Used elements are conceptually completely integrated.*

This rule is used to determine the necessary complexity and keep structural integrity. It is more general and abstract; however, it can have many consequences. The rule makes sure that every element used in the BROS model is checked for its correct and complete realization. It is a rather trivial task but can be critical if

[4] If a starting event is even more specific in its inner behavior, the event can be modeled as a scene on its own (e.g., the event `sign` to create the role `Customer` can also be modeled as a scene `Signing`).

failure leads to inconsistent models. In contrast to the first rule (respect the elements' definitions), this rule goes beyond that and takes care of the elements' environment. Not only the element must be specified precisely according to its definition, but also the conceptual embedding of elements into the model has to be rolled out: modeling subsequent elements and constraints that are in need to be modeled together with the former element in order to keep the construct and structure with integrity. This includes, but is not limited to, the following checks:

- Coordinated tokens for creation, assigning, and using identities
- Follow-up multiplicities (e.g., of roles) when nesting scenes and compartments
- Groups unify the inner roles' relationships and multiplicity
- Ports are used for accessing roles in contexts
- Constraints that are derived by other constraints

6.2.2 Validation Rules

Validation of a BROS model concerns the "content" and semantics of the model, which are not covered by the rules proposed by the verification. Nevertheless, since every adaptation use case is different from each other, there are no exact instructions on how to validate the exact model of every enterprise. Instead, the rules are more generic (like the verification rules) and strive for completeness with the goals and intentions defined in the adaptation instruction *p.AOD*. The validation instruction itself is part of the review phase (cf. Section 5.2.5) in instruction *r.VAL*.

Rule 7 *The domain (reference) model maintains its interpretation.*

The reference model is at the heart of the adaptation process. It provides the necessary structure to fulfill a particular purpose for a given domain. When adapting with the BROS method, the reference model is changed towards the enterprise-specific application model. The adaptation, therefore, extends the structure of the reference model (in an additive way and not the reference model itself) to retrieve the individually needed structure of the enterprise. While the reference model remains unchanged from the structural changes (also stated as verification rule), it has to be made sure that the original intention did not get lost during the adaptation. The meaning and semantics of a reference model are as important as the structure itself. The semantics are developed by domain experts and put in that way as it is. Thus, the modeler, as a user of the reference model, should adhere to the baseline intentions given by the reference model. In general, it can be challenging to extract the overall meaning of the whole model, but that is not in the focus of this validation rule. Rather, the single elements and their relation to each other should be included in the application model. Objects should, after adapting them with the BROS method, fulfill roles that do not entirely redefine the object and make "new objects." The same holds for relationships: connecting elements that are not intended to interact with each other is not recommended. Another example is the omitting of necessary fields ob objects. Many things may modify the reference model interpretation, and all of them should be avoided in order to keep the integrity of the reference model semantics.

Rule 8 *The goals, boundaries, and scope of the adaptation are adhered to.*

During a BROS adaptation method implementation, the defined adaptation statements (within the GPD) are a leading factor for developing the enterprise-specific application model. The statements include the necessary changes and expectations of the current adaptation. They are derived by a definition of goals and comes with a boundary for the model (see *p.AOD* instruction of the preparation phase in Section 5.2.1) to lead the implementation itself. Further, the intentions of the adaptation are used as checking the adaptation afterward for success (as a kind of acceptance test in instruction *r.VAL*, cf. Section 5.2.5). The goals, boundaries, adaptation statements, and other leading data are necessary to guide the adaptation in the right way and see whether it is successful. Thus, they must be adequately defined and, during the adaptation process, must be obeyed [5]. If the goals and adaptation statements are not met, the preparation phase should be considered again.

Rule 9 *Adaptations are as small as possible and as large as necessary.*

Each adaptation is an interference with the syntactic and semantic integrity of the reference model. Although an adaptation is intended to map a generic reference model to a specific application model, unnecessary adaptations should be avoided in order to reduce the error potential on the one hand and to maintain the given integrity as much as possible on the other hand. In contrast, the reference model needs to be adapted as detailed as possible in order to represent the very own enterprise structure and processes. A right balance is needed between these two ambitions. In general, a reference model has to be adapted and tailored as much as possible for the own enterprise's application model, but not more adaptation as definitely needed is done (with loose coupling). In case of doubt, it is better to leave an adaptation aside for the time being and use the reference model concept as it is intended. Especially partitioning and separation of concerns is a method to achieve a practicable (smaller size) adaptation.

[5] However, if the goals and boundaries are set in a wrong way beforehand, the adaptation cannot undo the initially defined misconception.

Rule 10 *Every entity has its necessary context.*

The concept of context is the main driver of the adaptation process. Context is used as a defining element for other elements, e.g., roles and events. Nevertheless, context is more than scenes and compartments. Context is the nature of an element's environment in which it is valid. The BROS language defines scenes and compartments as the main context elements, however, every element has an implicit context assigned (even elements outside of scenes and compartments). In theory, the whole application model has the context of the actual enterprise the application model is modeled for. This rule is, however, not based on syntax (like the verification of appropriate context modeling), but considers the semantic side of context. For each element (and that is definitely every element in the model) is has to be made sure that the element is placed in the correct semantic context. That means, in the case of a role, one should always ask oneself whether it actually makes sense in the current context or should rather be put into another context (for example, because it is more universally valid). Also, relationships need a context and can also be inter-contextual, which makes it more difficult to find the right context for relationships. As a result of this rule, every element used is defined by its most suitable semantic context.

Rule 11 *The events are coupled to existing and used background processes.*

Events are the representatives of time in BROS. Within an application model, the events are responsible for the predicate "behavior-aware." In an ideal adaptation use case, the events are not simple points in time but connected to a significant background process. This leads to a more transparent process adjustment in BROS models (e.g., if a background process changes) as well as a logical flow of context and roles. The set of related background processes is chosen in instruction *p.BPS* (cf. Section 5.2.1). An event is intended to point towards a position of such background processes to indicate the effect based on that process (e.g., an event that points towards a split node within a BPMN background process). In general, a BROS model can make use of any event that is needed to describe a role's or scene's lifetime. However, the events (in case of an actual adaptation use case) must not be random but based on any background process point. Alternatively, events defined in the BROS model have to be specified in the business process flow to indicate its triggering behavior. In particular, so-called *dead events* (never triggered events[6]) should be avoided. Finally, please note that the events in a BROS model do never state a process flow themselves but should be tightly connected to an existing background process trigger point.

[6] Similar to "dead marking" in petri nets [284].

Rule 12 *Semantic inconsistencies are resolved.*

The last rule about solving the inconsistencies is very vague and unspecific since inconsistencies are highly dependent on the use case and model implementation. Inconsistencies in models are often the primary cause of succeeding errors when using the models. Please note that this rule considers semantic inconsistencies (and not syntactically incorrect models). Finding, identifying, and solving the semantic inconsistencies of models is a challenging task and can hardly be automated[7]. Some smaller inconsistencies are easier to identify, e.g.,

- Missing init or return events for scenes
- Non-logical constraint constructs
- Confusion of static and dynamic roles
- Wrong fulfillment assignments
- Misplaced/missing relationships between two roles
- Misplaced/missing fields or operations of entities
- Never instantiated scenes

Other inconsistencies are more complex and require a deep analysis of the model's content to identify occurring inconsistencies. Often, the analysis requires a view on design time and runtime simultaneously. Especially when dealing with temporal effects or logical constraints, inconsistencies are often a pitfall[8]. Some examples of complex inconsistencies are the following:

- Events on different timelines affecting the same entities
- Object identities that are passed from one context to another
- Constraints that are dependent on the runtime object identity
- Unwanted but "inherited" relationships from the reference model[9]
- Infinity loops when using events

6.3 Design Patterns

Patterns for software engineering and design are one of the most commonly used tools to construct useful applications (within small dimensions). Patterns are templates that provide a knowledge-based solution to common problems [92, 253]. Often, they solve the problem for a certain assumption (e.g., to be more effective, adaptive, or usable)

[7] Models that are based on a formal language are in advantage in such a case; however, BROS is not.

[8] Which is why a reference model's BROS adaptation should be kept simple but precise.

[9] The reference model's relationships between objects are still present, BROS only modifies existing relationships or create new ones.

on a particular basis layer. The problems are often design issues ("How to represent something?"), the basis layer can be of any technology or abstraction (hardware composition up to business modeling and requirements). The pattern provides a solution on that basis layer. It is important to note that the solution is not unique, but just a single possible one. Nevertheless, the pattern's solution is often driven by practical experiences and commonly accepted. An often referenced (general) pattern definition is from Alexander [5]:

> "Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice." [5, p. 10 (X)]

Further, Beck et al. [21] share and extend this definition's view with their definition of a design pattern:

> "A design pattern is a particular form of recording design information such that designs which have worked well in particular situations can be applied again in similar situations in the future by others." [21, p. 103]

There is a common misunderstanding of design patterns to be restricted to object-oriented patterns [253]. On the contrary, patterns are available for in multiple forms and within the full range between technical and abstract specification, e.g., patterns for programming languages [130, 215], microservices [208], enterprise architectures [82], business processes [7], or workflows [259].

Regarding BROS, the basis layer is between the conceptual business logic and the technical software design (like UML). Although BROS patterns are not derived from commonly used solutions of frequent problems (the BROS language is a rather young modeling language, yet), the BROS patterns are templates to several issues that might arise when starting with modeling. In this thesis, six patterns are stated, firstly to introduce solutions to common modeling issues, and secondly as examples of how to solve follow-up issued with the BROS methodology thinking. Each pattern states its title, problem explanation, and (possible) solution.

Shared Return

Destroy a scene when another scene is finished by a return event.

To destroy a scene in correlation with the destruction of another scene is a common use case in parallel sequences of behavior flows. There is no BROS language element like a "shared return event." Thus, this *shared return* pattern introduces a structural construction of events and scenes that act like such an element.

Figure 6.4 shows the concept of a shared return pattern modeled in BROS. It is, nonetheless, rather trivial: if the scene `X` returns with event `a`, the scene `Y` is exited by event `a` as well. Both events use the same identifier, thus, they are handled as the same conceptual event, and a trigger would affect both of them. Alternatively, an event cause can be used between the scene `X` and the exit event to indicate their relationship.

Fig. 6.4 Shared return pattern

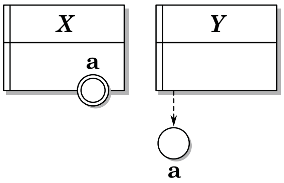
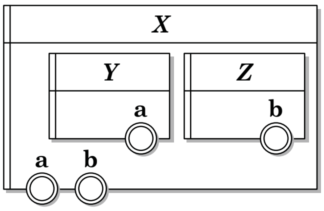


Fig. 6.5 Terminator pattern



Terminator

Propagate the destruction of one nested sub-scene to the surrounding scene and all other nested sub-scenes.

In BPMN, there is a concept called termination event that causes the whole process and all related and nested participants to exit [184]. In BROS, every scene has its own return and exit events. If a nested sub-scene ends itself via a return event, the surrounding scene is still valid and active. Thus, this simple *terminator* pattern provides a solution to end a surrounding scene and all neighbor scenes whenever any sub-scene is finished.

The pattern is shown in Figure 6.5. Assumed that the events `a` and `b` are based on BPMN termination (or similar) events, the same events with the same identifier are added to the surrounding scene `X` to indicate that, whenever such an event is triggered, all sub-scenes end due to the return of the surrounding scene `X`.

Overheat

After the *n*-th occurrence of an event, the role of an object switches to another.

There are use cases, where, after a certain number of repeated event occurrences, a role switches to another role, e.g., a `Letter` plays the `Reminder` role until the third occurrence of `letter send` event, then the object switches to the `Warning` role. For that purpose, the *overheat* pattern is introduced, switching to another role dependent on the number of occurrences of an event.

The pattern is illustrated in Figure 6.6. The object `O` plays the role `R` as soon as the event `a` gets triggered. The role `R` remains until either event `b` occurs for the *n*-th time (and switches to role `S`) or event `c` cancels the role `R`. The multiplicity `n..*` is used to indicate that the event triggers from the *n*-th occurrence on.

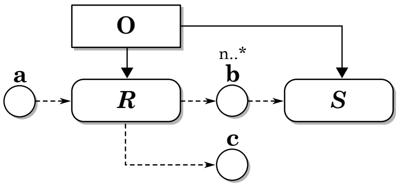


Fig. 6.6 Overheat pattern

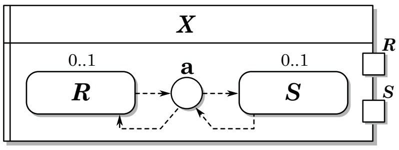


Fig. 6.7 Event switch pattern

Event Switch

Using a single event to switch repeatedly between two roles.

Switching between roles can be tricky when there is only a single event that is responsible for creating and destroying the roles. In such a case, there must be help from a scene and role multiplicities which role must be destroyed and which role is to be created. Counting the occurrences is not possible (like in the overheat pattern) since it can happen irregularly. Further, there cannot be two events, as both events would be connected to different points in the background process flow. The *event switch* pattern tries to solve this issue.

Figure 6.7 shows the BROS model construction of the pattern. A scene `X` contains the switching roles `R` and `S` (including their ports). An event `a` should act as the switch event; it has create and destroy relationships towards both roles. The roles have a multiplicity of `0..1` as an indicator that there is at most one active role instance at a time. The event's effect loop switches the roles as long as the event `a` gets triggered. There can be multiple active scene instances of `X` in parallel, nevertheless.

Moving Identity

Using a unique identity in a flow of scenes.

Especially when modeling the enterprise's processing chains (different stations for the same product), one is in need of stating the exact identity that moves around the different processing units. The *moving identity* patterns make use of tokens to model state the identity that participates as a role in different scenes. There is hardly a possibility to model this use case without tokens (sometimes, a role implication constraint may help) as identity management is rare in the core BROS concept set.

In Figure 6.8, the pattern is stated. The object `O` fulfills the role `R` while getting an identifier token ≪1≫ assigned. When the event `b` occurs, the scene `X` (representing

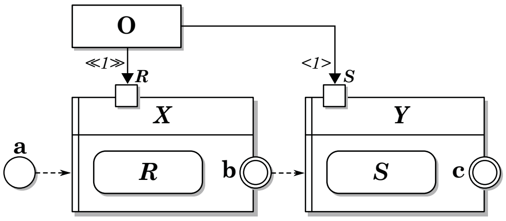


Fig. 6.8 Moving identity pattern

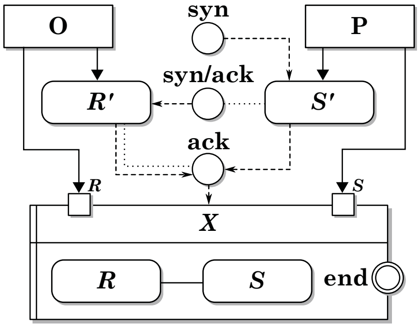


Fig. 6.9 Three-way handshake pattern

the first processing unit) ends, and its return creates the next scene (the next processing unit). To ensure that the same identity is used for the next scene as well, a token <1> indicates that it is actually the same identity used. The fact that both scenes are (time-wise) sequentially processed, a role implication cannot be used (the former roles are destroyed when the next are created). Please note that the token label does not count as a counter or multiplicity: it simply states that the same identity (type) must be used.

Three-way Handshake

Create two partner roles with a three-way confirmation check.

The *three-way handshake* pattern is tightly connected to the related protocol type, called a three-way handshake exchange, mainly used in computer network technology (i.e., TCP/IP) to authenticate two parties with each other[10]. The common names for the exchanged messages to establish such a network connection are `SYN`, `SYN+ACK`, and `ACK`. Usually, BROS models do not represent complete business behavior flows; technical processes like this might be useful nonetheless.

The pattern is shown in Figure 6.9. After the `syn` event, the intermediate role `S'` is created fur preparation purposes. After the triggering of event `syn/ack`, the second intermediate role `R'` is created as well. Finally, the last exchange event `ack` ends the process of exchange, and the objects can interact with each other within the created scene `X` until event `end` gets triggered.

[10] `https://developer.mozilla.org/en-US/docs/Glossary/TCP_handshake`

Chapter 7
Case Study

"Software users and developers are demanding systems whose complexities often exceed our abilities to construct them." [231, p. 19]

Abstract The BROS modeling language can be used for structural modeling, specialized for OO-based reference model adaptation via roles. The analysis and specification of a language is, however, only a theoretical statement. To show the applicability and usability of a language, a practical statement is needed. This issue holds for BROS as well. The practical demonstration can be done in several ways, e.g., a use case example or qualitative study. Since the BROS language is rather new, a use case is required to complete the specification of the new language. The real-world use case is used to demonstrate two issues: (a) that the BROS language is applicable in practice, working on standard OO-based reference models in combination with conventional BPMN models as background processes, and (b) to illustrate the benefits, advantages, and drawbacks of the BROS language by comparing it to standard UML class diagram adaptation. In this chapter, the BROS language is used for dynamic and structured adaptation of an existing structural reference model from the cloud service provider domain concerning a new feature, a so-called "partner program" process. Further, by comparing the BROS adapted model with a traditional UML-based adaptation, the benefits of the new BROS methodology are shown (e.g., extended expressiveness and flexibility towards changing requirements and features).

7.1 The Cloud Service Provider Use Case

For this use case, a German small-sized cloud service provider (CSP) enterprise was consulted to apply BROS on its reference model, according to available background processes. The CSP was involved in the whole adaptation process and guided the adaptation with requirements and suggestions.

> "A cloud service provider is a third-party company offering a cloud-based platform, infrastructure, application, or storage services. [. . .] companies typically have to pay only for the amount of cloud services they use, as business demands require."
> [`https://azure.microsoft.com/en-us/overview/what-is-a-cloud-provider/`, acc. 03.2020]

A cloud service provider is a sub-part of the cloud computing business: it provides scalable and individual solutions concerning the consumed computing resources in terms of minimal entry costs, pay-as-you-go cost model, and flexibility [222].

> "Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction." [173, p. 2]

According to the *National Institute of Standards and Technology* (NIST) [173], a cloud computing service has five essential characteristics: on-demand self-service (automated provisioning of computing capabilities), broad network access (access over the network), resource pooling (the CSP's resources are optimized for several customers), rapid elasticity (flexible resource allocation), and measured service (automatic control and measurement). Further, cloud computing service offerings can be classified within two dimensions [173]:

1. *Service Model*, including Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS), and Infrastructure-as-a-Service (IaaS)[1]
2. *Deployment Model*, using the classes private cloud, community cloud, public cloud, and hybrid cloud

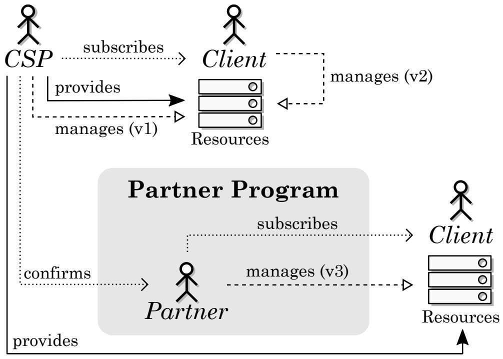
The CSP, as a cloud computing service, in this use case analysis, is a small-size enterprise (with around ten employees) acting in the German market. It provides isolated cloud resources to individual customers, thus, it operates in the private cloud class as a deployment model. The CSP operates for multiple clients and offers a diversity of (cloud computing) products: managed servers, websites, online storage, and other related products. It can mainly be classified as an IaaS-Service, although PaaS and SaaS products are available as well. The customers are both private clients and enterprise clients, and the number (as individual tenants) is in the range of 100 to 500 clients. As a business model, the CSP concentrates on selling resources in small sizes for clients that do not require large infrastructures or excessive amounts of bandwidth. In particular, the management of the provided resources remains at the client's side (although the client could decide to make use of the additionally provided management services as well).

The use case is based on how a client's resources are managed and by whom. Usually, there is the decision of the clients to manage related resources on their own or to pay the CSP for the related extra services. To provide more convenience for the client and to be flexible for the client's demands, the CSP wants to offer a new management method: the *partner program*. This program should act as a new way of how resources are managed:

1. "It allows customers to resell purchased cloud services and resources to their own customers" [222], allowing clients to act as intermediary retailer themselves.
2. A client can define or accept an (other) external client to manage the resources.

[1] In reality, there are a lot more online services that claim to be a service class on their own, e.g., Databases (DBaaS), Business (BaaS), Container (CaaS). However, they are not recognized as a foundational class but rather a service flavor.

Fig. 7.1 The CSP use case of reselling resources as a partner client to other clients



Cloud computing resource reselling is an essential feature but, nevertheless, a significant step towards profitable B2B markets [200]. Basically, a new feature is required to be implemented that allows clients of the CSP to register as a *partner*, then be able to manage other customer's accounts after a presumed invitation agreement (see Figure 7.1). According to Böhm et al. [35], the involved market actors in this partner program are specified as follows:

> "*Partner* as a synonym for value added resellers who either aggregate modular services to create value-adding, complex solutions for specific requirements (*Aggregator*) or integrate cloud services into the existing IT landscape (*Integrator*) and *Customer* as consumers who receive these services." [222, p. 5]

Use Case Description Summary

A cloud service provider (CSP) maintains an in-house OO-based reference model, stating the structure of the cloud infrastructure, resource management, and client handling. Clients of the CSP can order cloud resources that are managed either by themselves or by the CSP. A new feature (the "partner program") is going to be implemented that allows clients of the CSP to manage other client's accounts and resources as a third option. A managing client ("partner") then acts as a reseller for the cloud resources (see Figure 7.1).

For development purposes, the CSP has built up an in-house reference model, consisting of the domain's business objects (including major technical entities) in an object-oriented manner. It is directly combined with the CSP's database schema and, therefore, all changes and modifications regarding a new feature have to be in compliance with this self-built reference model (the complete description of the reference model is given in Section 7.2.1). In the interest of this thesis, the use case is implemented using the BROS adaptation method (cf. Chapter 5). Therefore, the reference model is used as the foundational template (cf. Section 4.2.1). Together with the CSP's business processes (mainly the background process described in Section 7.2.2), BROS was used to adapt the reference model to implement the new feature. The developed role-based adaptation realizes different roles and contexts to achieve the reselling feature in a non-invasive and transparent way. More focus was paid to the use of multiple BROS language features rather than the minimal implementation of the adaptation for the use case.

The complete use case was analyzed, developed, and evaluated together with the CSP enterprise's stakeholders, including the specification of boundaries and goals of the use case's adaptation. Although the use case and the related BROS solution are already published (partly) in Schön et al. [222], a more in-depth view of this use case is given in accordance to evaluate the BROS language and adaptation method in this thesis.

7.2 Adaptation Environment

As written in Section 7.1, the adaptation was made using the given (in-house) reference model of the CSP and related background processes. The domain model consists of the main (business) objects that are going to be adapted (see Section 7.2.1), and the background processes provide information about behavioral nature (see Section 7.2.2). The adaptation of the reference model results in a BROS model that is (in contrast to modeling from scratch) a concrete adaptation of given entities and relationships by the reference model.

7.2.1 Domain Reference Model

The CSP does not use an external, standardized reference model (possibly provided by a standardization organization) but uses its own internal and in-house reference model. According to the CSP, this ensures and enforces

> "[...] every employee involved in development to adhere to this domain model in order to prevent non-transparent interfaces and to ensure fast and inexpensive software system evolution." [222, p. 5]

The reference model consists of around 150 entities (business objects as well as technical entities) and is used as a source for the internal software structure. Many subsequent technologies depend on that reference model (e.g., database structures, interface design, web views). It is a rather small reference model, compared to other commonly available reference models for industry (e.g., the BIAN banking domain reference model[2]). Nevertheless, it is, according to the small company size of the CSP, of advantage

> "[...] as it provides a sufficiently sized but still comprehensible overview of the whole system" [222, p. 5].

[2] https://www.bian.org

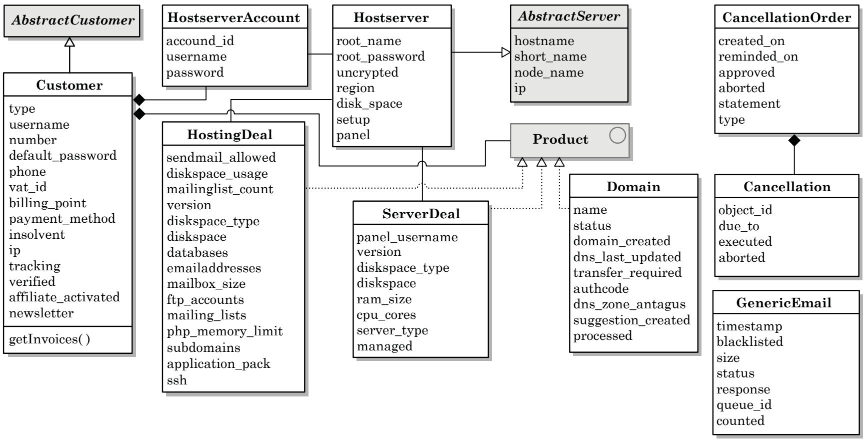


Fig. 7.2 The used part of the CSP's detailed reference model for the use case (from [222])

Developments (maintenance as well as new features) and evolution of the system is, due to the enterprise's development principles, tightly oriented and connected with the reference model. New features and maintenance fixes have to be made according to the constraints. The reference model itself is subject to evolutionary changes: whenever a new feature is required, the reference model is adapted (in the common UML invasive way) to meet the new requirements. Thus, fast and favorable evolution and modification of the enterprise's software system can be achieved. This, in turn, is beneficial to strengthen and expand the enterprise's position on the cloud computing market [90, 181] due to low costs and short time-to-market.

Due to its in-house nature, the owner of this reference model is part of the enterprise as well. Thus, the reference model can be modified and "released" in its new version rather quickly and in an uncomplicated way (the feedback loop is very short and uncomplicated as well). This fact concludes that an adaptation (e.g., with BROS) is not that necessary for new features since the small enterprise structures allow an easy reference model evolution (which may be more efficient than an adaptation). Nevertheless, the use case is, due to its small size, an excellent demonstrator for the BROS language and adaptation method. In addition, an adaptation can still be advantageous for smaller reference models (in contrast to reference model evolution).

Figure 7.2 states the part of the reference model that is used for this use case[3]. The use case's boundary (that is, the partner program) is rather small, which is why the number of required reference model entities can be limited to the shown objects in Figure 7.2. Operations are omitted to maintain readability and clarity within the model. Further, only the direct relationships between entities are shown, thus, some entities are not (directly) connected to the others (e.g., `GenericEmail`). All shown entities are then used as objects in the BROS model.

[3] To show all 150 entities would not fit here.

The properties, meanings, and relationships of the single entities in the reference model in Figure 7.2 are as follows [222]:

- The `Customer` entity functions as the main business object. It represents the client with all the information needed. Several technical objects are not shown, e.g., outsourced attribute data objects. The `Customer` inherits from *`AbstractCustomer`*, an interface intended for creating new customer types in the future[4]. The `Customer` aggregates an account (`HostserverAccount`) and may use several `Products` as cloud resources.
- The `HostserverAccount` object enables the access of a `Customer` for a `Hostserver`, which hosts either a hosted product `HostingDeal` (e.g., a web page) or a concretely provided server resource `ServerDeal` (a virtual machine).
- "`HostingDeal` [objects] are hosting plans with shared resources on a server (e.g., storage, databases, email addresses)" [222, p. 5] They use the `Product` interface and can, therefore, be bought by a `Customer`. The `HostingDeal` can be accessed via an obtained `HostserverAccount`.
- In contrast to hosted products, the `ServerDeal` objects are used for rented virtual servers with a specified set of allocated resources regarding, e.g., CPU, memory, disc space, bandwidth, SQL batch query size limit. Since they use the `Product`, they can be bought by a `Customer` as well.
- The `Domain` is a `Product` as well, but it does not require an `HostserverAccount` to "log in" into it (like the two other server-based products). However, it can still be bought by a `Customer` as an internet identification service for other cloud resources.
- The `Product` object is realized as an interface and specifies the three main resources (`HostingDeal`, `ServerDeal`, `Domain`) as obtainable products by a `Customer`.
- The `Hostserver` represents the "real" server that is running `HostingDeal` and `ServerDeal` instances. It inherits from *`AbstractServer`* as a specification interface. A `HostserverAccount` can be used by a `Customer` to gain access to the hosted products.
- The customer can cancel licenses by the creation of a `CancellationOrder` object. It contains single `Cancellation` objects that refer to the products and contracts that are going to be canceled.
- The `Cancellation` is an object mainly used within `CancellationOrder` and represents a single position in the cancellation list.
- At last, the `GenericEmail` object is, as the name implies, an object used for all different kinds of cases that involve an email object. However, within the represented part of the reference model, it is not connected to other elements directly.

All the named objects take part in the adaptation. Usually, the subset of needed objects from the reference model is part of the *s.ROL* instruction (to find suitable fulfillments). Due to readability, the needed objects are already pre-selected in this

[4] Usually, the reference model would be changed by inheritance when adding new customer types. When using BROS, the interface would not be necessary as the customer just plays different roles.

thesis (stated in Figure 7.2). It would not be a hard task for the reference model owner (a group of employees of the CSP) to change the reference model according to the new feature (in fact, this was already underway). Nevertheless, the reference model proposes a reasonable basis for an adaptation, and the use case already implies the natural understanding of a role (clients as partners).

7.2.2 Background Process

The adaptation of the CSP's reference model is based on the provided use case description of the new feature. Although BROS can be used and applied without any behavioral information that leads the adaptation, a set of background processes is recommended (see Section 4.2.2; usually done in instruction *p.BPS*).

During the collaboration with the CSP, several business processes were identified that are related to the new partner program feature but do not meet the complete requirements. There was no suitable process that can be used to derive scenes and events for the adaptation. Thus, it was decided to implement a new business process that should map the new functionality in its procedure. The goal was a process description that is not too detailed and covers all exceptional cases but represents the basic procedure and branches sufficiently. A workshop with all related stakeholders was initiated to identify the needed process description[5]. As a result, a top-layer BPMN model was developed, which covers the functionality of the partner program

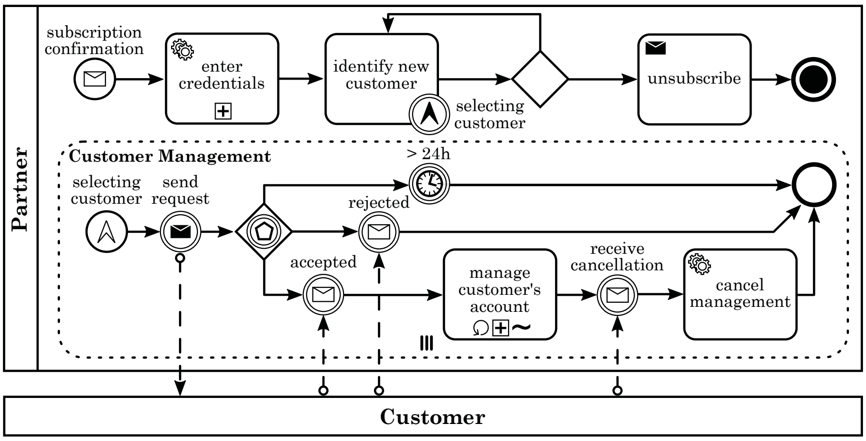


Fig. 7.3 The used CSP background process for the use case (from [222])

[5] The identification, analysis, creation, and management of business processes is an extensive research field (e.g., [2, 57, 213]) and can not be covered by this thesis. It is recommended to collaborate with experts for enterprise business processes to construct relevant and critical business processes.

(see Figure 7.3). It is, due to its new development, rather abstract and considers the process on a more business level (that is, omitting some technical details as well as a validation and verification) for understanding. This process is going to be extended and refined by the CSP in the future, however, the impacts of process evolution on the BROS model are low (due to the generic nature of events they can often be reused if the background process evolves).

The generated BPMN process acts as the main background process for the adaptation. The process was designed based on the expectations, ideas, and suggestions of the CSP's stakeholders [222]. Thus, it represents the feature in its intended way. The notation via BPMN helps to model the feature in its details accordingly. As a background process, it is mainly used to derive BROS events and scenes that define the temporal properties of the adaptation's structure (i.e., roles).

The background process for the feature, shown in Figure 7.3, is structured and specified with the following milestones:

1. A customer becomes a `Partner` with the event `subscription confirmation`.
2. The `Partner` has to enter its own information via `enter credentials`.[6]
3. The `Partner` identifies other customers (`identify new customer`) and selects one of them. This activity is followed by a decision split that returns to the same activity again. The BPMN event `selecting customer` leads to a parallel sub-process `Customer Management`, which can be executed in parallel for multiple selected customers.

    a. The sub-process `Customer Management` starts with the event `selecting customer`, triggered by the `identify new customer` activity.
    b. The `Partner` sends a request (`send request`) to the related `Customer`, who has the chance to answer the request.
    c. If the `Customer` answers the request with `rejected` or does not answer within 24 hours (>`24h`), the sub-process `Customer Management` ends immediately.
    d. If the `Customer` answers with `accepted`:
        i. An `manage customer's account` activity succeeds with the following properties: it is repeatable, contains more activities, and refers to a creative process. It represents the abstract activity of handling the chosen costumer's booked resources (that is, the *manages (v3)* arrow in Figure 7.1).
        ii. When the `Customer` sends a cancellation and the `Partner` gets `receive cancellation` event, the activity `cancel management` is executed.
        iii. Finally, the sub-process `Customer Management` ends.

4. If the `Partner` uses the activity `unsubscribe`, the whole process ends together with all its executed sub-processes.

The background process is indeed not the most detailed and impeccable process. Nevertheless, it is used to identify the temporal constraints of the BROS model, thus, it does neither need to be technical or thoroughly refined nor validated according to the BPMN standard. For the use case, the process has the right granularity and

[6] This activity is, for example, not going to be modeled as an event since it does not modify the lifetime of any other element.

functionality. The advantage of BROS is the flexibility of events, such that they do not require the most specified and defined background processes but an indication coupling to distinct points in the process flow.

Besides the feature background process, there are other processes in the CPS's repertoire. However, they are not that important for the use case as they are not directly related. This is not the standard case, as usually, BROS adaptations can surely make use of multiple processes as background processes (like the case rental example in Chapter 6).

7.3 Adaptation Execution

The adaptation is made in two different ways:

- using the BROS adaptation method, and
- adapting the reference model with standard UML.

This serves two reasons: firstly, one can retrace and comprehend the application of the BROS modeling language as well as the adaptation method, and secondly, the modeling results can be compared with each other in order to conclude the benefits and drawbacks using the BROS method (see Section 7.4). In this section, the BROS adaptation is used first (cf. Section 7.3.1), and the UML adaptation is made afterward (cf. Section 7.3.2). The use case is the same for both approaches. The partner program is to be modeled based on the reference model and the (new) BPMN business process. The leading property factors are "non-invasiveness" and "accurate implementation" of the use case. The BROS approach follows the BROS adaptation method (cf. Chapter 5), while the UML is an ad-hoc adaptation that tries to implement broadly similar decisions and sequences as BROS does.

In both cases, the CSP and all related stakeholders were involved in the adaptation process, taking their decisions, recommendations, and expectations into account.

7.3.1 BROS-based Adaptation

The BROS adaptation of the use case was made via the BROS adaptation method from Chapter 5.

> "The application of BROS for adaptation benefits from the given background process as the modeler's main source of business knowledge for conducting the adaptation." [222, p. 6]

The adaptation of the use case (in this thesis) utilized the five phases of the BROS adaptation method and its instructions[7]. The entire adaptation process is mapped as precisely as possible to the individual steps, although this does not always apply

[7] The original use case in Schön et al. [222] states that the step-by-step adaptation guide is not available. Nevertheless, it was not published but still present (cf. Chapter 5).

in individual cases. The BROS section is, therefore, divided into the related phases, and the model is built phase-wise. The specification phases of the BROS adaptation method are done iteratively but are stated and described as single paragraphs.

> Usually, the implementation of each instruction is not made block-wise (like listed below) but done in different order and composition. Due to the iterative phase design, the phases of structure, behavior, and context specification are mixed during the adaptation process, and instructions were executed multiple times. The listing below is a summary of the instruction types and phases, which combines the decisions in order to increase readability and transparency.

7.3.1.1 Preparation Phase (cf. Section 5.2.1)

Adaptation Objective Definition (p.AOD) The first step encounters the feature on its conceptual level. As stated in Section 5.2.1, the *p.AOD* instruction defines the goals and boundaries of the adaptation by setting the organizational context, goals and sub-goals, adaptation statements, and the boundary. The GPD is generated in this instruction as well. The use case, its goals, and limitations are already described in Section 7.1. Thus, the related information was collected and stated accordingly (e.g., as adaptation statement “Partner can send requests to other clients.”). The GPD was not existing, thus, newly created to track the adaptation metadata in an XML file.

Reference Model Selection (p.RMS) The reference model is described in Section 7.2.1. Because the CSP has only one reference model available, the choice is limited. The requirements for the reference model were met, particularly the granularity and domain completeness. The model was partially updated and improved because some concepts were not sufficiently maintained. The meta data were added to the GPD.

Background Process Selection (p.BPS) Like the *p.RMS* instruction, the selection of related background processes is limited due to the use case. The new feature has a BPMN process model (cf. Section 7.2.2) that is going to be used as the primary background process. Although it is not verified with the BPMN standard, it is syntactically and semantically useful as a source of temporal information of BROS language elements. No other processes of the CSP were needed for the use case. The complete process and its metadata were added in the GDP file with the priority *M*.

7.3.1.2 Context Specification Phase (cf. Section 5.2.2)

Package (c.PAC) The adaptation is a new one and, therefore, gets a new package (as stated in the context phase’s instruction in Section 5.2.2). The package is named after the new feature (`Partner Program`) and is, for now, semantically meaningless but

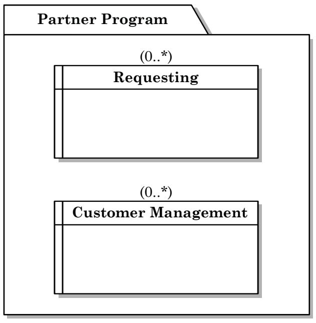


Fig. 7.4 The CSP's application model after the context specification of the BROS adaptation

encapsulates the adaptation as an application model. It contains all other adaptation elements (see Figure 7.4), including the contexts.

Dynamic Context (c.DYC) In the BROS language definition, a scene is defined as a "temporal collaboration context of roles and events, related to the same business logic" (cf. Section 4.4.3). Regarding the use case, two dynamic contexts were identified:

- `Requesting`, and
- `Customer Management`,

the former as a process-like scene and the latter as a state-like scene. Both are derived from the background process. The scenes are based on activities and separable timelines during the process and fulfill the scene checks (has an identity, is temporal, is not atomic, is unique). The scenes are placed in the model side by side (see Figure 7.4). They contain the roles and events specific to the background process: the `Requesting` deals with the invitation of another customer by the partner, and `Customer Management` is a scene for managing the related customer's cloud resources until the cancellation occurs.

Static Context (c.STC) There are no compartments needed for the use case. A possible compartment could be `Product` that combines the three cloud resource products (under the condition that the boundary would be defined accordingly). Nevertheless, the products were not modeled that way, and thus, no compartment was introduced.

7.3.1.3 Structure Specification Phase (cf. Section 5.2.3)

Role (s.ROL) The roles state the representatives and participants of the objects. The CSP uses the progressive adaptation variant, that is, using the given reference (business) objects to define the required roles, and roles are used as representatives. After a discussion with all stakeholders, the use case implies the following roles (as stated in Figure 7.5), which are constructed via the *s.ROL* instruction:

- The `Partner` role is the most prominent role as it denotes the partner within the partner program process. Clearly, a customer can subscribe to be a partner, thus, gets the `Partner` role. Some fields are added (without indexes as they are only additive to the object's fields). The partner role is fulfilled by the `Customer` object.

- The customer is used in two kinds of customers: `Person` and `Company` customers. They are going to be static roles as both roles "only" type the customer in a category and are not dependent on any temporal constraint. Both are fulfilled by the `Customer` object as well. In case a distinction would not have been necessary, a single role Customer would have been sufficient. Contrary to the `Person` customer, the `Company` customer uses other fields (e.g., `vat_id`).
- The `Cancellation` role is fulfilled by the `Cancellation` object of the reference model and participates in the process as well.
- A role `Pending Customer` is placed inside of the `Requesting` scene, fulfilled by a customer (`Person` or `Company`) role (as a deep role). It denotes a customer that is asked about resource management. The role has two operations that are necessary to implement. Since the role is needed every time the scene starts, the `Pending Customer` role is an init role of the `Requesting` scene with a multiplicity `1..1`.
- An `Invitation` role is generated inside the `Requesting` scene as well. It represents the `GenericEmail` object within this process flow. The role carries a field for special information regarding the contract. It is, like the `Pending Customer`, an init role for the scene.
- The `Managed Customer` role is placed inside the `Customer Management` scene as an init role and represents the customer (`Person` or `Company`) role (thus, fulfilled as a deep role).
- The `Manager` role is a representative of the `Partner` role (as a deep role inside of the `Customer Management` scene) and acts as the managing unit for the customer's resources. It has some admin-related fields and is an init role for the scene.
- The product can be divided into two kinds: `Domain Product` that is fulfilled by the `Domain` object, and `Hosting Product` that is fulfilled by the `HostingDeal` and `ServerDeal` objects. Both roles are part of the `Customer Management` scene.
- The role `Admin Account` represents the `HostserverAccount` object inside the `Customer Management` scene, responsible for the customer's products.

The result of all *s.ROL* instructions are illustrated in Figure 7.5, where the roles are implemented in the application model. Each role comes with a fulfillment (and needed ports) and is placed according to its context.

Association (s.ASC) The associations introduced into the model are manifold. The roles generated by the *s.ROL* instructions have to be connected in order to communicate with each other. The *s.ASC* instruction was implemented several times (usually an association follows a role implementation to connect it). As a result, roles are connected in a way how they work together (e.g., `admin` between the roles `Manager` and `Admin Account`). Associations are, by their nature, very diverse in their function. Role associations are no exception so that they take on the most different functions in the model. A further specification of the association functions would be possible, but this was not included in the use case.

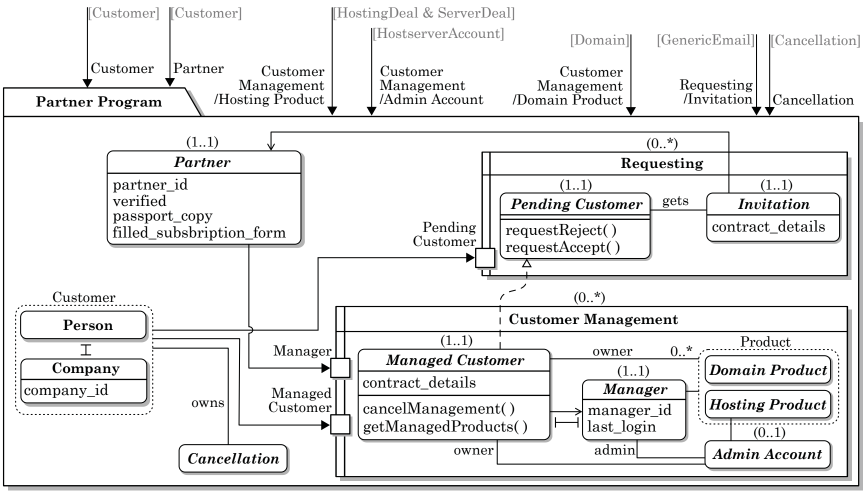


Fig. 7.5 The CSP's application model after the structure specification of the BROS adaptation

Constraint (s.CON) Constraints are used to define a certain logic between role fulfillments at runtime. In this use case, the *s.CON* instruction was used three times:

- a prohibition between the roles `Person` and `Company`, since both can not be played by the same object instance simultaneously,
- another prohibition between the roles `Managed Customer` and `Manager` as a manager can not act as a reseller simultaneously, and
- an implication from the role `Managed Customer` towards the role `Pending Customer` to ensure that the managed customer was invited before the reseller may manage the related resources.

Group (s.GRP) Groups help with organization and simplification of multiple roles (cf. Section 4.5.4). The model at hand (see Figure 7.5) uses two groups, implemented by the *s.GRP* instruction: `Customer` and `Product`. Both encapsulate similar roles that act like types and can be considered to be exclusive with each other. They can be handled as a single combined entity regarding the associated relationships (e.g., the whole `Customer` group is associated with `Cancellation`). However, single role references are still possible (e.g., the association between `Admin Account` is only towards `Hosting Product` instead of the whole group). As a result, the two groups are implemented to simplify the types of customers and products.

Port (s.POR) The port concept (cf. Section 4.5.3) is not further used in the use case's application model. The three available ports are already introduced by the *s.ROL* instruction (to access roles within a context).

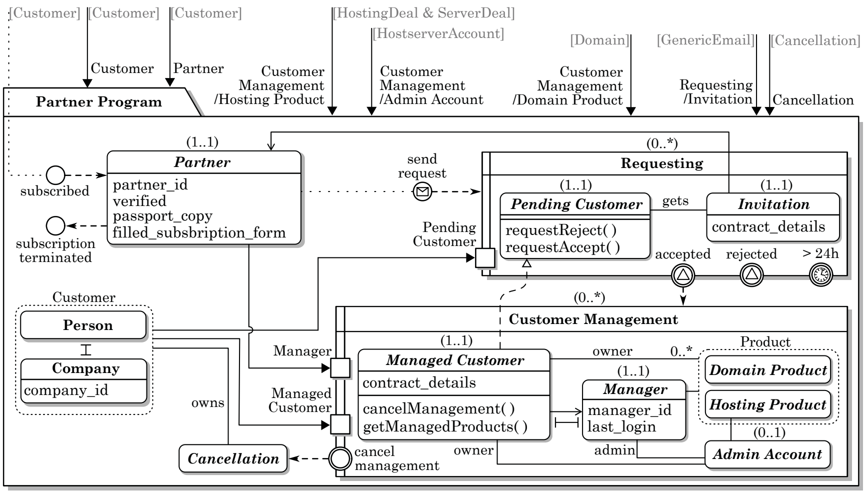


Fig. 7.6 The CSP's application model after the behavior specification of the BROS adaptation

7.3.1.4 Behavior Specification Phase (cf. Section 5.2.4)

Event (b.EVE) The event instruction *b.EVE* is the primary instruction of the behavior phase. The events are identified and specified according to the background process (cf. Section 7.2.2). As stated in the event specification (see Section 4.4.4), events are used to determine specific points in time that affect other BROS concepts. Since the application model consists of two scenes (`Requesting` and `Customer Management`), the init and return (or exit) events are to be defined. Further, other events for single roles may occur during the process flow execution and need to be considered as well. As a result of the *b.EVE* instructions, the application model (see Figure 7.6) consists of the following events:

- The two events `subscribed` and `subscription terminated` are used to create and destroy the `Partner` role, making it a dynamic role (please note the italic letters). Both events are connected to the background process: the former with the BPMN event `subscription confirmation`, the latter with the activity `unsubscribe`. The state of "being subscribed" could be modeled as a whole scene alternatively, with both events as start and endpoint in time[8].
- According to the background process, the `Partner` sends an invitation to a selected customer. This is represented via the BROS event `send request`, which has the `Partner` role as its cause and creates the `Requesting` scene as init event.
- The BPMN flow of the customer to decide between the single options are not modeled as they do not have any impact on the roles. The scene `Requesting` (started by the event `send request`) instantiates the roles `Pending Customer` and

[8] However, the stakeholders decided to use a dynamic role instead of a scene, as this is more concise and sufficient for the use case.

Invitation. The outcomes are modeled as the scene's return events: accepted, rejected, and >24h. Only the first one has a creation effect; the other two return the scene without further effects. The return events are tightly connected to the choice of the customer on how to react to the invitation. The >24h event is considered to be a timed event (still dependent on the process), the others are regarded as signal style since they reflect a Boolean answer.

- The return event accepted returns the scene Requesting and acts as init event for the next scene Customer Management. Within this scene, the Partner manages the Customer's resources as a Manager. After considering the feedback from the CSP, no events are introduced into the Customer Management scene.
- Every scene is in need of a start and endpoint in time. Thus, the Customer Management scene ends with the cancel management event, bound to the background process BPMN activity cancel management. The event returns the scene and creates a role for a cancellation item (which, in turn, could be used for, e.g., accounting purposes). With this, the process flow ends.

Token (b.TOK) The token concept (cf. Section 4.5.5) is, for this use case, not relevant. There are no tokens needed to state the identity of certain roles' players. The implication constraint could be implemented as a token construct alternatively. However, constraints should be used before using tokens, if possible (see style guidelines in Section 6.1.2). Another possibility is to use "create tokens" to mark the Invitation and Cancellation fulfillments to create new object identities when fulfilling the roles (instead of using an existing object identity).

7.3.1.5 Review Phase (cf. Section 5.2.5)

Verification Check (r.VER) The verification of the model was rather complicated since the verification rules were not settled as (now) described in Section 6.2.1. Nevertheless, the model (shown in its full representation in Figure 7.7 and Appendix C.3) was checked for correct syntax and appearance (the style guide). Many errors were corrected during the implementation phase; the *r.VER* instruction was used multiple times to put the model in a syntactically and stylistically sound state. However, some other issues are still present as they were retrieved after the finalization phase, e.g.:

- Not all associations have a name attached
- Groups are missing a multiplicity
- Some associations are not drawn according to the recommended style
- Naming is sometimes inaccurate with regard to the background process (e.g., Requesting instead of Inviting, the Person as a vague role name)
- Arcs of crossing lines are not always drawn horizontally
- Many deep roles are used
- Events (as entities) do not carry a shadow

As future work, the model would be corrected and maintained in order to eliminate the issues stated above. For all of them, a solution is possible. The new versions would be made as evolutionary changes and recorded in the GPD.

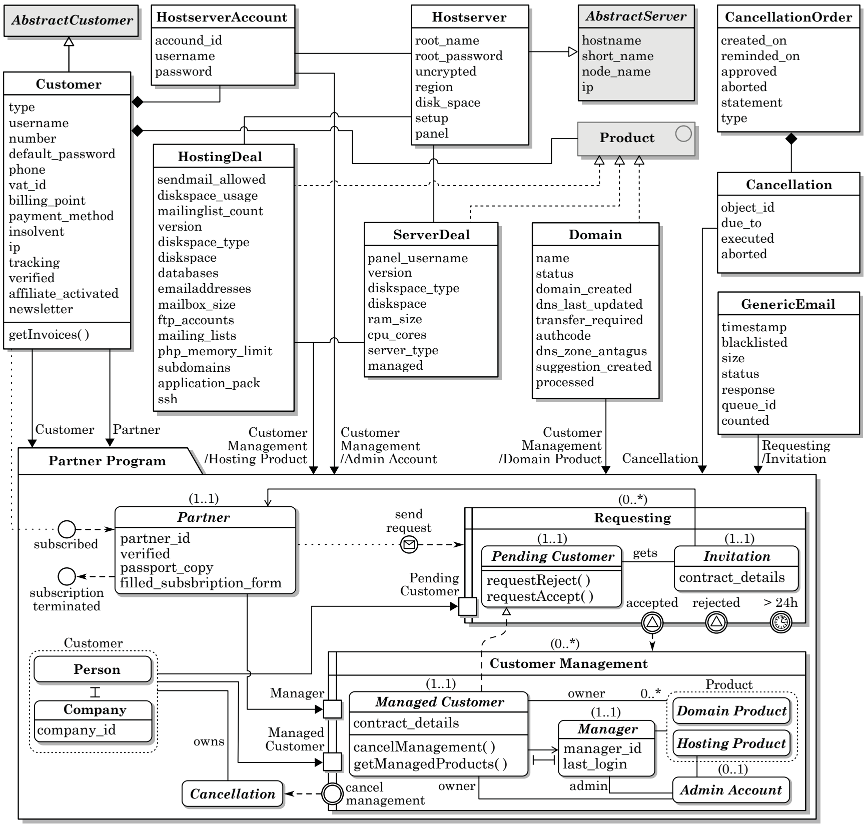


Fig. 7.7 The CSP's final application model as BROS adaptation of the reference model (from [222]; a large version of this model can be found in Appendix C.3)

Validation Check (r.VAL) Contrary to the syntactical check, the *r.VAL* instruction checks the used semantics of the model (in particular with regard to the *p.AOD* instruction). Since every instruction was done together with the CSP's stakeholders, the semantic mismatches are reduced to a minimum (like developing a software project in an agile style, e.g., [53, 234]). As the illustrated model in Figure 7.7 was constructed step by step (or, rather, instruction by instruction), it was easier to detect semantic inconsistencies. However, after the finalization phase, some known semantic errors did not pass the checks of the *r.VAL* instruction and are in need to be corrected in the next model evolution, e.g.:

- The implication can not occur in its current state as the scene `Requesting` ends before the scene `Customer Management` starts. This leads to the fact that both involved roles are never valid at the same time. A possible solution would be the usage of tokens.

- The instances (respectively, the identities) of the roles `Cancellation` and `Invitation` have to be created newly while the other roles' identities are "chosen" from existing ones.
- The interpretation of the model is not harmed, however, sometimes unclear (e.g., whether the association `admin` between the roles `Manager` and `Admin Account` represents a new association or refines the existing composition between the objects `Customer` and `HostserverAccount`)
- The adaptation consists of some unnecessary items (e.g., nested scenes) that can be either replaced or removed.

Besides the issues above, the model holds for many other checks of the validation rules (see Section 6.2.2), e.g., all contexts are aligned in a correct way, events are covered by the background process, the goals and boundary are met. Rule number 12 is, however, prone to errors as many sources of errors are possible (like listed above).

Feedback Loop (r.FBL) The usage of the feedback loop (see Section 5.3) is optional, dependent on whether the user of the reference model found any misconception in the original reference model. In this use case, no reference model misconception had to be reported. Even if some misconceptions would arise, the way to the reference model owner is concise, thus, the reference model could be adapted quickly by the owner (especially as this reference model is an in-house model and in the CSP's own area of responsibility).

Documentation (r.DOC) The adaptation was hardly documented, although it would increase transparency for future adaptations. Since the CSP is usually not affected by role-based modeling (or the BROS modeling in particular), there has not yet been a suitable structure for effectively creating and storing documentation for such adaptations. Nevertheless, the findings and experiences from the use case were recorded in a structured document so that further adaptations could be made easier if necessary.

7.3.2 UML-based Adaptation

To show the benefits and drawbacks of the BROS adaptation method, it was decided to implement the new feature (cf. Section 7.1) as a common and traditional UML class diagram ad-hoc adaptation. To apply a UML-based adaptation, the reference model have to be presented in UML as well, which is the case for this use case.

UML adaptation is primarily changing the reference model and modify it towards the aimed state. It should, as described in Chapter 2, not be intended to change the reference model significantly as invasive modifications have significant drawbacks (e.g., insufficient standardization, impeded reaction on reference model evolution). For that case and to solve this issue, BROS was designed originally[9].

[9] In fact, the BROS language has developed into an adaptation-independent language that can also be used to model software without the need for adaptation.

The UML-based adaptation method was applied to see whether and how BROS differs in its functionality and qualities. For the adaptation with UML, the use case description was followed, and the invasiveness was reduced to a minimum. In particular, the latter point results in using inheritance to be most non-invasive towards the reference model. Together with the CSP's stakeholders, the alternative UML-based adaptation was conducted. Since the CSP's modelers were more familiar with UML (compared to BROS), the model is aligned with their decisions and how they would possibly realize the new functionality concerning non-invasive and simple changes. The resulting model is shown in Figure 7.8 and represents the adaptation based on inheritance; the darkly shaded classes are the newly introduced ones. For the adaptation, the reference model (cf. Section 7.2.1) was used as a template, and the background process (cf. Section 7.2.2) guided the design decisions.

The UML-based adaptation was made following the non-invasive principle and accurate reproduction of the use case's feature. Together with the CSP, it was done ad-hoc and did not follow a defined guideline. The design decisions that led to the final adapted model are summarized below. The final model is a possible solution and not the only viable UML-based solution.

Partner Program The class `Customer` can subscribe to become a partner, thus, a `Partner` class is added as an inherited class of `Customer`. It has several fields related to its functionality. As a partner, the class has a relationship with other customers in order to manage their resources (`HostingDeal`, `ServerDeal`, and `Domain`).

There is a new class `Manager` for partners that are actually managing the resources of other customers via an association with the class `HostserverAccount`.

Managed Customer The individual customer of the CSP needs to be adapted towards the use case, thus, a new class `ConcreteCustomer` was modeled that inherits from the normal `Customer` class. It represents a customer involved in a partner program. It has new fields and operations to act and react within the use case (e.g., `requestApprove()`). The `ConcreteCustomer` is associated with the `Partner` and `Manager`.

To model the requirement that a customer can be divided into a person or a company, two classes were used: `Person` and `Company`. Both inherit from `ConcreteCustomer` to be typed as such. The `Company` carries the extra fields required by the use case.

Invitation and Cancellation To include the invitation concept of a `Partner` for a `ConcreteCustomer` the `Invitation` class was added, which inherits from `GenericEmail`. In contrast, the `Cancellation` object is already available in the reference model. Thus, a new class for the cancellation of the management process is not necessary, and the reference model class `Cancellation` is associated directly with the `ConcreteCustomer`.

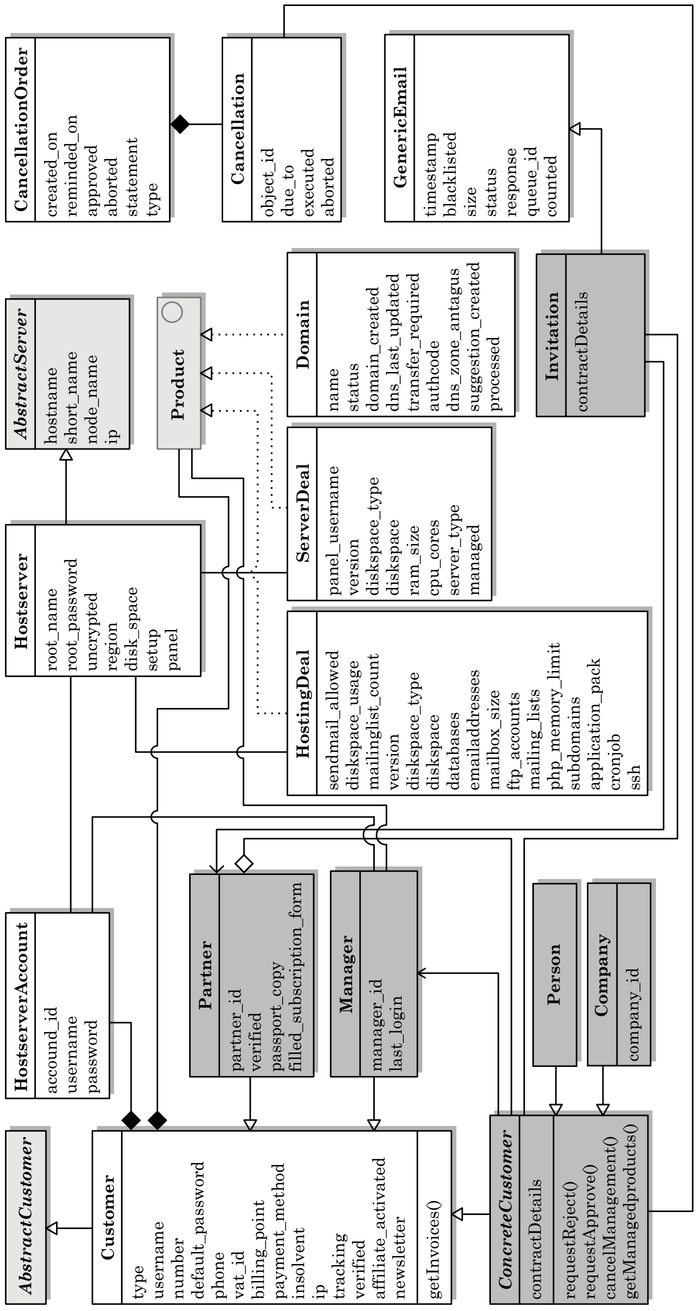


Fig. 7.8 The CSP's alternative application model as UML-based adaptation (from [222])

7.4 Approach Comparison

After building the model for the use case in BROS and UML, this section compares both approaches concerning the language and the adaptation method. It provides an overview of both approaches together with best practices and lessons learned from the use case.

A comparison can be made in the most different ways and abstraction levels. The method of comparing two things with each other can vary broadly as well. The thesis at hand presents a comparison in six aspects of the language and the adaptation method and uses different criteria for each aspect. The aspects are based on functional and non-functional requirements, which were subject to investigation, and the results of the comparison are derived from a discussion workshop with modeling experts from the CSP. The comparison of the use case can not represent every single adaptation use case since every adaptation is different. However, most results can be generalized as conclusions for a multitude of adaptation use cases.

> The comparison is intended to show the benefits and drawbacks of the BROS language and adaptation method to assess the general adaptability provided by the approaches. It is not intended to replace a extensive user study or formal evaluation. The comparison highlights the differences, problems, and solutions that have been identified in the construction and final model of the two approaches. Both approaches are not granted an exact value, and the results are presented in a purely objective manner.

The comparison results of the six aspects are summarized in Table 7.1, with a particular focus on expressiveness. A black circle (●) indicates a well-supported property of that adaptation approach and is used as a highlight of the respective approach. A half-filled circle (◐) represents that the property is indeed supported; however, it is not characteristic for the respective approach: some elements foster the property's concepts but are rather secondary and may lead to other drawbacks. A white circle (○) indicates that the approach is not able or intended to support the property on a large scale, even if there are already existing concepts to this. The comparison and the summary table originates from Schön et al. [222] as well. Nevertheless, the aspects within the table are slightly extended and regrouped, partly re-assessed, and explained in more detail. The detailed description and evaluation statements are listed below.

Table 7.1 Comparison of the BROS and UML-based adaptation method (adapted from [222]) (● = well supported, ◐ = semi-supported, ○ = barely supported)

	Expressiveness	Simplicity	Flexibility	Decoupling	Integrity	Time Representation
BROS	●	◐	●	●	○	◐
UML	◐	●	○	◐	●	○

Conceptual Expressiveness

The conceptual expressiveness[10] assesses the possible model statements that can be derived by applying and using the respective language. An adaptation method that exploits a more expressive modeling language is usually able to provide more accurate model statements. The constructed use case BROS model in Figure 7.7 has, without doubt, many more elements (regarding both quantity and types) than the UML model in Figure 7.8. A large set of elements and element types can be of advantage but also come with major drawbacks.

> "On the one hand, more elements often provide more expressiveness. On the other hand, a large amount of model elements frequently results in hard to read and more complex models." [222, p. 8]

In the use case, it was observed that the usage of the BROS adaptation method has its benefits regarding the system design due to more expressivity of the foundational BROS language [222]. The property of expressiveness is one of the most important since BROS was designed to increase expressiveness while being non-invasive. During the use case study, for the CSP and related stakeholders, the increasing expressivity was a significant positive characteristic for using the BROS adaptation method over the UML-based variant.

Further, a targeted structure (e.g., implied by a requirement) can be achieved with possible more expression accuracy as the extended set of model elements can be used to model fine-grained and more suitable statements (including behavior-aware statements). As the BROS language is based partly on UML and CROM, fundamental concepts of both were adopted and extended. This fact leads to use cases that can benefit from the extended features of BROS compared to standard UML.

The context-oriented language design is able to push the separation of concerns actively. Roles that belong to a particular scene or compartment are acting in a settled environment. It was experienced that

> "[. . .] encapsulation of roles (representing the available objects) within scenes as a collaboration of participants makes adaptions explicit and maintainable." [222, p. 8]

Context as an individual and explicit modeling element can lead to new system design (in UML, the context is still missing). Thus, use cases that make use of context as core element profit from using BROS. Such context is a significant increase in conceptual expressivity.

A given use case can be captured more naturally since use cases often make use of the role concept implicitly: personal roles (e.g., customer, clerk, driver, employee) but also object roles (e.g., company car, rental, purchase agreement, storage, conference room, crusty pizza) are frequently used in user stories and similar requirements that act as drivers the system layout (e.g., "A customer can sign up other people as camper drivers."). This leads to the fact that the roles may be already known and, therefore, can be used easier and more effectively to design the conceptual enterprise model design compared to the UML variant (where roles are not individual model elements).

[10] This property was originally called "Adaptability" in Schön et al. [222]. However, there was some confusion and misunderstanding with the naming.

> "Instead of introducing new subclasses with fixed type and identity, the roles can be played or dropped by an object if a change in the object's representation in a certain context is needed." [222, p. 8]

For example, during the use case, it was experienced that, in BROS, it is effective to make a distinction between `Partner` and `Company` as roles of `Customer` and add both to the application model as new structure added to existing structures. In contrast, with UML-based adaptation, there is often only the inheritance concept to stay non-invasive. Indeed, inheritances are still possible and a powerful concept. Always inheriting from existing objects has some major drawbacks, though:

> "Creating `Person` and `Company` subclasses has, however, the disadvantage that we always have to decide between either using, e.g., a `Company` object or a `Partner` object, as we do not have"combined" subclasses like `PartnerCompany` (except with multi-inheritance, which will lead to new problems, though). However, even if we would use a mechanism for combined subclasses for different property dimensions (e.g., the multi-inheritance), this would lead to huge "inheritance trees," resulting in possible objects like `PendingPartnerCompany` that are far from being "real" business objects and are more like states." [222, p.8]

The static nature of UML objects partly leads to a loss in conceptual expressiveness (compared to the dynamic identity nature of roles) since objects can not switch their structure and behavior according to new circumstances at runtime like roles. Neither can a `Customer` switch to a `Partner` (without destroying the object an recreating it) nor can an object have more than one specification of structure and behavior (as like to have multiple roles) of `Customer` and `Partner` simultaneously.

Role indexes for specified structure and behavior (that is, fields and operations) make it easy to define context-based functionalities of objects with roles without touching an object's identity. In contrast, objects in UML can only add new structure and behavior when using inheritance. However, UML can make use of other constructs (e.g., super-classes and interfaces) that could solve such issues if the reference model allows such constructs.

Simplicity

It has already been written in "Conceptual Expressiveness" that a large number of model elements may result in more expressiveness and increases complexity. During the use case study, there were a number of cases where, for example, stakeholders were not sure whether the model correctly mapped the requirements or how a certain structure was reproduced in the model using BROS.

There is a disadvantage when using BROS for not aligned adaptation use cases (e.g., very simple or small adaptations):

> "[. . .] inclusions of [BROS elements] within the model can lead to more difficulty in understanding. The focus on using roles as context-dependent adaptations in process collaborations can be a disadvantage if only simple (process-independent) adaptations are to be made, though." [222, p. 8]

For example, if the adaptation only consists of adapting a `Car` towards a `Camper`, the full package of objects, roles, scenes, events, compartments, and other BROS elements would be over-engineered. BROS can only be of use when the adaptation use case and requirements are stated accordingly. But if the use case indicates the play of different roles, BROS has its advantage.

In general, it was perceived that the BROS language is currently unable to keep up with UML in terms of simplicity, firstly because UML is more widely used and expertise is available, and secondly, because it appears to be easier to use due to the smaller number of concepts. Nevertheless, UML, in its full potential, can be difficult as well, especially in use cases that are not aligned with the traditional OO-based paradigm. Nevertheless, simple UML models often have a preferential application. In the future, the BROS language needs to improve in order to increase confidence in the application and understanding of the concepts.

Flexibility and Dynamics

Roles in BROS are always fulfilled by objects (either directly or indirectly via a deep role). The idea of using roles as representatives and participants in particular contexts [54] is naturally of an advantage when adapting a template reference model [220]. Roles can be introduced as they are needed to use fixed objects for certain use cases. This fact leads to a considerable advantage compared to UML, as UML is not intended to be dynamic. The traditional OO-based paradigm thrives with the class-inheritance concept (which is, indeed, a useful and relevant feature) that limits the flexibility due to fixed types [85]. Switching the types for UML objects (together with related structure and behavior) is not an easy task and often results in "delete and recreate" of the objects. Further, using objects with more than one type is an obstacle in UML as well. Since the BROS language was constructed as a flexible and dynamic adaptation language, it has, therefore, an advantage compared to UML. The concepts of BROS are aligned to be flexible (e.g., roles, events, indexes, scenes).

Considering the reusing of elements and already implemented model parts, BROS and UML seem to be equally capable. The strong separation of concerns and dynamic encapsulation of roles into scenes is experienced as a substantial benefit when: (a) reusing (or extending) existing model elements or parts, and (b) adapt a template multiple times into different adaptations.

> "In contrast, changes and extensions in a UML-based application model can be rather complicated because of the possibly high number of subclasses that have to be managed." [222, p. 8]

Reusing in UML can be achieved by introducing new inheritance relationships from already inherited classes. However, constructs that build on each other are a risk that can hinder future developments. A separation by context as in BROS is not easily possible.

Decoupling (Non-Invasiveness)

As already stated in "Flexibility and Dynamics," BROS is designed to be non-invasive. It uses the fulfillment concept to attach roles to objects that do not violate given constraints. Role indexes perform additive changes concerning the template object. Via encapsulation of roles into scenes and compartments, the reference model objects are "sorted" without changing the template.

> "The role-based abstraction from the objects helps to decouple the adaptation from the template. However, some minor changes within the code may be necessary, like changed method calls or triggers of invoker elements." [222, p. 8]

In theory, the UML approach can be non-invasive as well since the inheritance concept can be used to derive sub-classes from reference model objects (respective classes). In practice, the large number of sub-classes introduced by inheritance to be non-invasive leads to a decrease in usability (as well as readability) and increases overhead classes and inappropriate domain entities.

Integrity

The integrity of a model can be seen in many different ways. In this thesis, integrity is considered to be sufficiency in (a) formal descriptiveness, (b) technical consistency, and (c) reference model compliance. All three subjects are essential when validating and verify a model for a given use case. The importance of the integrity of an application model can vary depending on the use case: if one adapts a standardized, public reference model, integrity has a higher value than adapting an in-house reference model as the constraints are possibly less strict.

The formal descriptiveness is not given for the UML-based adaptation methods, and neither has BROS a realized formal language yet. Some approaches formalize the UML language in different ways (e.g., [44, 94]) for certain UML diagram types, however, they are not standard and not considering the adaptation of reference models. Since CROM already established a formal language [150], it can probably be adapted to support BROS as well. BROS has some formal constructs that were used for internal development reasons but are not officially released (see Appendix C.1). In comparison, UML has more possibilities regarding formalization.

Technical consistency is given in UML and BROS as well, whereby UML can rely on a long history of its application. UML is the status quo if it comes to system design, especially the UML class diagram. Current reference models are, if modeled in full details, very often modeled in UML. This is also the reason why, during the development of BROS, it was a requirement to be able to work on current UML reference models. The technical details specified in UML are foundational for many implementation issues, and UML patterns were developed to deal with implementation problems. Nevertheless, UML abstracts certain details as it is a modeling language (e.g., the context of operations). BROS, which is partly based on UML (and CROM) design decisions, can take over many concepts from it that are useable for technical specifications. However, especially the role part of BROS is not that continuously developed in terms of technical pervasiveness. Although some approaches are already implementing the role and related concepts on a technical side (e.g., [121, 159, 251]), it actually can not compete with the possibilities of UML.

The last point, the reference model consistency, is a conceptual constraint. UML has, compared to BROS, fewer possibilities of changing given template entities and relationships. Inheritance as the main adaptation mechanism can only be used for a limited number of adaptation operations. Nevertheless, this limitation leads to higher compliance with the reference model since the inheritance only refines the reference model entities. Also, other concepts of UML are highly standardized and, thus, more reliable concerning the reference model. In BROS, the concepts are standardized as well but on a smaller scale. Its novelty requires more input from practitioners.

> "In BROS, a role may change an object severely. Applying a role may theoretically lead to a complete redefinition of an object, e.g., removing everything and adding new operations. While this is in the responsibility of the modeler, the integrity towards the original model is hardly ensured in BROS." [222, p. 9]

The possibility to change the complete reference model objects does not contribute to the reference model consistency, even in a non-invasive way. Then, the modeler is in charge to comply with the needed integrity.

Time and Behavior Representation

The temporal description that is using behavior information for adaptation or even represent it in the model is often used to model new features and requirements that affect the reference model (or the already adapted application model).

> "[. . .] BROS has the huge advantage of a temporal modeling element, called events. With this feature, BROS can model statements like, e.g., "after 24 hours without reply the invitation becomes invalid" [. . .]" [222, p. 9]

Considering UML, temporal expressions and statements are not possible to implement natively but only by secondary notations (e.g., comment boxes) or complex extensions (e.g., the MARTE UML profile [232]). Although BROS provides events and scenes as a temporal behavior description, the elements can not represent a complete workflow or process (which is not intended, though).

> "The temporal specification that comes with the event feature, which abstracts the time affecting the structural elements (despite the drawbacks) is a fully new feature the UML-based approach does not support at all. However, its specification is not comparable with a fully-featured procedural model." [222, p. 9]

There are mechanisms to bind events towards existing background processes, but BROS models can not serve as process models. This is not considered as a drawback, however, it decreases its relevance in the category of "Time and Behavior Representation". In general, the usage of events and scenes was well accepted in the modeling process of the use case's involved stakeholders as they provide a useful adaptation mechanism and are reasonably easy to understand in their nature and application.

7.5 Use Case Summary

The use case described in this thesis was the first-ever implemented application of a BROS adaptation in the "real world" with an existing and industry-related reference model and background process. The cloud service provider enterprise and its partner program use case gave a foundational insight into the complex integration of BROS concepts. It was the first attempt and still has its improvable issues. Nevertheless, it shows that the BROS language can be used for the objectives it was designed for: as a language for a systematic adaptation method, consisting of high structural and behavioral expressiveness, able to work on object-oriented (industry) reference models (see Section 2.2.3). Although there is room for improvement, pointed out by the use case study, the basic functionality of BROS was shown.

The models developed during the use case implementation only show one possible option, and many different versions could be viable as well. Further, the use case study can not replace an extensive user study or qualitative expert interviews and surveys. Instead, the use case was used to verify the BROS concepts and their application regarding a non-exemplified reference model and background process. Besides the use case shown in this chapter, BROS was used, demonstrated, and tested in several other applications (e.g., in the pizza use case [223], in combination with requirements engineering and capability modeling [224], or at a very early stage as a small flight ticket system [220]). Surely, BROS needs further propagation into industry and modeling applications.

According to the experiences made in this use case, BROS can excel particularly through its integration of processes and contexts. The possibility of introducing business behavior is an advantage for conventional business modeling regarding the structure. Events and scenes can be used to derive model statements that are reflecting the actual requirements regarding business logic. In combination with roles, the overall approach is in its expressivity ahead of the traditional UML-based adaptation. In particular, the roles as an abstraction of the reference model objects are an effective and non-invasive adaptation mechanism. Roles as representatives and participants in a business logic context can express a given object in an excessively individualized perspective. Basically, each requirement can make use of its own role in its own contexts, based on the same reference model object. The contexts introduced by BROS are a benefit compared to UML since UML does not offer such constructs. A context can, if chosen appropriately, be of enormous advantage as it separates different concerns in an easy, naturalistic, and understandable way. As a conclusion of the use case experiences, this increased expressiveness, where any element can be used optionally, leads to more detailed and individual application models.

> "In sum, we consider BROS as a more advanced conceptual modeling language due to more expressiveness especially with respect to business logic and in adaptations targeting the technical perspective of software systems." [222, p. 10]

Nevertheless, BROS is not a solution for all modeling problems. There are two significant disadvantages identified during the use case adaptation implementation (also represented in Table 7.1 as BROS' drawbacks):

1. The technical feasibility (and formal description) is not developed continuously (as stated in "Integration" of Section 7.4). Although the BROS language is based on actual technical details, it has some drawbacks considering the implement-ability in technical terms. Some BROS concepts are still unspecified with respect to their technical realization (e.g., tokens, events). UML is, in contrast, often clearly defined its technical feasibility.
2. The second issue is concerning the usability of the BROS language (also issued by "Simplicity" in Section 7.4).

   > "BROS usability seems to be less than the UML-based approach. While most modelers are confident with UML, BROS is rather new, and a modeler needs to familiarize with the BROS language concepts. The resulting models are quite complex and harder to read, which may also foster errors when modeling or reading BROS models." [222, p. 10]

   The drawback can lead to inconsistent, discrepant, or false used models and model elements. Thus, awareness of the language has to be increased.

In conclusion, the use case was a success in terms of execution and findings. The results show that (a) BROS can be used for real industry-related models, and (b) the BROS language has its benefits compared to the traditional UML-based adaptation. Despite its drawbacks, BROS offers a systematic method to adapt reference models in a new way, combining structure and behavior in a single concept without losing its precision. If the use case is of a rather simple nature, and the adaptation task does not require the extended expressivity, BROS can not compete with the UML-based approach. However, when applying BROS for use cases that are amenable for the new concepts, the application model can usually be modeled in higher expressivity and accuracy regarding the use case's requirements.

Acknowledgements Many thanks to the involved (anonymized) cloud service provider for providing internal information about the used reference model, system structures, and business processes as contributions for this case study as well as the excellent cooperation.

Chapter 8
Conclusion

"We both said a lot of things that you are going to regret.
But I think we can put our differences behind us.
For science. You monster." – *GLaDOS, "Portal 2", 2011*

After the main components of the BROS methodology and related areas have been stated in all previous chapters, it is the task of this chapter to summarize and conclude the topic. The role-based paradigm in software engineering is not a new one but has not yet developed its full potential. There are a lot of missed opportunities in which roles could have a tremendously positive effect. Not all problems can be solved with roles, but we live in a role-based world and, thus, software engineering should be aligned to that, too. The BROS methodology has done its part in conceptual modeling, but there are still many issues that need to be tackled for BROS as well as for other role-based applications. But inevitably, roles will, at least in their native natural understanding, always affect the subjects related to software engineering. And therefore, the applied arts of software engineering will always carry a role-based essence.

8.1 The Value of Role-based Adaptation

Roles are an important and useful concept that definitely needs more representation in software engineering, especially modeling. Although already established in several works (see Section 3.3), roles are still underrepresented regarding their potential. This thesis aimed to increase the perception and value of roles in modeling software systems, a fundamental subject of software engineering. The developed BROS language and method can not solve all existing issues, however, it plays its part in advancing (reference) modeling and adaptation as a discipline.

Role-based adaptation, as described with BROS in this thesis, has some fundamental advantages and benefits that are already frequently highlighted in this thesis (e.g., in Section 7.4). Still, to close the thesis with a more general view of BROS and answer the research questions appropriately, this section states the three most essential parts of why using BROS can improve the reference model adaptation and role-based modeling in general.

1 *The combination of structure and behavior.*

Structure and behavior of a domain are two essential points that need to be taken into account when designing software systems and enterprise models in particular. Considering context, roles, and time constraints, a system can be more adequately described through more expressiveness. BROS combines both parts but prioritizes the structural part. This so-called "behavior-aware structural modeling" is considered to be a solution for adaptations that are driven by new features and functions described as business processes. Structural modeling is needed in order to construct and build systems. Nevertheless, modeling structure that can be dependent on behavior impacts is of advantage for many purposes.

BROS uses multiple concepts to combine both viewpoints, the structure and behavior side. This contributes to the research question 3 ("How to express certain business logic in a structural adaptation?"), stated in Section 2.2.2 and complies with the research objective 3 ("Structural and Behavioral Expressiveness," cf. Section 2.2.3). To achieve this, BROS introduces and uses concepts that cover the foundational natures of model elements: objects as fixed and determined identities, roles as dynamic and fragile representatives of objects, scenes as a context for roles and events, and events as instantiated points in time, as well as other elements (see Sections 4.4 and Section 4.5). This leads to a modeling strategy that is different from traditional UML modeling. With the role-based modeling paradigm, which captures the structure with respect to the behavioral side, BROS can be applied to improve many modeling use cases (adaptations as well as modeling from scratch). The novelty and recent introduction of BROS do not do any detraction to this, as the BROS methodology can be developed further, and the concepts can stand on their own (and thus may also impact other modeling approaches).

2 *Non-invasive and traceable (business) object adaptation.*

The second important benefit of role-based reference modeling with BROS considers the decoupling property. In essence, the often mentioned non-invasiveness, as well as traceability of adaptation-based changes, leads to an entirely new policy in modeling. Instead of making changes and modifications directly in the reference model, with BROS, the modifications are exclusively made in the role-based application model part and do not interfere with the reference model (see Section 4.1).

BROS has, as one core functionality, the fulfillment mechanism to let objects play different roles at runtime (see Section 4.4.2). Fulfillment relationships do not change the fulfilling object invasively: they bind a role to an object's identity and change its structure and behavior via indexes dependent on specified contexts. This is a novel construct for reference model adaptation. The adaptation does not harm the reference model as a template, as, e.g., an ad-hoc modification of new structure or behavior would do. However, the fulfillment is not the same as inheritance either: while inheritance extends its generalization object (or class) to add new structure and behavior, it still relies on (and is fixed to) the inherited type, independent from any context. The inheritance is a very effective mechanism in object-oriented modeling, but it can not be compared with roles as both are entirely different mechanisms. Besides its non-invasiveness, the fulfillment offers a second property: due to its "attaching" behavior, fulfilled roles can be traced and easily spotted in a model. The sole adaptation mechanism in BROS (concerning the objects) is done via fulfillments. If, for example, a reference model evolution occurs (cf. Section 5.4), the adaptation can easily be re-designed to match the provided new objects by the reference model. Thus, the BROS adaptation method (see Chapter 5), in combination with its non-invasiveness and traceability, contributes to the research question 1 ("How to adapt a structural reference model in a systematic way?" cf. Section 2.2.2) and the related research objective 1 ("Systematic Adaptation Method," cf. Section 2.2.3).

3 *Multiple adaptations based on existing reference models.*

The third substantial advantage regarding the BROS' role-based adaptation is considering the flexibility of multiple and diverse adaptations based on (already established) industry-standard reference models. When developing a new approach, it is recommended to stick to established conventions and related works in order to convince other people to accept the developed artifacts and possibly continue their development. One of the main requirements, according to the BROS development, was that it should be able to function on already established reference models in the industry. This requirement is an enabler to distribute the BROS methodology further.

To serve this requirement, BROS has the benefit of using roles for (business) objects. The fulfillment concept can be used for regular OO-based reference model entities (e.g., classes in a UML class diagram reference model). The property of non-invasiveness (cf. Section 4.1) is an essential driver of this fact: roles are attached to given OO-based objects and modify them by role indexes towards the needed use case adaptation. Thus, BROS can also be used for (nearly) every OO-based reference model that fulfills one assumption: it has to have the right granularity. BROS adaptation has to be based on a structure that has dedicated objects that can be adapted (cf. Section 4.2.1). One the one hand, a too coarse-grained reference model does not benefit from an adaptation with roles and scenes as the provided objects do not reflect the necessary identities. On the other hand, a too fine-grained reference model suffers from the sheer amount of possible object identities as well as more limitations regarding the choice of possible players. But if this assumption is respected, BROS can help to adapt common and established industry reference models that are already in use. This contributes directly to the research question 2 ("How to reuse standardized, object-oriented reference models for adaptation?", cf. Section 2.2.2) and the research objective 2 ("Reuse of OO Models," cf. Section 2.2.3).

8.2 Contributions of the BROS Methodology

In this thesis, several artifacts were developed, all aligned with each other. To achieve the objectives stated in Section 2.2.3 (which answer the research questions in Section 2.2.2), the artifacts were previously provided in individual chapters that are assigned to the single objectives. This section states the main artifacts of the BROS methodology in a short way as a summarized overview, as shown in Figure 8.1.

BROS Modeling Language

The BROS language is the foundational artifact of the thesis (see Figure 8.1). All other artifacts are based on the developed modeling language that utilizes roles, events, and scenes to adapt and specify objects. Its full name, "Business Role-Object Specification," represents the intention and purpose: to specify (given) business objects and transform them into role objects. Primarily used on a conceptual level, BROS unfolds its full potential in software architecture and design issues. In combination with a reference model adaptation use case, BROS can help to individualize an OO-based template model with higher expressiveness and loose coupling.

The BROS language is presented and specified in Chapter 4. It lists its intentions and goals (cf. Section 4.1) to enable a classification in the software development stack. The major language design goals are stated and explained in separate sections. This is followed by the explanation of two major characteristics of BROS, the structural and behavioral background information (cf. Section 4.2), and how to use

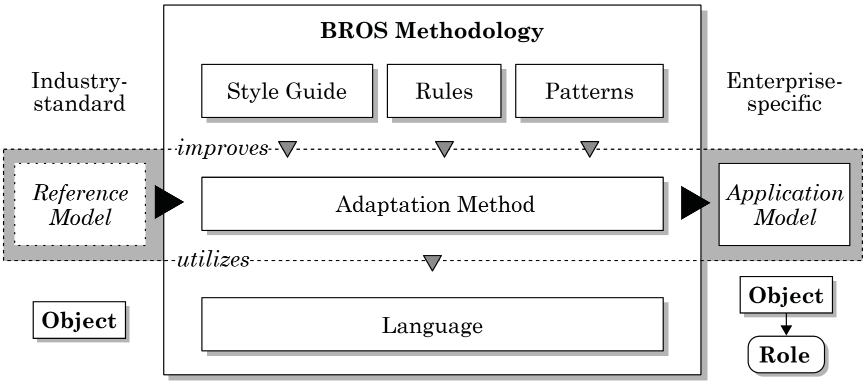


Fig. 8.1 The main artifacts of the thesis in relation (copy of Figure 2.6)

both for a successful adaptation and software design model. The proposed metamodel (cf. Section 4.3) specifies the formal part of the language: the elements and their relationships. At last, all available BROS concepts are listed, described, and explained, divided into the core concepts (cf. Section 4.4) and extended concepts (cf. Section 4.5).

The BROS modeling language chapter contributes to objective 3 (“Structural and Behavioral Expressiveness,” cf. Section 2.2.3) as the language provides the tool for the extended expressiveness. Further, it is part of the general objective 2 (“Reuse of OO Model”). The BROS language was published (partially) in Schön et al. [223].

BROS Adaptation Method

The second artifact is considered to be the BROS adaptation method. It describes the usage of the previously described BROS language to adapt given reference models towards a tailored application model via a systematic way (see Figure 8.1). The BROS adaptation method makes use of several phases that contain several instructions for adaptation implementation. The BROS adaptation method is designed as an iterative, three-part specification process (context, structure, and behavior), with an upstream preparation and subsequent review. The instruction-based design of all phases allows for two advantages: (a) the instructions to specify the adaptation can be individualized as well (by choosing and sorting the instructions), and (b) the set of instructions can be further extended. In general, documentation is more straightforward with detailed instructions as atomic adaptation parts.

The BROS adaptation method is described in-depth in Chapter 5. It introduces the core ideas of adapting a reference model with BROS, including all participating components, different variants, phases, and the instruction set (cf. Section 5.1). This is followed by the implementation description by stating all phases and related instructions to adapt a given OO-based reference model into an individually tailored application model (cf. Section 5.2). The chapter and the related artifact is concluded with an explanation of the feedback loop tool (cf. Section 5.3) as well as stating best practices and limitations of the adaptation method (cf. Section 5.4).

The BROS adaptation method chapter contributes to all three objectives: objective 1 (“Systematic Adaptation Method”), objective 2 (“Reuse of OO Models”), and objective 3 (“Structural and Behavioral Expressiveness,” cf. Section 2.2.3), as the adaptation method is the binding point for all related artifacts (see Figure 8.1).

BROS Modeling Practice Manual

As a third major artifact, the BROS modeling practice manual is developed and described in this thesis. It consists of three subordinated subjects: the style guide, rules for verification and validation, and a (small) set of modeling design patterns. This artifact states the “How to model correctly and appealingly.” While the basics of modeling are stated and defined in the former chapters (the BROS language and the adaptation method), the modeling practice is not foundational. It does not interfere with the basic definition and is, so to say, a bonus. Following the modeling practice manual and applying its guidelines, rules, and patterns, lead to simpler and beautiful models, checked for syntax and semantics. The modeling practice manual helps the modeler dealing with the possible complexity and novelty of BROS.

The complete modeling practice manual is stated in Chapter 6. It is, as described previously, divided into three main sections:

1. The style guide (cf. Section 6.1) refers to a set of guidelines on how to model the general model (with respect to readability, simplicity, naming, and general directives), individual model elements, and adaptation elements in a simple, effective, and beautiful way. It was partly developed on the basis of Ambler [11], who developed such a style guide for UML 2.0.
2. The consistency of a BROS model (cf. Section 6.2) is divided into syntax checks (verification) and semantics checks (validation). In contrast to the style guide, the consistency is a set of rules that have to apply to the modeled BROS application model. As every application model is different, the rules are more generic but still tangible.
3. The last part of the chapter states a set of BROS modeling design patterns (cf. Section 6.3) derived by use case discussions. Although the set of patterns is rather small[1] (six patterns of different complexity), they are intended to demonstrate the generalization of BROS constructs related to specific modeling problems. To this point, the sample patterns provide a good overview of how to encounter modeling problems with generic solutions.

The modeling practice manual contributes to objective 1 (“Systematic Adaptation Method”) and objective 2 (“Reuse of OO Models,” cf. Section 2.2.3) as its three major parts (style guide, consistency rules, design patterns) are involved into the adaptation process with BROS (see Figure 8.1).

BROS Application Demonstration

Besides the primary artifacts above, the use case should be highlighted as well, as it demonstrates all the other artifacts in a combined way: the utilization of the BROS language for an adaptation regarding the BROS adaptation method while complying with the BROS modeling practice guidelines and rules. The use case is based on a real reference model and background process and was the first real-world application of BROS in cooperation with other modeling experts. The use case artifact shows that BROS can actually be used to model a specific application model (feature) and can help the modeler to express individual requirements. Further, the use case was taken to identify the benefits and drawbacks of the BROS adaptation by comparing it to standard UML-based adaptation of the same reference model. A subsequent comparison was made to analyze the differences.

[1] This set is undoubtedly subject to extension in future developments when more reliable real-world modeling use cases are practiced.

The use case is stated in Chapter 7. It starts with an introduction into the use case environment, including the description of the cloud service provider (CSP), the description of the problem statement, and the feature to be implemented (cf. Section 7.1). This is followed by an explanation of the adaptation's environment (cf. Section 7.2), most importantly, the reference model and the used background process. As the chapter's central part, the adaptation is implemented using two different ways: first, the BROS adaptation method, and second, a traditional UML-based adaptation (cf. Section 5.2). With the help of the CSP's domain experts, the two resulting application models are then finally analyzed and compared with each other in order to identify the significant differences and whether BROS achieves its goals (cf. Section 7.4). The results of the complete use case are summarized at the end (cf. Section 7.5).

As an actual implementation of the adaptation method (together with the language and modeling practice manual) based on a given reference model and background process, the use case contributes to the objective 2 ("Reuse of OO Models," cf. Section 2.2.3). The BROS use case study was published (partially) in Schön et al. [222].

8.3 Limitations

The BROS language, the adaptation method, and modeling practice manual are, of course, neither a universal solution to all adaptation problems nor do they always fit the adaptation use case. Besides all its benefits and the advantages of role-based adaptation in general (see Section 8.1), BROS comes with several limitations and drawbacks which are not to be concealed. But not only BROS should be considered here, but also the limits of the thesis itself are explained in this section.

Boundaries of BROS

The drawbacks of BROS are an essential limiting factor when considering the limitations of the whole thesis. The specific drawbacks are already stated in the analysis and comparison of the use case (see Section 7.4), thus, they will not be added here again. It is a fact that BROS can not deliver excellent performance and be applied in all cases.

First of all, the BROS language might be missing some features or concepts that are necessary to adapt specific use cases. There are use cases that can not be adapted with the BROS core or extended concept set. In such cases, either there are workarounds or patterns available (like "shared return" pattern, cf. Section 6.3), or the problem has to be solved with another approach. The combination of BROS with other approaches (e.g., aspect orientation) has not been investigated yet.

Further, the set of statements representable with current BROS concepts is limited to use cases that benefit from using roles, events, and scenes. For use cases where the adaptation is such simple that it is not necessary to construct a BROS model (e.g., add a field `solarpanel` to the `Car` object), applying BROS would probably be over-engineered for the use case as it is neither necessary to use a role for it nor to put it into a specific context[2]. Also, including too many BROS elements (e.g., scenes) for simple adaptations will reduce the usefulness of the BROS methodology for the use case due to less readability and increased complexity.

The mentioned drawbacks analyzed in Section 7.4 are, after all, a limitation of BROS as well. Especially the technical implementability, as well as its integrity (the formal language in particular), are limiting factors that might reduce the applicability of BROS for use cases. Both things are needed in order to ensure consistency within the whole development stack. BROS models that are neither formalized (or checked formally) nor transformable to source code limits the BROS model to a purely conceptual overview without further meaning. While this can be intended by the stakeholders, often, a model is considered to be a blueprint for software development that goes further than an overview model.

Considering the conceptual side of BROS: the BROS language allows for model statements that are (at least) questionable. The BROS language was designed to be as expressive as possible, leading to design decisions that can result in inconsistent models if applied in the wrong way (e.g., the possibility of relationships to cross the context boundaries). The modeler is in charge of modeling the BROS model in a consistent way; nevertheless, this can not be ensured. Although the verification and validation rules are an aid, they can not provide any guarantee in this respect.

Last, as the BROS language is a reasonably new modeling language, the ideas and design decisions might be wrong. The development of the BROS methodology (the language, the adaptation method, and the practice manual) are considered to be correct, however, some modifications will surely be necessary (e.g., to solve misconceptions). The BROS language requires further real-world applications to demonstrate and prove its full potential as a modeling language capable of what it promises. Further, the BROS adaptation method is based on the experiences made from several use cases, however, it has to prove itself as well. Both are in need of time and further evaluations. Although BROS is based on UML and CROM (which are indeed more sound in their developed landscape than BROS), the improvements introduced by BROS go beyond them. This extension could lead to inconsistent states regarding both the adaptability of a use case with the new concepts and the combination with already established concepts of UML or CROM.

Environment Assumptions

Not only BROS in its dimensions, but the dependency of BROS regarding its environment is limiting its application as well.

The first assumption about BROS is that it is a conceptual modeling method for the early phase of software development (system architecture, design, and specification). It benefits from given (or new) requirements that need to be implemented on given

[2] Only if this is the only modification so far. Otherwise, adding a `Car` role within an (at least static) context like `RTC_car` could definitely make sense.

structural models (either a reference model or any other given structural OO-based model). Contrary to that, BROS' value stagnates in the further software development process as the initial phase is done so far, and the adapted application model remains unchanged (for the moment until the next adaptation occurs). Thus, in highly progressed software projects, newly introduced BROS will diminish in value. Especially when already implementing in technical terms, introducing BROS can be a difficult task. It is recommended to include BROS (and the whole role-based paradigm) as early as possible in the software development stack.

Secondly, BROS is dependent on the domain side (or the reference model in particular). Not only must the reference model be detailed enough, but the setting of the domain has to be right. BROS can not improve a domain that is not sufficiently built for BROS. This issue holds in particular when the domain is not object-oriented but, e.g., transactional. As an example, the domain of "IoT internal technical device" could be unsuitable if it contains too many technical classes and too few business objects (e.g., the objects `MessageController` instead of `Message`). Thus, the domain has to be suitable in terms of its OO structure.

As written in the previous chapters and sections, the BROS adaptation (together with its language) comes with several drawbacks with regard to the expected reference model. Besides the right domain structure of the reference model, it needs to be just in the right granularity to be reasonably adaptable by roles. A too coarse-grained or too fine-grained reference model limits the degree of freedom when choosing and applying roles, contexts, and scenes. This is the main assumption taken when using BROS for reference model adaptation. However, if using BROS for a greenfield approach (and designing systems from scratch), the right granularity of objects is within the responsibilities of the modeler.

BROS uses domain information to design systems appropriately. If that information is not given, BROS can not be applied in such a way that it reflects the domain. Besides the reference model, the background processes are especially crucial for domain information. Without background processes, the scenes and events can not be based on the enterprise's business processes, and the structure loses its "behavior-aware" attribute. Of course, behavior can be included in any case (that is, using scenes and events without any background process coupling), however, that would result in a system design that can not be validated (it would rather be a system build from scratch). Background processes can be derived from anything that describes a behavior flow (e.g., requirements' plain text, BPMN models, event chains, cf. Section 4.2.2). If this is not the case, and the domain does not provide any further information, the adaptation can reach its limits.

Another reason why BROS might be limited in its application is that the enterprise needs to be prepared for the whole role paradigm. Modelers can use BROS as a modeling tool for their ideas; this is feasible and, if applicable, can help to express specific statements. Nevertheless, using BROS for an operational system design without preparing all stakeholders and systems into the new paradigm can lead to project failure. Changing and introducing a new paradigm is something that goes beyond the mere modeling of requirements and affects all responsible parties involved, from business managers to developers. But BROS can help to illustrate and implement necessary changes in a simple way. Last, the enterprise must support the adaptation process and BROS modeling in general. If software development is done in an ad-hoc manner (e.g., without an in-depth elaboration of software design and architecture models), BROS can not be of advantage.

8.4 Outlook and Future Work

As a closing of this thesis, this section illustrates a future image of the BROS methodology together with possible extensions and future work.

Domain Transfer and Collaboration

Mainly to increase BROS' prominence in both the research and industry application, BROS needs to be used further in related areas and subjects. This can be done either by using and applying the BROS language and adaptation method for more use cases or using the BROS concepts in related (research) areas. This was, for example, already done in Schön et al. [224], where BROS was combined with the requirement and capability modeling area [249] to extend the usage of BROS and include the BROS mechanism into an entirely different subject. As a result, the developed metamodel is appropriate for a capability modeling approach considering BROS roles in certain events. This kind of additional and further work pushes BROS and its related ideas further into the spotlight, especially in the reference model adaptation area. As another example, a combination of BROS with the software product line domain could lead to a beneficial outcome as well.

Further, a combination of BROS (with its role-based paradigm) with other paradigms can be of interest, e.g., a combination of roles and aspects can lead to an even more expressive modeling language. An application of BROS in non-role-oriented domains may lead to new functionality and application areas. A (fewer BROS dominant) example is the inclusion of the role paradigm into maturity modeling are by Bley and Schön [30], where maturity models got extended by a role-based component to derive a new kind of maturity model type, which in turn can be of use when modeling with BROS.

Technical Implementation and Modeling Editor

Another way to improve BROS is by extending the continuity towards a technical implementation. CROM, as a conceptual base for BROS, is almost completely realized as a library for *Scala* (a *Java*-based programming language) that allows rather easy use of roles and compartments for programming by including the developed library *SCROLL* [159, 160] in the header. There are currently developments to integrate the BROS language specification as an additional feature into *SCROLL*. However, the BROS language specification needs further investigation regarding its technical implementation before events and scenes can be integrated into *SCROLL*. In general, the technical feasibility of BROS is, therefore, a rewarding task. Further, a prototype BROS modeling editor *Framed.io* was developed (see Appendix C.2). This editor turned out to be of advantage when modeling simple BROS models without large circumstances. It is web-based and, therefore, easily accessible. This editor definitely requires further development to be used as an operational BROS modeling platform, however, future features are already stated (e.g., more BROS concept support, account management, template functionality for reference models, views). An appropriate developed and easily accessed editor with extensive functionality and modeling comfort will make a difference in applying BROS for other use cases since then BROS can be tested and applied accordingly.

Formal (Adaptation) Language

Besides its technical representation, BROS can tremendously benefit from a consistent formal language that is able to do both: (a) to describe current model statements and constructs, and (b) to formalize the adaptation statements. The former includes a formal language that is capable of describing the elements and their relationships in detail (e.g., objects, roles, scenes, associations), while the latter is stating precise commands that are done to adapt the reference model. In particular, a formalization of temporal descriptions (e.g., of events by temporal and interval logic [73, 145]) could improve the BROS expressiveness tremendously. Appendix C.1 states some formal representations of BROS that were used during its development. However, it is not complete; a completely formal language can solve many current problems in BROS. Also, (formal) compliance to existing background processes of BROS models is of importance since events and scenes should always be coupled to existing processes. A first approach was made to verify a BROS model according to a given BPMN process (see Appendix B.2). This and other approaches will be beneficial tools when modeling BROS in further use cases.

BROS Extensions and Refinement

BROS is, by nature, a modular and extendable language and adaptation method. Every component of the BROS methodology is designed to be modifiable and extendable (see Section 4.1) in order to increase its expressiveness. To achieve this, the different dimensions of BROS are designed as individually composable parts (e.g., an adaptation phase consists of several instructions). The dimensions that are particularly eligible to be extended are the following.

1. *Modeling Language* The BROS modeling language can be extended by new concepts, that is, entities or relationship elements, which take over new parts or tasks in the application model.[3]
2. *Adaptation Method* As the primary process of how to adapt a reference model with BROS, the BROS adaptation method can be improved further by introducing (or refining) the instructions of the phases.
3. *Style Guide* The guidelines stated by the style guide are predestined for being extended. The different guideline sections are already structured such that new guidelines can be introduced easily.
4. *Verification and Validation Rules* Like extending the style guide, the extension of the verification and validation rules can be of advantage. The current state of both is rather generic and can firmly be improved by new and concrete rules.
5. *Patterns* The patterns are generic solutions to common problems. As with time passes, new solutions to common problems will arise, thus, the pattern set can be extended to introduce BROS more easily in modeling projects.

Further, BROS can be refined or extended with new aspects and without explicit substitution of existing parts. E.g., to enable BROS for a co-evolution would be a tremendous increase in BROS' applicability. Its already stated non-invasiveness would allow such constructions (see Section 3.2.3). Other parts of BROS would benefit from further extensions as well.

Further Real-World Application

The BROS language, adaptation method, and modeling practice manual were tested once in cooperation with a cloud service provider (see Chapter 7). It was the first insight into the application of BROS in a real-world domain. However, it requires further application of BROS to address two essential points:

- to develop BROS and its concepts further, and
- to transfer BROS into other areas and more use cases (also, concerning "Domain Transfer and Collaboration").

Both points are essential when suitably developing BROS. Use cases can help to identify missing concepts that are needed in particular use cases or to refine existing concepts that might not be defined in the right way. E.g., currently, further work is done to apply BROS on other use cases regarding customer order processes for a software development enterprise.

[3] For example, an event type for globally (instead of locally) thrown events (e.g., by a thick edge line), events that trigger relationships, or "catch" events that trigger when interrupting a scene by an exit event could be introduced.

Further Evaluation Methods

Besides using BROS for more use cases, which can be considered as a part of evaluation as well, BROS can make use of more (and different) evaluation approaches. A first insight into the evaluation of the BROS language was already done by comparing different language evaluation approaches (according to the taxonomy building method from Nickerson et al. [183]) to identify possible further evaluation approaches for the BROS language (see Appendix B.3). As a next step, one of the evaluation approaches will be selected and, thus, BROS evaluated. Nevertheless, a lot of different evaluation approaches are possible (e.g., user surveys, expert interviews, functional or formal comparisons) to strengthen the foundational basis of a BROS application and to identify issues regarding provided functionality by BROS (e.g., usability, expressiveness, missing concepts). Further, a qualitative analysis of the language and model can be done (e.g. [127, 147, 178, 259]) to identify further advantages and drawbacks of BROS. A highly tested and evaluated language can increase its trust and, thus, its application for further use cases. With that, BROS can grow and claim its place in the software modeling world.

Appendix A
Publications

A.1 Primary Publications

The primary publications contribute directly to the subjects covered in this thesis and adequately reviewed. They were written and published during the thesis' project period. Since they only cover the BROS methodology subjects partly, they are stated in this thesis in more detail.

Paper I

Role-based Adaptation of Domain Reference Models: Suggestion of a Novel Approach

(2018) **Hendrik Schön** [220]

Reference models capture domain knowledge for reuse and specification through user-defined adaptation. They work as a blueprint for either structure or processes, and the user can apply adaptations to fit specific demands. This research-in-progress paper is about a novel approach that combines reference model adaptation with the concept of roles in order to strictly separate the core elements from user made adaptations. The advantage of using roles as the sole adaptation mechanism is threefold: (a) the structure of the reference model can be combined with behavior of objects, (b) the exclusive use of roles allows broad adaptation without the restriction of other mechanisms (like inheritance) with more expressiveness, and (c) current reference models can continue to be used without modification. Consequently, the roles enriched final application model can be used to describe systems in more detail, and, if available, can be implemented with a role supporting programming language.

Paper II

Business Role-Object Specification: A Language for Behavior-aware Structural Modeling of Business Objects

(2019) **Hendrik Schön**, Susanne Strahringer, Frank J. Furrer, Thomas Kühn [223]

Representing and reusing the business objects of a domain model for various use cases can be difficult. Especially, if the domain model is acting as a template or a guideline, it is necessary to map the enterprise's individual structure and processes on the shared domain model. Structural modeling languages often do not meet this requirement of reusing structures and complying to established processes. We propose a modeling language called BROS (Business Role-Object Specification) for describing the business objects' structure and behavior for structural models, based on a given domain model and process models. It utilizes roles for a use case related specification of business objects as well as events as interfaces for the business processes affecting these roles. Thus, we are able to represent and adapt the business object in different contexts with individual requirements, without changing the underlying domain model. We demonstrate our approach by modeling a simple case.

Paper III

Adaptation of a Cloud Service Provider's Structural Model via BROS

(2019) **Hendrik Schön**, Raoul Hentschel, Katja Bley [222]

Conceptual models of specific domains provide a general overview of a software system design. Structural models, as a technical version of conceptual models, serve as development blueprints that may have to be adapted to fit special enterprise-specific demands. However, a flexible and dynamic adaptation of the design is necessary to respond fast and efficiently to new or changing requirements for reducing time and costs in the later development. In this paper, we utilize the "Business Role-Object Specification" (BROS), a role-based modeling language, for dynamic and structured adaptation. We demonstrate our approach by adapting an existing structural model from the cloud service provider domain with regard to a partner program process. Further, by comparing the BROS adapted model with a traditional UML-based adaptation, we are able to evaluate both approaches and show the benefits of the new BROS adaptation method, e.g., extended expressiveness and flexibility towards changing requirements and features.

A.2 Secondary Publications

The secondary publications do not contribute directly to the thesis' subjects but are rather cooperation publications that utilize either the role paradigm or BROS in other domains. Like the primary publications, the secondary publications were published during the project period of the thesis. Although they are related to the thesis' topic, they do not provide foundational BROS methodology knowledge.

Paper IV (German)

Gute Softwarearchitektur ist Business Value

Hendrik Schön, Frank J. Furrer [221]

Dieser Beitrag verknüpft zwei Begriffe: Softwarearchitektur und betriebswirtschaftlicher Wert. Es wird ein Ansatz präsentiert, um die quantitative Kausalität zwischen guter Softwarearchitektur (d. h. hoher betriebswirtschaftlicher Wert der Software) und schlechter Softwarearchitektur (d. h. niedriger betriebswirtschaftlicher Wert der Software) mittels eines Modells aufzuzeigen.

Paper V

Role-Based Runtime Model Synchronization

Christopher Werner, **Hendrik Schön**, Thomas Kühn, Sebastian Götz, Uwe Assmann [277]

Model-driven Software Development (MDSD) promotes the use of multiple related models to realize a software system systematically. These models usually contain redundant information but are independently edited. This easily leads to inconsistencies among them. To ensure consistency among multiple models, model synchronizations have to be employed, e.g., by means of model transformations, trace links, or triple graph grammars. Model synchronization poses three main problems for MDSD. First, classical model synchronization approaches have to be manually triggered to perform the synchronization. However, to support the consistent evolution of multiple models, it is necessary to immediately and continuously update all of them. Second, synchronization rules are specified at design time and, in classic approaches, cannot be extended at runtime, which is necessary if metamodels evolve at runtime. Finally, most classical synchronization approaches focus on bilateral model synchronization, i.e., the synchronization between two models. Consequently, for more than two models, they require the definition of pairwise model synchronizations leading to a combinatorial explosion of synchronization rules. To remedy these issues, we propose a role-based approach for runtime model synchronization. In particular, we propose role-based synchronization rules that enable the immediate and continuous propagation of changes to multiple interrelated models (and back again). Additionally, our approach permits adding new and customized synchronization rules at runtime. We illustrate the benefits of role-based runtime model synchronization using the Families to Persons case study from the Transformation Tool Contest 2017.

Paper VI

Contextual and Relational Role-Based Modeling Framework

Thomas Kühn, Christopher Werner, **Hendrik Schön**, Zhao Zhenxi, Uwe Assmann [152]

Model-driven Software Development (MDSD) approaches struggle when modeling context-dependent and dynamic systems, as their underlying metamodels cannot capture context-dependent concepts and relations. By contrast, role-based modeling has been studied for more than 35 years as a promising paradigm to model context-dependent and dynamic systems. Although some approaches have considered the application of roles on the metamodel level, no approach employed a contextual and relational role-based metamodel as the basis of a modeling framework. To remedy this, we employ the Compartment Role Object Model (CROM) which is a contextual and relational role-based modeling language, as the underlying metamodel of a novel Role-based Modeling Framework (RMF). In particular, our framework is able to generate inter-operable *Java* source code that permits the programmatic creation, manipulation, and persistence of role-based models. We illustrate the applicability of RMF by modeling a small system with context-dependent concepts and relations, generating corresponding *Java* source code, and employing it to load, manipulate, and store role-based models.

Paper VII

A Role-Based Maturity Model for Digital Relevance

Katja Bley, **Hendrik Schön** [30]

For several decades, maturity models have been regarded as a magic bullet for enterprises' economic growth processes. In these models, domains are structured and divided into (mostly linear) levels that are used as benchmarks for enterprise development. However, this approach has its shortcomings in the context of complex topics like digitalization. As we understand it, digitalization defines a conceptual approach to a phenomenon that is individually important and promising in different ways for different enterprises. Consequently, existing maturity models in their current form are not able to reproduce the full extent of digitalization for enterprises, as the models are too general—especially for SMEs of different sizes and from different sectors. In our research, we propose a maturity model approach that introduces the concept of roles as a possibility to depict enterprises' specific components, which are then added to a static core model. By doing so, the resulting maturity model is more flexible and scalable for SMEs' specific needs. Furthermore, we introduce a new assessment approach for defining whether improving digitalization is truly relevant and worthwhile for the enterprise.

Paper IIX

A Role-Based Capability Modeling Approach for Adaptive Information Systems

Hendrik Schön, Jelena Zdravkovic, Janis Stirna, Susanne Strahringer [224]

Most modeling approaches lack in their ability to cover a full-fledged view of a software system's business requirements, goals, and capabilities and to specify aspects of flexibility and variability. The modeling language Capability Driven Development (CDD) allows modeling capabilities and their relation to the execution context. However, its context-dependency lacks the possibility to define dynamic structural information that may be part of the context: persons, their roles, and the impact of objects that are involved in a particular execution occurrence. To solve this issue, we extended the CDD method with the BROS modeling approach, a role-based structural modeling language that allows the definition of context-dependent and dynamic structure of an information system. In this paper, we propose the integrated combination of the two modeling approaches by extending the CDD meta-model with necessary concepts from BROS. This combination allows for technical development of the information system (BROS) by starting with capability modeling using CDD. We demonstrate the combined meta-model in an example based on a real-world use case. With it, we show the benefits of modeling detailed business requirements regarding context comprising environment- and object-related information.

Paper IX

Overcoming the Ivory Tower: A Meta Model for Staged Maturity Models

Katja Bley, **Hendrik Schön**, Susanne Strahringer [31]

When it comes to the economic and strategic development of companies, maturity models are regarded as silver bullets. However, the existing discrepancy between the large amount of existing, differently developed models and their rare application remains astonishing. We focus on this phenomenon by analyzing the mod-els’ interpretability and possible structural and conceptual inconsistencies. By an-alyzing existing, staged maturity models, we develop a meta model for staged ma-turity models so different maturity models may share common semantics and syntax. Our meta model can therefore contribute to the conceptual rigor of exist-ing and future maturity models in all domains and can be decisive for the success or failure of a maturity measurement in a company.

Appendix B
Student Assistance

During the thesis' project lifetime, several works of students were mentored, advised, and reviewed in order to improve BROS. The three main works of students are contributing to the BROS methodology and are presented in this chapter as a summary.

B.1 BROS Editor Development

Development of a Web-based BROS Modeling Editor

Supervisors: **Hendrik Schön**, Thomas Kühn
Students: Lars Westermann, Sebastian Leichsenring
URL(s): `https://github.com/Eden-06/FRaMED-io` (repository, acc. 03.2020)
`https://eden-06.github.io/FRaMED-io` (web-editor, acc. 03.2020)

This project aims for the web-based reimplementation of the FRaMED 2.0 editor[1] for the family of role-based modeling languages (CROM). The goal was to develop a prototype of a web-based editor for BROS models. As the development was made agile, the requirements were created during the development phase. Before the editor was created, BROS models had to be created by hand. The new editor is able to create a BROS model according to the specification in this thesis. However, its focus is on a more technical layer: while the complete BROS language specification (in this thesis) is more extensive compared to the editor's functionality, the model constructs created with the editor are deeper integrated into the CROM specification and thus aim to be implemented in a programming language.

The developed editor, as an important artifact of the BROS landscape, is further described in Appendix C.2.

[1] `https://github.com/Eden-06/FRaMED-2.0`

B.2 Consistency of BROS and BPMN

Evaluating the Consistency between Business Process Models and Business Role-Object Specifications

Supervisors: **Hendrik Schön**, Thomas Kühn
Student: Lars Westermann

The student's work aims to develop a web-based tool to verify the consistency of a given BROS model towards a given BPMN model as a background process. For this purpose, the model elements of the BPMN and BROS models are compared, and consistency problems are discovered.

For that, the elements of the models are analyzed as instances of the respective metamodels of the language. In addition, six (exemplary) consistency rules are established, which are checked on the meta-analysis of the elements (e.g., *"Rule 2: For each BPMN swimlane there is a related BROS role."*). Warnings are then given to the modeler if, e.g., some required BPMN elements in the BROS model are missing.

The two models (BPMN and BROS) are given as input. The BROS model is modeled in *Framed.io* modeling editor (cf. Appendix C.2 and Appendix B.1) and BPMN uses the data structure provided by the *Bpmn.io* modeling editor[2].

The tool instantiates the metamodels of both languages as a graph-based structure. Afterward, the matching phase ("Matching" in Figure B.1) connects the different elements semantically with each other. The matching is not the consistency check but only connects belonging elements between both models via the "oracle function" $f(x)$, e.g., a BPMN swimlane starts with a BROS event[3]. Thus, both graphs are connected by semantic relationships. As a next step, the tool checks the consistency by applying the given consistency rules ("Verification" step in Figure B.1) on the built and connected graph. For that, the tool uses *Prolog* implication rules to describe

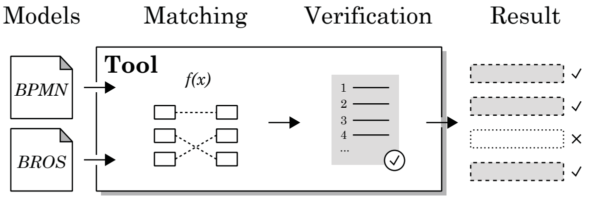
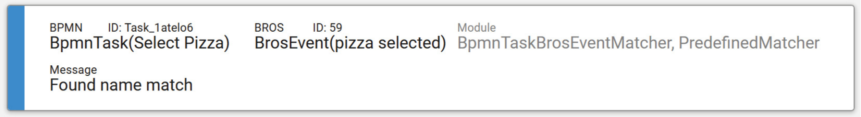


Fig. B.1 Representation of the process flow and business logic of the tool

[2] See `https://bpmn.io`; the use of this tool is only for demonstration purposes, and the processed input structure can be changed to other BPMN tools as well.

[3] This function "knows" which elements are connected with each other as this is needed. As for now, this is done via complex name comparison.



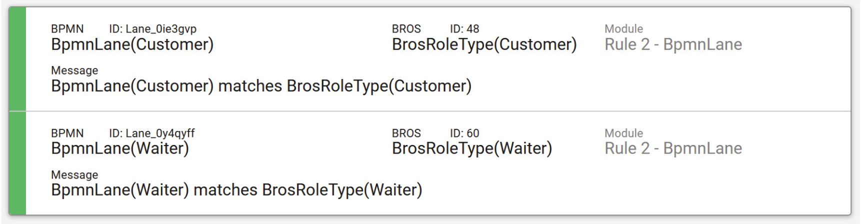
BPMN ID: Task_1atelo6
BpmnTask(Select Pizza)

BROS ID: 59
BrosEvent(pizza selected)

Module
BpmnTaskBrosEventMatcher, PredefinedMatcher

Message
Found name match

Fig. B.2 A successful matching of two elements



BPMN ID: Lane_0ie3gvp
BpmnLane(Customer)

BROS ID: 48
BrosRoleType(Customer)

Module
Rule 2 - BpmnLane

Message
BpmnLane(Customer) matches BrosRoleType(Customer)

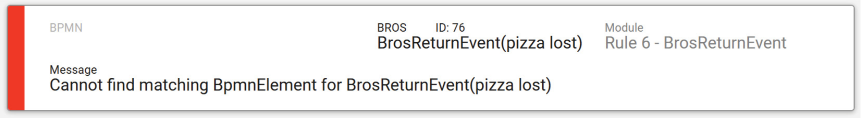
BPMN ID: Lane_0y4qyff
BpmnLane(Waiter)

BROS ID: 60
BrosRoleType(Waiter)

Module
Rule 2 - BpmnLane

Message
BpmnLane(Waiter) matches BrosRoleType(Waiter)

Fig. B.3 A successfully checked constraint



BPMN

BROS ID: 76
BrosReturnEvent(pizza lost)

Module
Rule 6 - BrosReturnEvent

Message
Cannot find matching BpmnElement for BrosReturnEvent(pizza lost)

Fig. B.4 A failed checked constraint

the given consistency rules and work on the meta-graph in a formal way, e.g., as described in Listing B.1 for rule 2.

```
rule_2 (Bpmn) :- bpmn(Bpmn,"Swimlane") ->
    (
        bros(Bros,"RoleType"),match(Bpmn,Bros)
    ).
```

Listing B.1 Prolog implementation of the example rule

The *Prolog* rules are either apply or not on the graph structures, thus, the consistency of the whole graph structures is checked against the requirements (the consistency rules). The set of rules can be extended as they are outsourced to individual "rule files" that contain the related *Prolog* code.

Finally, the user gets the result as a list of (a) the connected matchings, and (b) the checked results of the graphs ("Result" in Figure B.1). The former is for transparency reasons, the latter the result that the modeler is interested in. An example matching item is shown in Figure B.2, a positive rule check in Figure B.3 and a negative one in Figure B.4 (it says that "return event *pizza lost* is missing in BROS but modeled in the respective BPMN background process"). As a demonstrating use case for this work, the pizza order example from Schön et al. [223] was used. The BROS and BPMN models were modeled via the respective tools and checked for consistency. For that, errors were introduced into the BROS model, and then the tool was checked to see if it found all the inserted errors. Due to the extendable nature of the tool architecture, it is possible to make it an integral component of BROS developments.

B.3 Evaluation Methods for BROS

Evaluation of Modeling Languages and Artifacts

Supervisors: **Hendrik Schön**, Susanne Strahringer
Student: Sebastian Leichsenring

This student's work is aimed to pursue two issues: (a) to identify the possible evaluation methods for BROS, and (b) to evaluate the BROS language (and models) with it. Up to now, only the former part is completely done, and the latter part is currently under development. The evaluation of BROS is an essential part of assessing its applicability in practice as well as the improvement of the BROS language and adaptation method[4]. The use case proposed in Chapter 7 is some kind of evaluation but cannot replace a thorough evaluation using established methods.

To identify the possible methods of further BROS evaluation, the objective of this work was to develop a scientifically created evaluation method taxonomy within a morphological box. It was decided to apply the taxonomy development method by Nickerson et al. [183] to develop a meaningful taxonomy regarding several IS dimensions. As an intermediate step, a web-based tool (called *Taxonomy for Modeling Language Evaluation*, TMLE) was created that can be used to develop taxonomies based on the Nickerson et al. [183] approach[5]. The tool also represents the developed taxonomy as a morphological box with additional embedded information.

Additionally, to identify a taxonomy suitable for language evaluation methods, the TMLE tool was used with a configured set of parameters so that a resulting taxonomy for language evaluation methods can be used to select an appropriate evaluation method for BROS. For that, the following schema in Listing B.2 was calculated by the tool concerning modeling language evaluation approaches.

```
T(modeling evaluation approaches) = {
    origin domain (node–edge–modeling,IS [2]),
    key characteristics (ontological meta–model–comparison, ... [8]),
    key aspects (global consistency,goals,language ... [46]),
    quality characteristics and guidelines (functionality,reliability,
    efficiency, ... [52]),
    fundamentals (FRISCO report,ABC–method, ... [8]),
    subject of evaluation (modeling language,artifact [2])
}
```

Listing B.2 The resulting taxonomy for modeling language evaluation approaches

The first part of the work is completed by demonstrating a workflow of how to use the TMLE tool for evaluation method identification for BROS. This workflow also contains the appropriate selection of the taxonomy properties fitting to the requirements (e.g., a suitable evaluation approach should consider the "modeling

[4] The focus is, however, the BROS language (and not the adaptation method).

[5] This is, for now, domain-independent and can be used to develop taxonomies for different purposes.

language", and uses the properties "structure-oriented" and "process-oriented"). Until now, the calculated taxonomy (only) provides an overview of possible language evaluation methods based on the required properties for BROS. In the follow-up project, a method will be chosen and applied to evaluate BROS further.

Appendix C
Additional Topics

Some topics and subjects are not part of the main thesis' boundary, but were developed during the project and are still worth mentioning in order to provide inspiration, starting points, and help for further BROS development.

C.1 Preliminary Adaptation DSL

> The notation presented in this section is not suitable for operational modeling. These are excerpts from a preliminary textual DSL used for the development of BROS and are not guaranteed to be consistent or complete.

Element Notation

The *element notation set* describes a set $\mathbb{E}$ of *element notations* e, where each e is a statement that describes a part of the model (that is, entity or relationship).

$$\mathbb{E} = \{e_0, e_1 \ldots e_n\}$$

In sum, an ideal set $\mathbb{E}$ describes the complete model in its layout. To define the single e was not focused in this thesis, however, some of the internally used possibilities are listed in Table C.1 (and can possibly be of use when developing a complete formal DSL for BROS).

Adaptation Notation

The *adaptation notation set* describes a set $\mathbb{A}$ of *adaptation notations* a, where each a is a statement that describes a transition (atomic adaptation) of the application model

Table C.1 Possible notation of individual BROS elements

Element	Notation	Comment	Example
Entities			
Object	$\underline{O}$	underlined, capitalized	$\underline{\texttt{Car}}$
Role	R	capitalized	Camper
Scene	$X\{\{\ldots\}\}$	double curly brackets	Long-term_Rent{{...}}
Compartm.	$X\{\ldots\}$	single curly brackets	Store{...}
Event	e	lowercase	confirmed
Package	$\ldots^{P}$	package superscript for something	$\texttt{Camper}^{\texttt{RTC}}$
Port	$\#\ldots$	something prefixed with #	#Camper
Group	$G[\ldots]$	single square brackets	Car[...]
Token	$\langle t\rangle$ / $\langle\langle t\rangle\rangle$	single (use) or double (assign) angle brackets	⟨1⟩
Multiplicity	$\ldots^{[\ldots]}$	as superscript (also for any other entity property like repetition)	$\texttt{Camper}^{[0..1]}$
Relationships			
Fullfillm.	$-*->$	asterisk ($*$) in-between	Car -*-> Camper
Assoc.	$<-->$	line (possibly with direction)	Driver -- Camper
Implic.	$->-$	in-between arrow	VIP ->- Client
Prohib.	$-\lvert\lvert-$	in-between double lines	Employee -||- VIP
Eqival.	$-<>-$	in-between angle brackets	Tenant -<>- Driver
Create/Destr.	$\cdot\cdot>$	two dots with arrow	sign ..> Client
Cause	$\cdot\cdot$	two dots	Tenant .. sign
Aggreg.	$<>--$	line with angle brackets	Company <>-- Store
Composit.	$<<>>--$	line with double angle brackets	Car ≪≫-- Engine

state to another. Thus, the single a can be considered to be adaptation instructions (cf. Section 5.1.4).

$$\mathbb{A} = \{a_0, a_1 \ldots a_n\}$$

$$AM_{n+1} = AM_n \cap \mathbb{A}$$

Thus, the adaptation statement set describes basically the adaptation instruction set. However, the adaptation notation set is even more preliminary than the element notation set (as it was hardly used during the BROS development). Some examples of adaptation statements are listed in Table C.2. A chain of adaptation notations $a_0 \ldots a_n$ can then be used to describe the process of adaptation of a reference model (this can be compared with the *change patterns* introduced by Weber et al. [273]).

Table C.2 Possible notation of BROS adaptations

Adaptation	Notation	Comment	Examples

Instruction	$XYZ(\ldots)$	with brackets	ROL($\underline{\texttt{Car}}$ -*-> Camper)
			CON(Tenant ->- Driver)
			EVE(confirmed ··> Tenant)
			GRP(Car[Camper,Caravan])
			DYC(Long-term_Rent{{Lease$^{[1..1]}$,Camper}})

C.2 Tool “Framed.io”

In this section, the BROS modeling tool is introduced, an advancement of the standard editor for CROM. The *Framed.io* tool used to help with simple models and quick modeling tasks such that the models do not have to be modeled by hand. The project was done as a side project (cf. Section B.1) and developed in an agile way. It is a first approach and a prototype, but the current editor can already be used for simple tasks. Nevertheless, due to its early phase, it does not adhere to the modeling practice manual (and the style guide in particular) yet.

The project is hosted on *GitHub*[1] and the compiled editor (stable branch) can be accessed[2] via the URL

`https://eden-06.github.io/FRaMED-io`

Please note that the editor is in its first version and still in development. An introduction of how to use the editor is given in `https://github.com/Eden-06/FRaMED-io/wiki/User-interface`.

C.2.1 Software Architecture

Framed.io was realized as a web-based editor to be easily accessible. It is based on state-of-the-art architecture based on the *Eclipse* editor architecture of *Graphiti*[3] [38].

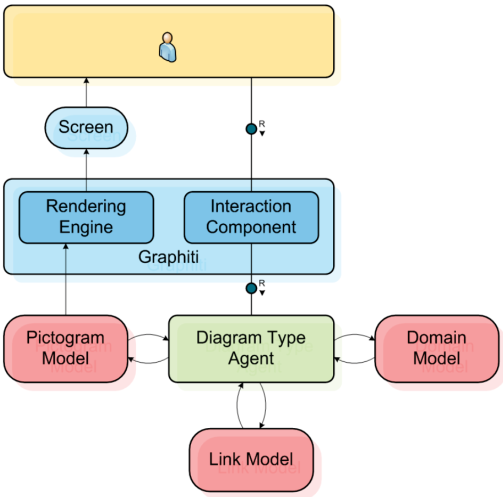


Fig. C.1 Brand et al. [38]: Graphiti modeling editor architecture

[1] `https://github.com/Eden-06/FRaMED-io`

[2] Last accessed: 20.03.2020

[3] `https://www.eclipse.org/graphiti/documentation/overview.php`

This architecture divides the editor structure into three main parts: a *Pictogram Model*, a *Link Model*, and a *Domain Model* as illustrated in Figure C.1 (from [38]).

The domain model is at the heart of the model representation. It contains the semantic information that is used to describe the domain.

> "The Domain Model contains the data which has to be visualized graphically. A developer would for example use the Ecore metamodel for a graphical Ecore editor. An editor for BPM (Business Process Management) would use the Businesses Process Modeling Notation." [38, p. 3]

In *Framed.io*, this is the semantic information of the BROS model. The pictogram model, in contrast, holds the information about the model itself, especially the arrangement of elements.

> "The Pictogram Model contains the complete information for representing a diagram. That implies that each diagram can be represented without the presence of the Domain Data. This requires a partially redundant storage of data, which is present both in the Pictogram Model and in the Domain Model. " [38, p. 3]

For *Framed.io* this is the same, and the pictogram model contains the layout data. Last, the link model combines both models in order to map user activities.

> "The Link Model is responsible for connecting data from the Domain Model and the graphical representation (that is,data from the Pictogram Model). These connections are again needed by many actions in the graphical editor. For instance, a deletion or a move of a graphical object needs also access to the associated object of the Domain Model in order to be able to make the necessary changes." [38, p. 3]

The benefit of this architecture is the support of different diagram types. Often, the domain model remains the same, while only its representation changes (e.g., the domain data of a car rental domain could be represented in a class diagram and a sequence diagram in UML). To exchange the representation, the domain model can remain in the editor's domain model representation, while only the pictogram model (and probably the linker model) need to be changed. For BROS, no other representations are expected, however, *Framed.io* could change the inner pictogram model and its linking model to support other diagram types with the same domain data. For switching between different diagram types, the *Diagram Type Agent* is used. Further, for interaction with the user, the agent can handle features like "create object" and, thus, support multiple models at the same time as all of them use the same domain

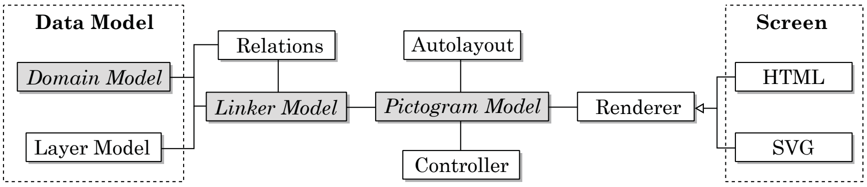
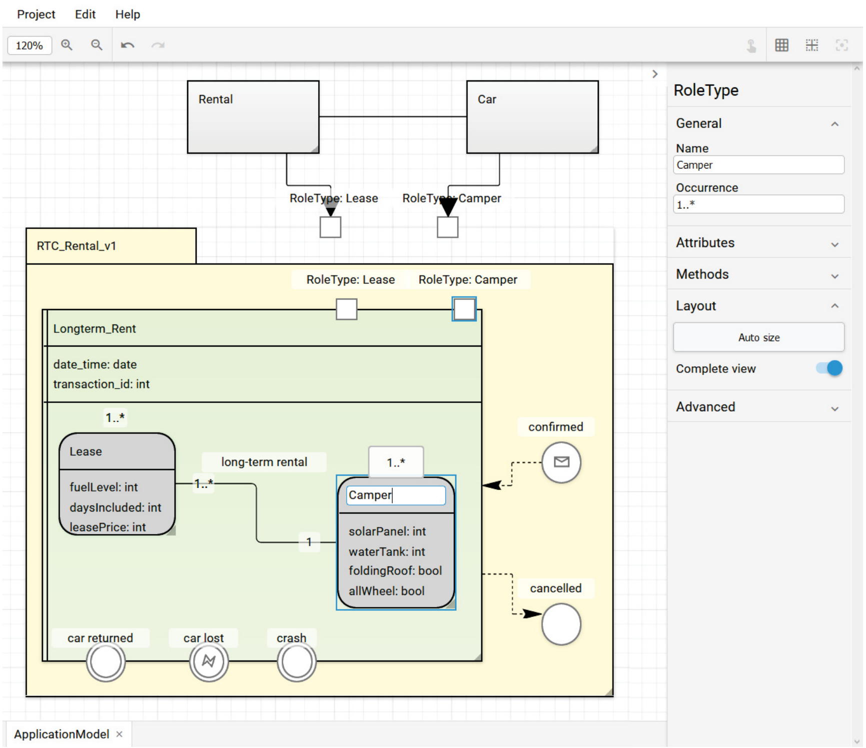


Fig. C.2 Framed.io architecture specification

model[4]. This architecture is the core of the final *Framed.io* architecture as well, illustrated in Figure C.2, and was extended with technical components. Regarding the technical aspect, the whole tool backend is written in *Kotlin*[5], a programming language using the *JavaScript* compiler[6]. Further, *Framed.io* makes use of different frameworks. E.g., the diagram layout (the node-relationship-representation) is currently rendered by *jsplumb*[7] and the auto-layout feature is supported by the *dagrejs*[8] framework.



Name: Camper

Occurrence: 1..*

Complete view: ●

Fig. C.3 Framed.io editor main window (example from Section 5.2.5)

[4] For further information about the individual *Graphiti* components please see Brand et al. [38].

[5] https://kotlinlang.org

[6] https://developer.mozilla.org/en-US/docs/Web/JavaScript

[7] https://jsplumbtoolkit.com

[8] https://github.com/dagrejs/dagre

C.2.2 Features

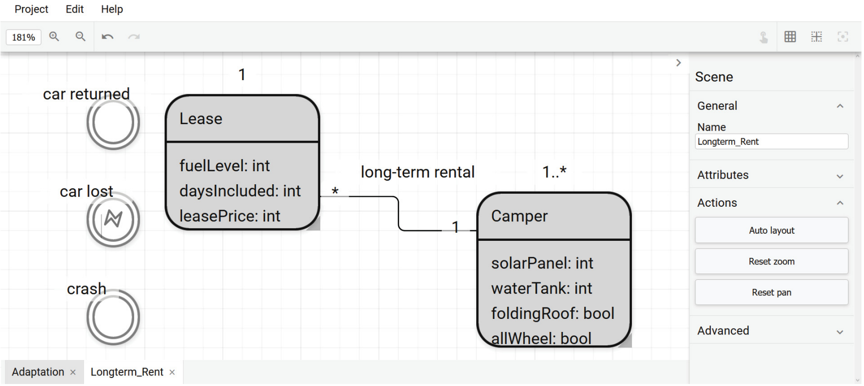
C.2.2.1 Creation and Adaptation

BROS models in *Framed.io* are modeled in the “what-you-see-is-what-you-get” way. Thus, the model can be used in any way to create the desired application model right in the editor. Figure C.3 illustrates the main window of *Framed.io*, where the example model from Section 5.2.5 is currently modeled.

Objects and their relationships represent reference models in BROS. There is no concrete functionality to “use a reference model,” but it has to be modeled in *Framed.io* (with objects and relationships) in the first place. The once modeled reference model can be loaded infinite times and adapted towards the own use case. For that, *Framed.io* supports most of the BROS concepts presented in Chapter 4. Especially the package concept is highlighted to represent different adaptations. Every element can be specified in detail, e.g., role’s fields and their types. Due to its early version, the *Framed.io* does not completely adhere to the modeling practice manual stated in Chapter 6, yet. The editor is not bound to adaptation but can also be used to model software from scratch.

The editor has some comfort features implemented, often assumed as standard when considering editor comfort features:

- *Step-in view* of any context: when stepping into a context, the perspective is seen as from inside and without any outer model elements (for example shown in Figure C.4 where step-in into `Longterm_Rent` was done); breadcrumbs at the bottom line indicate the user’s position
- *Flat view* of elements: the most element supports the flat view of itself that toggles the internal details between a full-fledged model and a textual listing (for example shown in Figure C.5 for a scene and Figure C.6 for a package); element pictograms indicate the type of the textual listing’s elements



Name: Longterm_Rent

Fig. C.4 Framed.io editor step-in feature (stepped into `Longterm_Rent` from Figure C.3)

- *Auto-size* for automatic adjustment of an element’s size regarding its inner elements
- *Auto-layout* for automatic distribution and placement of the selected element’s inner elements (or the whole model)
- General comfort features:
  - Metadata support for the model (e.g., author, creation date, modification date) to track adaptations
  - Step-wise back and forward navigation of taken actions
  - Automatic port generation when using relationships towards nested roles
  - Intelligent highlighting of roles and their ports (e.g., when hovering)
  - Automatic filtering of applicable elements (e.g., when modeling a fulfillment the editor only highlights the possible target roles)
  - Optional touch support
  - Continuous zoom
  - Right-click support for easy access to new elements and an element’s details
  - Shortcuts and multi-selection
  - Optional magnetic lines, grid, and element edges (for easier element placement)
  - Hide-able sidebar
  - Drag-and-drop of elements (e.g., into each other)

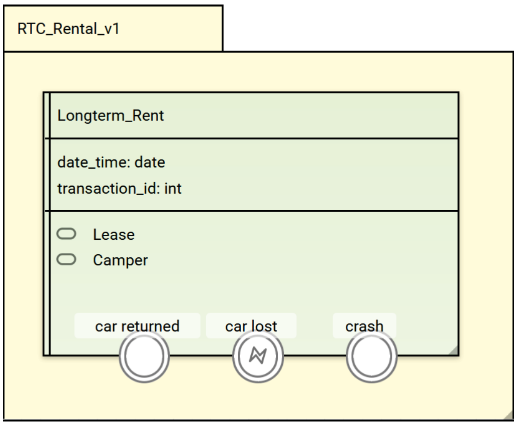


Fig. C.5 Framed.io editor flat view feature applied on a scene (flat view of `Longterm_Rent` from Figure C.3)

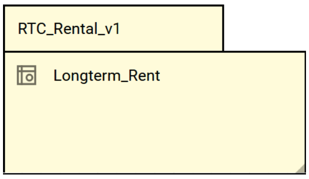


Fig. C.6 Framed.io editor flat view feature applied on a package (flat view of `RTC_Rental_v1` from Figure C.3)

C.2.2.2 Management, Storage and Sharing

The application model created with *Framed.io* can be saved and loaded via respective buttons on the main menu. When saving a model, it is stored as a JSON file and contains both the content and the layout. The following code listing is a short excerpt of such a file:

```
"root": ["io.framed.model.Package", {
    [...]
    "name": "RTC_Rental_v1",
    "children": [
        ["io.framed.model.Scene", {
            "id": 61,
            "name": "Longterm_Rent",
            [...]
            "children": [
                ["io.framed.model.RoleType", {
                    "id": 62,
                    "name": "Lease",
                    "occurrenceConstraint": "1..*",
                    [...]
"connections": {
    "connections": [
        ["io.framed.model.Relationship", {
            "id": 82,
            "sourceId": 62,
            "targetId": 63,
            "name": "long-term rental",
            "sourceCardinality": "1..*",
            "targetCardinality": "1"
        }
        [...]
"layer": {
    "56": {
        "data": {
            [...]
            "62": {
                "left": 16,
                "top": 32,
                "width": 227,
                "height": 132,
                "autosize": false,
                "data": {
                },
                "labels": [
                    [...]
                    "position": 0.5,
                    "id": "name"
                ]
            },
```

Listing C.1 The BROS model file contents of the stored model

This file can, if only objects and their relationships are used, be stored as a reference model and loaded again whenever an adaptation is needed. There is (for now) no repository that can handle reference models and application models automatically as this is future work. The model, saved as JSON file, can be loaded again into *Framed.io* to continue with the work. The file can be distributed and loaded at any time in any instance of *Framed.io*.

Further, this generated file can be used for the BPMN-BROS verifier (see Section B.2) to validate the modeled BROS model in compliance with a BPMN model that served as the background process.

C.2.2.3 Validation and Constraint Checking

The *Framed.io* tool makes simple checks on the syntax of the BROS language when modeling a BROS model. The tool does not check the model regarding its compliance with the official BROS verification (cf. Section 6.2.1) and validation (cf. Section 6.2.2) rules. However, it checks smaller issues that are constrained by the BROS language element's definition (e.g., a fulfillment can only end at roles, events can only create roles and scenes, return events are always the edge of a scene). If a modeling error occurs, *Framed.io* can handle in two different ways:

1. it can prevent the wrong placement or specification by simply not allowing it, or
2. allowing the wrong placement or specification but highlights in as an error (e.g., wrong naming).

Since *Framed.io* is focused on the technical consistency rather than supporting the complete BROS modeling palette, the *Framed.io* BROS models are more restricted and slightly different than the BROS specification written in this thesis. However, the constraints are introduced to prevent inconsistent model statements and too complex models. A prominent example is the *Framed.io* prevention of using spaces in a model element's name. Although spaces are allowed in the BROS specification, the editor marks this with an error frame as it is not common to use spaces in names as they often lead to problems considering technical feasibility and implementation.

In general, the *Framed.io* editor makes sure that quickly drawn BROS models are consistent. Nevertheless, for more complex applications of BROS, manual verification and validation are inevitable.

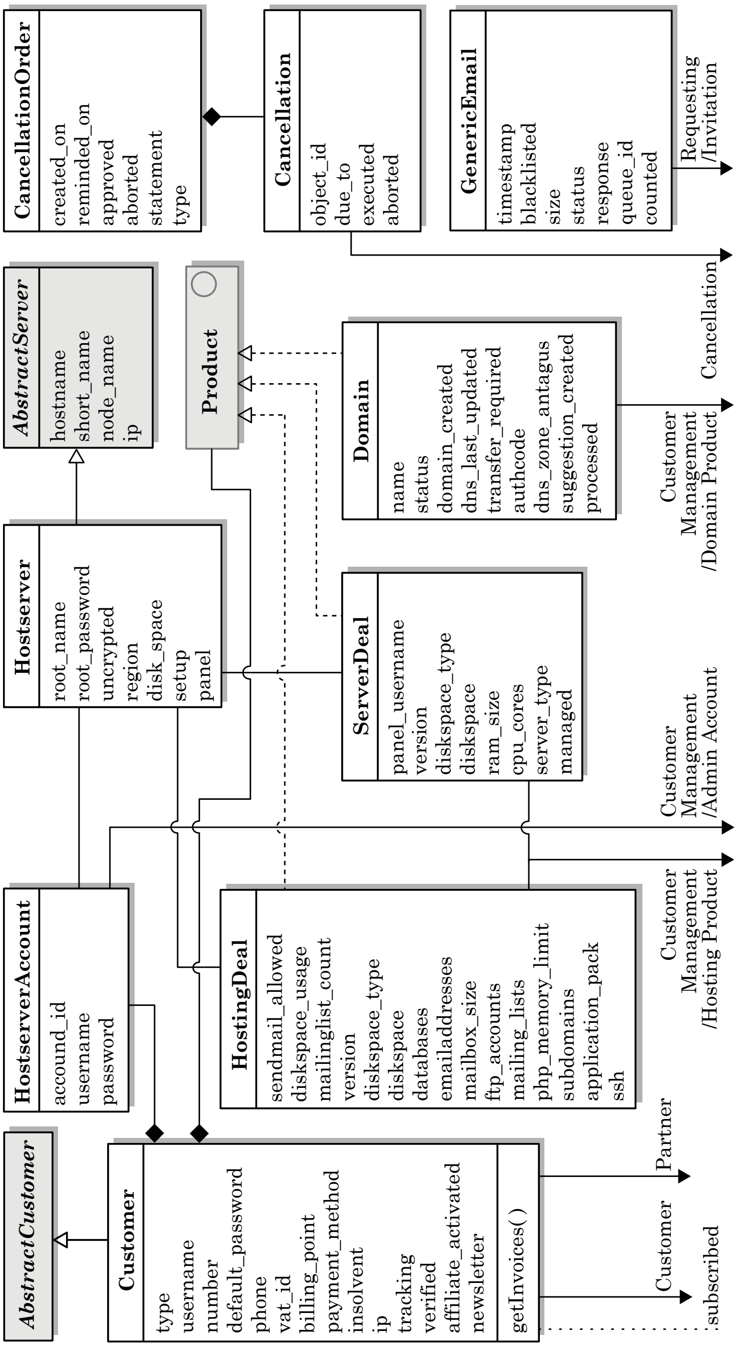
C.2.3 Future Development

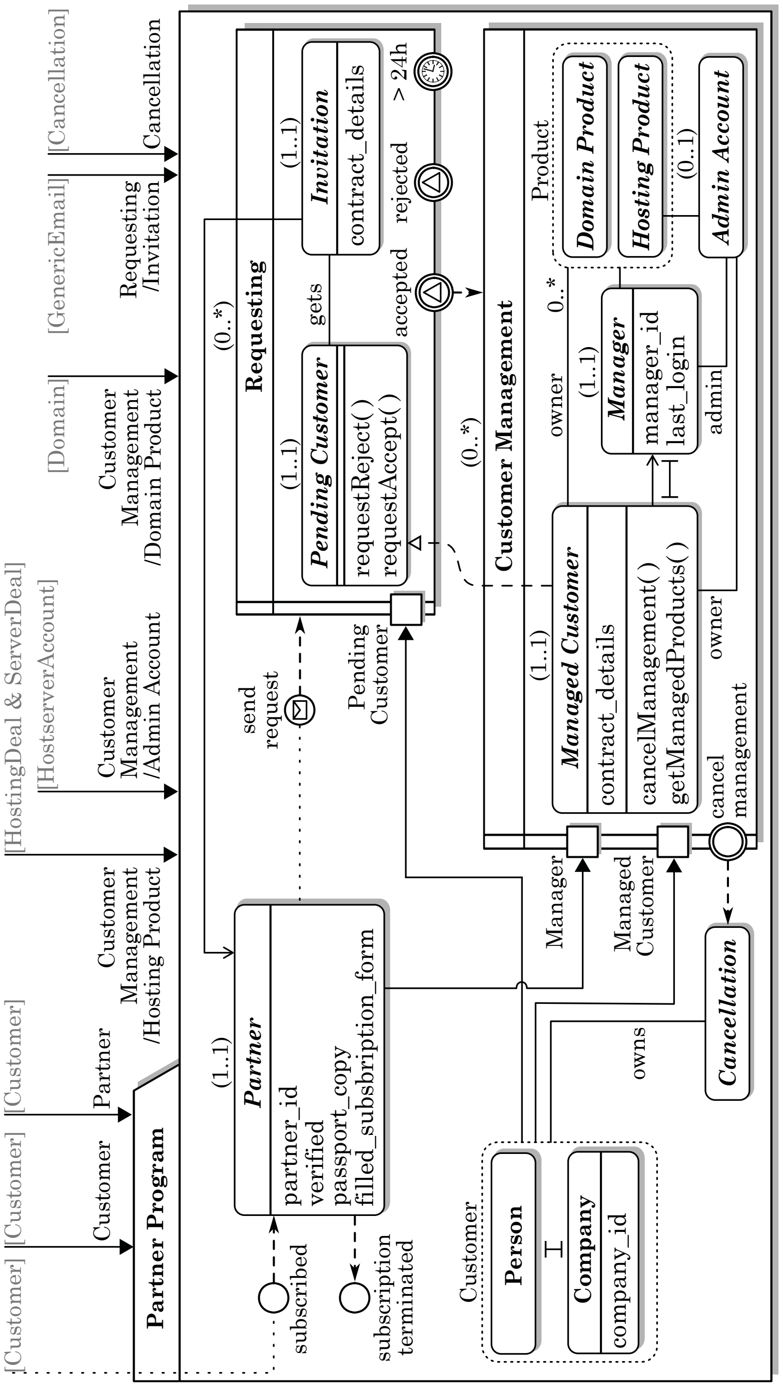
The *Framed.io* editor is currently in its first stable phase and can consistently model BROS models. It can be used to quickly create BROS models (from scratch or as adaptation) in a useful manner.

Nevertheless, as *Framed.io* is in its first operational version, the BROS editor has some issues in need of improvement:

- The BROS language specification supported by the editor needs to be extended in order to be compliant with the “standard” BROS specification; current restrictions need to be solved in accordance with the requirements.
- The style guide, validation, and verification of BROS (cf. Chapter 6) need to be supported. Especially the style guide would improve the editor in many ways (e.g., crossing line arcs, text formatting).
- The “reference model and its adaptation” use case could be more prominent, e.g., by supporting the creation of reference models with certain limitations and tags. In contrast, features regarding the adaptation can be useful too (e.g., view layers that hide the adaptation parts of the current reference model adaptation).
- More configurable options and settings could improve and further individualize the BROS model (e.g., colors, shapes, fonts, shadows).
- Although *Framed.io* already has optional touch support, the general support for mobile devices and the overall usability of the editor could be improved.
- Support for including BROS patterns would simplify their usage.
- The technical implementation can be improved with further features (e.g., movable relationship anchor points, label sizes).
- Regarding its technical implementation, several bugs need to be fixed.

C.3 Large Use Case Application Model




[Customer]
[Customer]
Customer
[Customer]
Partner
[HostingDeal & ServerDeal]
Customer Management /Hosting Product
[HostserverAccount]
Customer Management /Admin Account
[Domain]
Customer Management /Domain Product
[GenericEmail]
Requesting /Invitation
[Cancellation]
Cancellation
Partner Program
(1..1)
Partner
partner_id
verified
passport_copy
filled_subsbription_form
subscribed
subscription terminated
send request
Pending Customer
(0..*)
Requesting
(1..1)
Pending Customer
requestReject( )
requestAccept( )
gets
(1..1)
Invitation
contract_details
accepted
rejected
> 24h
(0..*)
Customer Management
(1..1)
Managed Customer
contract_details
cancelManagement( )
getManagedProducts( )
owner
(1..1)
Manager
manager_id
last_login
0..*
owner
admin
Product
Domain Product
Hosting Product
(0..1)
Admin Account
Manager
Managed Customer
cancel management
Cancellation
owns
Customer
Person
Company
company_id

References

1. Abelló, A., Samos, J., and Saltor, F. YAM2: A multidimensional conceptual model extending UML. *Information Systems 31*, 6 (Sept. 2006), 541–567.
2. Aguilar-Savén, R. S. Business process modelling: Review and framework. *International Journal of Production Economics 90*, 2 (July 2004), 129–149.
3. Ahlemann, F. Towards a conceptual reference model for project management information systems. *International Journal of Project Management 27* (2009), 19–30.
4. Albakour, M.-D. Adaptive domain modelling for information retrieval. *ACM SIGIR Forum 47*, 1 (June 2012), 59.
5. Alexander, C. *A Pattern Language: Towns, Buildings, Construction*. Oxford University Press, Aug. 1977.
6. Allen, J. F. Maintaining Knowledge about Temporal Intervals. In *Readings in Qualitative Reasoning About Physical Systems*. Elsevier, 1990, pp. 361–372.
7. Allweyer, T. *BPMN 2.0: Introduction to the Standard for Business Process Modeling*, second ed. BoD – Books on Demand, May 2016.
8. Almeida, J. P. A., Costa, P. D., and Guizzardi, G. Towards an Ontology of Scenes and Situations. In *2018 IEEE Conference on Cognitive and Computational Aspects of Situation Management (CogSIMA)* (June 2018), pp. 29–35.
9. Almeida, J. P. A., and Guizzardi, G. On the foundation for roles in RM-ODP: Contributions from conceptual modelling. In *2007 Eleventh International IEEE EDOC Conference Workshop* (2007), vol. 1, IEEE, pp. 205–215.
10. Almeida, J. P. A., Guizzardi, G., and Santos, P. S. J. Applying and extending a semantic foundation for role-related concepts in enterprise modelling. In *Proceedings of the 12th IEEE International Enterprise Distributed Object Computing Conference, EDOC* (2009), IEEE, pp. 31–40.
11. Ambler, S. W. *The Elements of UML 2.0 Style*. Cambridge University Press, May 2005.
12. Atkinson, C., and Kuhne, T. Model-driven development: A metamodeling foundation. *IEEE Software 20*, 5 (Sept. 2003), 36–41.
13. Averill, E. Reference models and standards. *StandardView 2*, 2 (1994), 96–109.
14. Awadid, A., and Nurcan, S. Consistency requirements in business process modeling: A thorough overview. *Software & Systems Modeling* (Nov. 2017), 1–19.
15. Baader, F., Borgwardt, S., Koopmann, P., Ozaki, A., and Thost, V. Metric Temporal Description Logics with Interval-Rigid Names. In *Frontiers of Combining Systems* (Cham, Sept. 2017), C. Dixon and M. Finger, Eds., vol. 10483 of *LNAI (LNCS)*, Springer International Publishing, pp. 60–76.
16. Bachman, C. W., and Daya, M. The Role Concept in Data Models. In *Proceedings of the 3rd International Conference on Very Large Data Bases, VLDB* (Tokyo, Japan, 1977), vol. 3, ACM, pp. 464–476.

17. Balci, O., Arthur, J. D., and Nance, R. E. Accomplishing reuse with a simulation conceptual model. In *2008 Winter Simulation Conference* (Dec. 2008), pp. 959–965.
18. Baldoni, M., Boella, G., and van der Torre, L. Bridging Agent Theory and Object Orientation: Agent-Like Communication Among Objects. In *Programming Multi-Agent Systems* (Berlin, Heidelberg, 2007), R. H. Bordini, M. Dastani, J. Dix, and A. E. F. Seghrouchni, Eds., vol. 4411 of *LNCS*, Springer, pp. 149–164.
19. Balko, S., Ter Hofstede, A. H., Barros, A., La Rosa, M., and Adams, M. Business Process Extensibility. *Enterprise Modelling and Information Systems Architectures 5*, 3 (Dec. 2015).
20. Bass, L., Clements, P., and Kazman, R. *Software Architecture in Practice*, second ed. SEI Series in Software Engineering. Addison-Wesley, Boston, 2003.
21. Beck, K., Crocker, R., Meszaros, G., Coplien, J., Dominick, L., Paulisch, F., and Vlissides, J. Industrial experience with design patterns. In *Proceedings of IEEE 18th International Conference on Software Engineering* (Mar. 1996), pp. 103–114.
22. Beck, K., and Cunningham, W. A laboratory for teaching object oriented thinking. *ACM SIGPLAN Notices 24*, 10 (Oct. 1989), 1–6.
23. Becker, J., Delfmann, P., Dreiling, A., Knackstedt, R., and Kuropka, D. Configurative Process Modeling - Outlining an Approach to Increased Business Process Model Usability. In *Proceedings of the 15th Information Resources Management Association International Conference, IRMA* (2004), Gabler New Orleans, pp. 1–12.
24. Becker, J., Delfmann, P., and Knackstedt, R. Adaptive Reference Modeling: Integrating Configurative and Generic Adaptation Techniques for Information Models. In *Reference Modeling: Efficient Information Systems Design Through Reuse of Information Models*, J. Becker and P. Delfmann, Eds. Physica, 2007, pp. 27–58.
25. Becker, J., Knackstedt, R., Janiesch, C., and Pfeiffer, D. Configurative Method Engineering - On the Applicability of Reference Modeling Mechanisms in Method Engineering. In *Proceedings of the 13th Americas Conference on Information Systems, AMCIS* (2007), pp. 1–12.
26. Becker, J., Rosemann, M., and von Uthmann, C. Guidelines of Business Process Modeling. In *Business Process Management: Models, Techniques, and Empirical Studies*, W. van der Aalst, J. Desel, and A. Oberweis, Eds., vol. 1806 of *LNCS*. Springer, Berlin, Heidelberg, 2000, pp. 30–49.
27. Bera, P., Burton-Jones, A., and Wand, Y. Improving the representation of roles in conceptual modeling: Theory, method, and evidence. *Requirements Engineering* (June 2017).
28. Biffl, S., Aurum, A., Boehm, B., Erdogmus, H., and Grünbacher, P. *Value-Based Software Engineering*. Springer Science & Business Media, Feb. 2006.
29. Blanc, X., Mougenot, A., Mounier, I., and Mens, T. Incremental Detection of Model Inconsistencies Based on Model Operations. In *Advanced Information Systems Engineering* (Berlin, Heidelberg, 2009), P. van Eck, J. Gordijn, and R. Wieringa, Eds., vol. 5565 of *LNCS*, Springer, pp. 32–46.
30. Bley, K., and Schön, H. A Role-Based Maturity Model for Digital Relevance. In *Digital Transformation for a Sustainable Society in the 21st Century* (Cham, 2019), I. O. Pappas, P. Mikalef, Y. K. Dwivedi, L. Jaccheri, J. Krogstie, and M. Mäntymäki, Eds., vol. 11701 of *LNCS*, Springer International Publishing, pp. 738–744.
31. Bley, K., Schön, H., and Strahringer, S. Overcoming the Ivory Tower: A Meta Model for Staged Maturity Models. In *Responsible Design, Implementation and Use of Information and Communication Technology* (Cham, 2020), M. Hattingh, M. Matthee, H. Smuts, I. Pappas, Y. K. Dwivedi, and M. Mäntymäki, Eds., vol. 12066 of *LNCS*, Springer International Publishing, pp. 337–349.
32. Boehm, B. Value-based software engineering. *ACM SIGSOFT Software Engineering Notes 28*, 2 (Jan. 2003), 4.
33. Boella, G., and Steimann, F. Roles and Relationships in Object-Oriented Programming, Multiagent Systems and Ontologies. In *Object-Oriented Technology - ECOOP 2007 Workshop Reader* (Berlin, Heidelberg, July 2007), M. Cebulla, Ed., vol. 4906 of *LNCS*, Springer, pp. 108–122.

34. Boella, G., and van der Torre, L. A Foundational Ontology of Organizations and Roles. In *Declarative Agent Languages and Technologies IV* (Berlin, Heidelberg, 2006), M. Baldoni and U. Endriss, Eds., vol. 4327 of *LNAI (LNCS)*, Springer, pp. 78–88.
35. Böhm, M., Koleva, G., Leimeister, S., Riedl, C., and Krcmar, H. Towards a Generic Value Network for Cloud Computing. In *Economics of Grids, Clouds, Systems, and Services* (Berlin, Heidelberg, 2010), J. Altmann and O. F. Rana, Eds., vol. 6296 of *LNCS*, Springer, pp. 129–140.
36. Böhme, S., and Lippmann, M. Decidable Description Logics of Context with Rigid Roles. In *Frontiers of Combining Systems* (Cham, 2015), C. Lutz and S. Ranise, Eds., vol. 9322 of *LNAI (LNCS)*, Springer International Publishing, pp. 17–32.
37. Brambilla, M., Cabot, J., and Wimmer, M. Model-Driven Software Engineering in Practice. *Synthesis Lectures on Software Engineering 1*, 1 (Sept. 2012), 1–182.
38. Brand, C., Gorning, M., Kaiser, T., Pasch, J., and Wenz, M. Development of high-quality graphical model editors. *Eclipse Magazine* (2011).
39. Breu, R., Grosu, R., Huber, E., Rumpe, B., and Schwerin, W. Systems, Views and Models of UML. In *The Unified Modeling Language* (1998), M. Schader and A. Korthaus, Eds., Physica, pp. 93–108.
40. Burger, E., Henss, J., Küster, M., Kruse, S., and Happe, L. View-based model-driven software development with ModelJoin. *Software & Systems Modeling 15*, 2 (May 2016), 473–496.
41. Butakov, N., Petrov, M., Mukhina, K., Nasonov, D., and Kovalchuk, S. Unified domain-specific language for collecting and processing data of social media. *Journal of Intelligent Information Systems 51*, 2 (Oct. 2018), 389–414.
42. Cabot, J., Olivé, A., and Teniente, E. Representing Temporal Information in UML. In *«UML» 2003 - The Unified Modeling Language. Modeling Languages and Applications* (Berlin, Heidelberg, Oct. 2003), P. Stevens, J. Whittle, and G. Booch, Eds., vol. 2863 of *LNCS*, Springer, pp. 44–59.
43. Caetano, A., Silva, A., and Tribolet, J. Object-oriented business process modeling with roles. In *Proceedings of the 7th International Conference on Information Systems Implementation Modeling* (2004).
44. Calì, A., Calvanese, D., De Giacomo, G., and Lenzerini, M. A Formal Framework for Reasoning on UML Class Diagrams. In *Foundations of Intelligent Systems* (Berlin, Heidelberg, 2002), M.-S. Hacid, Z. W. Raś, D. A. Zighed, and Y. Kodratoff, Eds., vol. 2366 of *LNAI (LNCS)*, Springer, pp. 503–513.
45. Cazzola, W., Ghoneim, A., and Saake, G. Software Evolution through Dynamic Adaptation of Its OO Design. In *Objects, Agents, and Features: International Seminar, Revised and Invited Papers* (Berlin, Heidelberg, July 2004), M. Ryan, J. Meyer, and H. Ehrich, Eds., vol. 2975 of *LNCS*, Springer, pp. 67–80.
46. Ceylan, İ. İ., and Peñaloza, R. Probabilistic Query Answering in the Bayesian Description Logic BEL*. In *Scalable Uncertainty Management* (Cham, 2015), C. Beierle and A. Dekhtyar, Eds., vol. 9310 of *LNAI (LNCS)*, Springer International Publishing, pp. 21–35.
47. Chen, P. P.-s. The entity-relationship model: Toward a unified View of data. *ACM Transactions on Database Systems* (1976).
48. Chinosi, M., and Trombetta, A. BPMN: An introduction to the standard. *Computer Standards & Interfaces 34*, 1 (Jan. 2012), 124–134.
49. Chrszon, P., Dubslaff, C., Baier, C., Klein, J., and Klüppelholz, S. Modeling Role-Based Systems with Exogenous Coordination. In *Theory and Practice of Formal Methods: Essays Dedicated to Frank de Boer on the Occasion of His 60th Birthday*, E. Ábrahám, M. Bonsangue, and E. B. Johnsen, Eds., vol. 9660 of *LNCS*. Springer International Publishing, Cham, 2016, pp. 122–139.
50. Cicchetti, A., Ciccozzi, F., and Pierantonio, A. Multi-view approaches for software and system modelling: A systematic literature review. *Software & Systems Modeling* (Feb. 2019).
51. Cicchetti, A., Di Ruscio, D., and Pierantonio, A. Managing Dependent Changes in Coupled Evolution. In *Theory and Practice of Model Transformations* (Berlin, Heidelberg, 2009), R. F. Paige, Ed., vol. 5563 of *LNCS*, Springer, pp. 35–51.

52. Clarke, D., Helvensteijn, M., and Schaefer, I. Abstract delta modeling. *ACM SIGPLAN Notices 46*, 2 (Jan. 2011), 13.
53. Cockburn, A., and Highsmith, J. Agile software development, the people factor. *Computer 34*, 11 (Nov. 2001), 131–133.
54. Colman, A. *Role Oriented Adaptive Design*. Dissertation, Swinburne University of Technology, Melbourde, Australia, Oct. 2006.
55. Colman, A., and Han, J. Using role-based coordination to achieve software adaptability. *Science of Computer Programming 64*, 2 (2007), 223–245.
56. Colman, A. W., and Han, J. Organizational Roles and Players. In *Roles, an Interdisciplinary Perspective: Ontologies, Programming Languages, and Multiagent Systems: Papers from the AAAI Fall Symposium* (Arlington, VA, 2005), pp. 55–62.
57. Curtis, B., Kellner, M. I., and Over, J. Process modeling. *Communications of the ACM 35*, 9 (Sept. 1992), 75–90.
58. Dahanayake, A., and Thalheim, B. Co-evolution of (Information) System Models. In *Enterprise, Business-Process and Information Systems Modeling* (Berlin, Heidelberg, 2010), I. Bider, T. Halpin, J. Krogstie, S. Nurcan, E. Proper, R. Schmidt, and R. Ukor, Eds., vol. 50 of *LNBIP*, Springer, pp. 314–326.
59. Dahchour, M., Pirotte, A., and Zimányi, E. A Generic Role Model for Dynamic Objects. In *Advanced Information Systems Engineering* (Berlin, Heidelberg, 2002), A. B. Pidduck, M. T. Ozsu, J. Mylopoulos, and C. C. Woo, Eds., vol. 2348 of *LNCS*, Springer, pp. 643–658.
60. Dávid, I., Ráth, I., and Varró, D. Foundations for Streaming Model Transformations by Complex Event Processing. *Software & Systems Modeling 17*, 1 (Feb. 2018), 135–162.
61. de Cesare, S., Henderson-Sellers, B., Partridge, C., and Lycett, M. Improving Model Quality Through Foundational Ontologies: Two Contrasting Approaches to the Representation of Roles. In *Advances in Conceptual Modeling* (Cham, Oct. 2015), M. A. Jeusfeld and K. Karlapalem, Eds., vol. 9382 of *LNCS*, Springer, pp. 304–314.
62. De Irigon, J. I. Adaptive Routing in Disruption Tolerant Networks. In *2019 IEEE 20th International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM)* (June 2019), pp. 1–3.
63. de Souza, S. C. B., Anquetil, N., and de Oliveira, K. M. A study of the documentation essential to software maintenance. In *Proceedings of the 23rd Annual International Conference on Design of Communication: Documenting & Designing for Pervasive Information* (Coventry, United Kingdom, Sept. 2005), SIGDOC '05, Association for Computing Machinery, pp. 68–75.
64. Dey, A. K. Understanding and Using Context. *Personal Ubiquitous Comput. 5*, 1 (Jan. 2001), 4–7.
65. Dick, J., Hull, E., and Jackson, K. *Requirements Engineering*, fourth ed. Springer International Publishing, 2017.
66. Diepenbrock, A., Rademacher, F., and Sachweh, S. An Ontology-based Approach for Domain-driven Design of Microservice Architectures. In *Informatik 2017* (Bonn, Germany, 2017), M. Eibl and M. Gaedke, Eds., vol. P-275 of *Lecture Notes in Informatics - Proceedings*, Gesellschaft für Informatik e.V., pp. 1777–1791.
67. Dittrich, Y. What does it mean to use a method? Towards a practice theory for software engineering. *Information and Software Technology 70* (Feb. 2016), 220–231.
68. Döhring, M., Reijers, H. A., and Smirnov, S. Configuration vs. adaptation for business process variant maintenance: An empirical study. *Information Systems 39*, 1 (Jan. 2014), 108–133.
69. Döhring, M., and Zimmermann, B. vBPMN: Event-Aware Workflow Variants by Weaving BPMN2 and Business Rules. In *Enterprise, Business-Process and Information Systems Modeling* (Berlin, Heidelberg, 2011), T. Halpin, S. Nurcan, J. Krogstie, P. Soffer, E. Proper, R. Schmidt, and I. Bider, Eds., vol. 81 of *LNBIP*, Springer, pp. 332–341.
70. Doyle, K. G., Wood, J. R. G., and Wood-Harper, A. T. Soft systems and systems engineering: On the use of conceptual models in information system development. *Information Systems Journal 3*, 3 (1993), 187–198.

71. Dreiling, A., Rosemann, M., van der Aalst, W. M. P., Sadiq, W., and Khan, S. Model-Driven Process Configuration of Enterprise Systems. In *Proceedings of the 2005 International Conference on Wirtschaftsinformatik* (2005), O. K. Ferstl, E. J. Sinz, S. Eckert, and T. Isselhorst, Eds., Physica Heidelberg, pp. 687–706.
72. Duclos, L. K., Vokurka, R. J., and Lummus, R. R. A conceptual model of supply chain flexibility. *Industrial Management & Data Systems 103*, 6 (Jan. 2003), 446–456.
73. Edelweiss, N., de Oliveira, J. P. M., and Pernici, B. An object-oriented temporal model. In *Advanced Information Systems Engineering* (Berlin, Heidelberg, June 1993), C. Rolland, F. Bodart, and C. Cauvet, Eds., vol. 685 of *LNCS*, Springer, pp. 397–415.
74. Egyed, A., Letier, E., and Finkelstein, A. Generating and Evaluating Choices for Fixing Inconsistencies in UML Design Models. In *2008 23rd IEEE/ACM International Conference on Automated Software Engineering* (Sept. 2008), pp. 99–108.
75. Evans, E. *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Addison-Wesley Professional, 2004.
76. Fettke, P., and Loos, P. Classification of reference models: A methodology and its application. *Information Systems and e-Business Management 1*, 1 (Jan. 2003), 35–53.
77. Fettke, P., and Loos, P. Ontological Evaluation of Reference Models using the Bunge-Wand-Weber Model. In *Proceedings of the 9th Americas Conference on Information Systems, AMCIS* (2003), pp. 2944–2955.
78. Fettke, P., and Loos, P. Using Reference Models for Business Engineering - State-of-the-Art and Future Developments. In *Innovations in Information Technology* (2006), pp. 1–5.
79. Fettke, P., Loos, P., and Zwicker, J. Business Process Reference Models: Survey and Classification. In *Business Process Management Workshops* (Berlin, Heidelberg, Sept. 2005), C. Bussler and A. Haller, Eds., vol. 3812 of *LNCS*, Springer, pp. 469–483.
80. Fettke, P., Loos, P., and Zwicker, J. Using UML for Reference Modeling. In *Enterprise Modeling and Computing with UML*, P. Rittgen, Ed. Idea Group Inc (IGI), 2006, pp. 174–205.
81. Forward, A., and Lethbridge, T. C. The relevance of software documentation, tools and technologies: A survey. In *Proceedings of the 2002 ACM Symposium on Document Engineering* (McLean, Virginia, USA, Nov. 2002), DocEng '02, Association for Computing Machinery, pp. 26–33.
82. Fowler, M. *Patterns of Enterprise Application Architecture*, first ed. Addison Wesley, Boston, Nov. 2002.
83. Fowler, M. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. Addison-Wesley Professional, Sept. 2003.
84. Fox, M., Barbuceanu, M., and Gruninger, M. An organisation ontology for enterprise modelling: Preliminary concepts for linking structure and behaviour. In *Proceedings 4th IEEE Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE '95)* (Apr. 1995), pp. 71–81.
85. Frank, U. Delegation: An important concept for the appropriate design of object models. *Journal of Object Oriented Programming 13*, 3 (2000), 13–17.
86. Frank, U. Enterprise Modelling: The Next Steps. *Enterprise Modelling and Information Systems Architectures (EMISAJ) 9*, 1 (2014), 22–37.
87. Frank, U. Multi-perspective enterprise modeling: Foundational concepts, prospects and future research challenges. *Software & Systems Modeling 13*, 3 (July 2014), 941–962.
88. Frank, U., Heise, D., Kattenstroth, H., Ferguson, D. F., Hadar, E., and Waschke, M. G. ITML: A domain-specific modeling language for supporting business driven it management. In *Proceedings of the 9th OOPSLA Workshop on Domain-Specific Modeling, DSM* (2009).
89. Frank, U., Strecker, S., Fettke, P., vom Brocke, J., Becker, J., and Sinz, E. The Research Field "Modeling Business Information Systems". *Business & Information Systems Engineering 6*, 1 (Feb. 2014), 39–43.
90. Furrer, F. J. *Future-Proof Software-Systems*. Springer Fachmedien, Wiesbaden, 2019.
91. Galton, A. States, Processes and Events, and the Ontology of Causal Relations. Technical Report, College of Engineering, Mathematics and Physical Sciences, University of Exeter, Exeter, UK, 2012.

92. Gamma, E., Helm, R., Johnson, R., and Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software*. Pearson Education, Oct. 1994.
93. Gerosa, M., and Taisch, M. A Logistic Service Provider Reference Model. In *Proceedings of the 13th Symposium on Information Control Problems in Manufacturing, IFAC* (Moscow, Russia, 2009), vol. 42, IFAC, pp. 1340–1345.
94. Giese, M., and Heldal, R. From Informal to Formal Specifications in UML. In *«UML» 2004 — The Unified Modeling Language. Modeling Languages and Applications* (Berlin, Heidelberg, 2004), T. Baar, A. Strohmeier, A. Moreira, and S. J. Mellor, Eds., vol. 3273 of *LNCS*, Springer, pp. 197–211.
95. Goldstein, A., Johanndeiter, T., and Frank, U. Business process runtime models: Towards bridging the gap between design, enactment, and evaluation of business processes. *Information Systems and e-Business Management 17*, 1 (Mar. 2019), 27–64.
96. Gonzalez-Perez, C., and Henderson-Sellers, B. *Metamodelling for Software Engineering*. Wiley Publishing, 2008.
97. Gorton, I. *Essential Software Architecture*. Springer Science & Business Media, Sept. 2006.
98. Gottschalk, F., van der Aalst, W. M. P., and Jansen-Vullers, M. H. Mining Reference Process Models and Their Configurations. In *On the Move to Meaningful Internet Systems: OTM 2008 Workshops* (Berlin, Heidelberg, 2008), vol. 5333 of *LNCS*, Springer, pp. 263–272.
99. Gottschalk, F., Van Der Aalst, W. M. P., Jansen-Vullers, M. H., and La Rosa, M. Configurable workflow models. *International Journal of Cooperative Information Systems 17*, 02 (June 2008), 177–221.
100. Graf, S., Ober, I., and Ober, I. A real-time profile for UML. *International Journal on Software Tools for Technology Transfer 8*, 2 (Apr. 2006), 113–127.
101. Graversen, K. B. *The Nature of Roles: A Taxonomic Analysis of Roles as a Language Construct*. Dissertation, IT University of Copenhagen, 2006.
102. Graversen, K. B., and Østerbye, K. Implementation of a role language for object-specific dynamic separation of concerns. In *AOSD03 Workshop on Software-Engineering Properties of Languages for Aspect Technologies* (2003), Citeseer.
103. Greefhorst, D., and Proper, E. *Architecture Principles: The Cornerstones of Enterprise Architecture*. Springer Science & Business Media, Apr. 2011.
104. Guarino, N., and Guizzardi, G. Relationships and Events: Towards a General Theory of Reification and Truthmaking. In *AI*IA 2016 Advances in Artificial Intelligence* (Cham, Nov. 2016), G. Adorni, S. Cagnoni, M. Gori, and M. Maratea, Eds., vol. 10037 of *LNAI (LNCS)*, Springer, pp. 237–249.
105. Guizzardi, G. *Ontological Foundations for Structural Conceptual Models*. Dissertation, University of Twente, Enschede, 2005.
106. Guizzardi, G. Agent Roles, Qua Individuals and the Counting Problem. In *Software Engineering for Multi-Agent Systems IV* (Berlin, Heidelberg, 2006), A. Garcia, R. Choren, C. Lucena, P. Giorgini, T. Holvoet, and A. Romanovsky, Eds., vol. 3914 of *LNCS*, Springer, pp. 143–160.
107. Guizzardi, G., Herre, H., and Wagner, G. Towards Ontological Foundations for UML Conceptual Models. In *On the Move to Meaningful Internet Systems 2002: CoopIS, DOA, and ODBASE* (Berlin, Heidelberg, 2002), R. Meersman and Z. Tari, Eds., vol. 2519 of *LNCS*, Springer, pp. 1100–1117.
108. Guizzardi, G., Wagner, G., Falbo, R. d. A., Guizzardi, R. S. S., and Almeida, J. P. A. Towards Ontological Foundations for the Conceptual Modeling of Events. In *Conceptual Modeling* (Berlin, Heidelberg, Nov. 2013), W. Ng, V. C. Storey, and J. C. Trujillo, Eds., vol. 8217 of *LNCS*, Springer, pp. 327–341.
109. Haber, A., Rendel, H., Rumpe, B., and Schaefer, I. Delta Modeling for Software Architectures. In *Tagungsband Des Dagstuhl-Workshop MBEES: Modellbasierte Entwicklung Eingebetteter Systeme* (2011), p. 10.
110. Hallerbach, A., Bauer, T., and Reichert, M. Capturing variability in business process models: The Provop approach. *Journal of Software Maintenance and Evolution: Research and Practice 22*, 6-7 (2010), 519–546.

111. Halpin, T. ORM 2. In *On the Move to Meaningful Internet Systems 2005: OTM 2005 Workshops* (Berlin, Heidelberg, 2005), R. Meersman, Z. Tari, and P. Herrero, Eds., vol. 3762 of *LNCS*, Springer, pp. 676–687.
112. Halpin, T. Temporal Modeling and ORM. In *On the Move to Meaningful Internet Systems: OTM 2008 Workshops* (Berlin, Heidelberg, Nov. 2008), vol. 5333 of *LNCS*, Springer, pp. 688–698.
113. Harkes, D., and Visser, E. Unifying and Generalizing Relations in Role-Based Data Modeling and Navigation. In *Software Language Engineering* (Cham, 2014), B. Combemale, D. J. Pearce, O. Barais, and J. J. Vinju, Eds., vol. 8706 of *LNCS*, Springer International Publishing, pp. 241–260.
114. Hartmann, T., Moawad, A., Fouquet, F., and Le Traon, Y. The next evolution of MDE: A seamless integration of machine learning into domain modeling. *Software & Systems Modeling* (2017).
115. He, C., Nie, Z., Li, B., Cao, L., and He, K. Rava: Designing a Java extension with dynamic object roles. In *13th Annual IEEE International Symposium and Workshop on Engineering of Computer-Based Systems (ECBS'06)* (Mar. 2006), pp. 7 pp.–459.
116. Hebig, R., Khelladi, D. E., and Bendraou, R. Approaches to Co-Evolution of Metamodels and Models: A Survey. *IEEE Transactions on Software Engineering 43*, 5 (May 2017), 396–414.
117. Henderson-Sellers, B., Eriksson, O., and Ågerfalk, P. J. On the Need for Identity in Ontology-Based Conceptual Modelling. In *Proceedings of the 11th Asia-Pacific Conference on Conceptual Modelling* (Sydney, Australia, Jan. 2015), M. Saeki and H. Kohler, Eds., CRPIT, pp. 9–20.
118. Hendricksen, D. *12 Essential Skills for Software Architects*. Addison-Wesley Professional, 2011.
119. Hennicker, R., and Klarl, A. Foundations for Ensemble Modeling – The Helena Approach. In *Specification, Algebra, and Software: Essays Dedicated to Kokichi Futatsugi*, S. Iida, J. Meseguer, and K. Ogata, Eds., vol. 8373 of *LNCS 8373*. Springer, Berlin, Heidelberg, 2014, pp. 359–381.
120. Herrmann, K., Voigt, H., Rausch, J., Behrend, A., and Lehner, W. Robust and simple database evolution. *Information Systems Frontiers 20*, 1 (Feb. 2018), 45–61.
121. Herrmann, S. Programming with roles in ObjectTeams/Java. In *Proceedings of the 2005 AAAI Fall Symposium* (2005).
122. Herrmannsdoerfer, M., Benz, S., and Juergens, E. COPE - Automating Coupled Evolution of Metamodels and Models. In *ECOOP 2009 – Object-Oriented Programming* (Berlin, Heidelberg, 2009), S. Drossopoulou, Ed., vol. 5653 of *LNCS*, Springer, pp. 52–76.
123. Hindawi, M., Morel, L., Aubry, R., and Sourrouille, J.-L. Description and Implementation of a UML Style Guide. In *Models in Software Engineering* (Berlin, Heidelberg, 2009), M. R. V. Chaudron, Ed., vol. 5421 of *LNCS*, Springer, pp. 291–302.
124. Hippchen, B., Giessler, P., Steinegger, R., Schneider, M., and Abeck, S. Designing microservice-based applications by using a domain-driven design approach. *International Journal on Advances in Software 10*, 3&4 (2017), 432–445.
125. Hochgeschwender, N., Schneider, S., Voos, H., Bruyninckx, H., and Kraetzschmar, G. K. Graph-based software knowledge: Storage and semantic querying of domain models for run-time adaptation. In *Proceedings of the 2016 IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots, SIMPAR* (Dec. 2016), IEEE, pp. 83–90.
126. Hofreiter, B., Huemer, C., Kappel, G., Mayrhofer, D., and vom Brocke, J. Inter-organizational Reference Models – May Inter-organizational Systems Profit from Reference Modeling? In *Business System Management and Engineering: From Open Issues to Applications*, C. A. Ardagna, E. Damiani, L. A. Maciaszek, M. Missikoff, and M. Parkin, Eds., vol. 7350 of *LNCS*. Springer, Berlin, Heidelberg, 2012, pp. 32–47.
127. Horkoff, J., Aydemir, F. B., Li, F.-L., Li, T., and Mylopoulos, J. Evaluating Modeling Languages: An Example from the Requirements Domain. In *Conceptual Modeling* (Cham, 2014), E. Yu, G. Dobbie, M. Jarke, and S. Purao, Eds., vol. 8824 of *LNCS*, Springer, pp. 260–274.

128. Huang, L., and Boehm, B. How Much Software Quality Investment Is Enough: A Value-Based Approach. *IEEE Software 23*, 5 (Sept. 2006), 88–95.
129. Hudak, P. Building domain-specific embedded languages. *ACM Comput. Surv. 28*, 4es (1996), 196.
130. Hunt, J. *Scala Design Patterns: Patterns for Practical Reuse and Design*. Springer International Publishing, 2013.
131. Indulska, M., Recker, J., Rosemann, M., and Green, P. Business Process Modeling: Current Issues and Future Challenges. In *Advanced Information Systems Engineering* (Berlin, Heidelberg, 2009), P. van Eck, J. Gordijn, and R. Wieringa, Eds., vol. 5565 of *LNCS*, Springer, pp. 501–514.
132. International Organization Of Standardization. ISO/IEC/IEEE 4201:2011 (E) - Systems and software engineering - Architecture description. Tech. rep., International Organization Of Standardization, 2011.
133. International Organization Of Standardization. ISO/IEC/IEEE 24765:2017 - Systems and software engineering - Vocabulary. Tech. rep., International Organization Of Standardization, 2017.
134. Jäkel, T., Kühn, T., Voigt, H., and Lehner, W. RSQL - a query language for dynamic data types. In *Proceedings of the 18th International Database Engineering & Applications Symposium, IDEAS* (2014), ACM, pp. 185–194.
135. Jarzabek, S., Ong, W. C., and Zhang, H. Handling variant requirements in domain modeling. *Journal of Systems and Software 68*, 3 (Dec. 2003), 171–182.
136. Johnson, P., and Ekstedt, M. The Tarpit – A general theory of software engineering. *Information and Software Technology 70* (Feb. 2016), 181–203.
137. Jouault, F., and Kurtev, I. Transforming Models with ATL. In *Satellite Events at the MODELS 2005 Conference* (Berlin, Heidelberg, 2006), J.-M. Bruel, Ed., vol. 3844 of *LNCS*, Springer, pp. 128–138.
138. Kamina, T., and Tamai, T. A smooth combination of role-based language and context activation. *FOAL 2010 Proceedings* (2010), 15.
139. Kaneiwa, K., Iwazume, M., and Fukuda, K. An Upper Ontology for Event Classifications and Relations. In *AI 2007: Advances in Artificial Intelligence* (Berlin, Heidelberg, Dec. 2007), M. A. Orgun and J. Thornton, Eds., vol. 4830 of *LNAI (LNCS)*, Springer, pp. 394–403.
140. Kipyegen, N. J., and Korir, W. P. Importance of software documentation. *International Journal of Computer Science Issues (IJCSI) 10*, 5 (2013), 223.
141. Kittlaus, H.-B., and Fricker, S. A. *Software Product Management: The ISPMA-Compliant Study Guide and Handbook*. Springer, Berlin, Heidelberg, 2017.
142. Klint, P., van der Storm, T., and Vinju, J. RASCAL: A Domain Specific Language for Source Code Analysis and Manipulation. In *2009 Ninth IEEE International Working Conference on Source Code Analysis and Manipulation* (Sept. 2009), pp. 168–177.
143. Korzetz, M., Kühn, R., Kegel, K., Georgi, L., Schumann, F.-W., and Schlegel, T. MilkyWay: A Toolbox for Prototyping Collaborative Mobile-Based Interaction Techniques. In *Universal Access in Human-Computer Interaction. Multimodality and Assistive Environments* (Cham, 2019), M. Antona and C. Stephanidis, Eds., vol. 11573 of *LNCS*, Springer International Publishing, pp. 477–490.
144. Kristensen, B. B. Object-Oriented Modeling with Roles. In *OOIS' 95* (London, 1996), J. Murphy and B. Stone, Eds., Springer, pp. 57–71.
145. Kröger, F., and Merz, S. *Temporal Logic and State Systems*. Texts in Theoretical Computer Science. An EATCS Series. Springer, Berlin, Heidelberg, 2008.
146. Krogstie, J. *Model-Based Development and Evolution of Information Systems: A Quality Approach*. Springer, London, 2012.
147. Krogstie, J. Quality of Conceptual Models in Model Driven Software Engineering. In *Conceptual Modeling Perspectives*. Springer, Cham, 2017, pp. 185–198.
148. Kubica, T., Shmelkin, I., and Schill, A. Towards a Development Methodology for Adaptable Collaborative Audience Response Systems. In *2019 18th International Conference on Information Technology Based Higher Education and Training (ITHET)* (Sept. 2019), pp. 1–10.

149. Kühn, T. *A Family of Role-Based Languages*. Dissertation, Technische Universität Dresden, Dresden, Germany, Aug. 2017.
150. Kühn, T., Böhme, S., Götz, S., and Assmann, U. A combined formal model for relational context-dependent roles. In *Proceedings of the 2015 ACM SIGPLAN International Conference on Software Language Engineering* (Pittsburgh, PA, USA, Oct. 2015), SLE 2015, Association for Computing Machinery, pp. 113–124.
151. Kühn, T., Leuthäuser, M., and Götz, S. A Metamodel Family for Role-Based Modeling and Programming Languages. In *Software Language Engineering* (Cham, 2014), B. Combemale, D. J. Pearce, O. Barais, and J. J. Vinju, Eds., vol. 8706 of *LNCS*, Springer, pp. 141–160.
152. Kühn, T., Werner, C., Schön, H., Zhenxi, Z., and Assmann, U. Contextual and Relational Role-Based Modeling Framework. In *Proceedings of the 45th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)* (Thessaloniki, Greece, Aug. 2019), IEEE, pp. 442–449.
153. Kühne, T. Matters of (Meta-) Modeling. *Software & Systems Modeling 5*, 4 (Nov. 2006), 369–385.
154. La Rosa, M., Dumas, M., ter Hofstede, A. H. M., and Mendling, J. Configurable multi-perspective business process models. *Information Systems 36*, 2 (Apr. 2011), 313–340.
155. Lankhorst, M. *Enterprise Architecture at Work*, fourth ed. The Enterprise Engineering Series. Springer, Berlin, Heidelberg, 2017.
156. Larman, C. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*, 3rd ed. Prentice Hall, Upper Saddle River, N.J, Oct. 2004.
157. Latella, D., Majzik, I., and Massink, M. Automatic Verification of a Behavioural Subset of UML Statechart Diagrams Using the SPIN Model-checker. *Formal Aspects of Computing 11*, 6 (Dec. 1999), 637–664.
158. Lê, L.-S., and Ghose, A. Contracts + Goals = Roles? In *Conceptual Modeling* (Berlin, Heidelberg, Oct. 2012), P. Atzeni, D. Cheung, and S. Ram, Eds., vol. 7532 of *LNCS*, Springer, pp. 252–266.
159. Leuthäuser, M. *A Pure Embedding of Roles*. PhD thesis, Technische Universität Dresden, Dresden, Germany, May 2017.
160. Leuthäuser, M., and Assmann, U. Enabling View-based Programming with SCROLL: Using roles and dynamic dispatch for etablishing view-based programming. In *Proceedings of the 2015 Joint MORSE/VAO Workshop on Model-Driven Robot Software Engineering and View-Based Software-Engineering* (New York, NY, USA, July 2015), MORSE/VAO '15, Association for Computing Machinery, pp. 25–33.
161. Li, C., Reichert, M., and Wombacher, A. Discovering Reference Models by Mining Process Variants Using a Heuristic Approach. In *Business Process Management* (Berlin, Heidelberg, Sept. 2009), U. Dayal, J. Eder, J. Koehler, and H. A. Reijers, Eds., vol. 5701 of *LNCS*, Springer, pp. 344–362.
162. Lieberman, B. A. *The Art of Software Modeling*. CRC Press, Dec. 2006.
163. Lilienthal, C. *Langlebige Software-Architekturen: Technische Schulden analysieren, begrenzen und abbauen*. Dpunkt. Verlag, Jan. 2016.
164. Löhe, J., and Legner, C. Overcoming implementation challenges in enterprise architecture management: A design theory for architecture-driven IT Management (ADRIMA). *Information Systems and e-Business Management 12*, 1 (Feb. 2014), 101–137.
165. Lucassen, G., Dalpiaz, F., van der Werf, J. M. E., and Brinkkemper, S. Forging high-quality User Stories: Towards a discipline for Agile Requirements. In *2015 IEEE 23rd International Requirements Engineering Conference (RE)* (Aug. 2015), pp. 126–135.
166. Luiten, G., Froese, T., Björk, B. C., Cooper, G., Junge, R., Karstila, K., and Oxman, R. An Information Reference Model for Architecture, Engineering, and Construction. In *Proceedings of the 1st International Conference on the Management of Information Technology for Construction* (Singapore, 1993), vol. 1993, pp. 391–406.
167. Maier, M. W. *The Art of Systems Architecting*. CRC Press, Jan. 2009.
168. Masolo, C., Guizzardi, G., Vieu, L., Bottazzi, E., and Ferrario, R. Relational Roles and Qua-Individuals. In *AAAI Fall Symposium on Roles, an Interdisciplinary Perspective* (Virginia, USA, 2005), AAAI Press, pp. 103–112.

169. Masolo, C., Vieu, L., Bottazzi, E., Catenacci, C., Ferrario, R., Gangemi, A., and Guarino, N. Social Roles and their Descriptions. In *Proceedings of the 9th International Conference on the Principles of Knowledge Representation and Reasoning* (Whistler, Canada, 2004), D. Dubois, C. Welty, and M. Williams, Eds., pp. 267–277.
170. Maximilien, E. M., and Singh, M. P. Conceptual model of web service reputation. *ACM SIGMOD Record 31*, 4 (Dec. 2002), 36–41.
171. Meekel, J., Horton, T. B., France, R. B., Mellone, C., and Dalvi, S. From Domain Models to Architecture Frameworks. In *Proceedings of the 1997 Symposium on Software Reusability* (New York, NY, USA, 1997), SSR '97, ACM, pp. 75–80.
172. Meinhardt, S., and Popp, K. Configuring Business Application Systems. In *Handbook on Architectures of Information Systems*, P. Bernus, K. Mertins, and G. Schmidt, Eds., International Handbooks on Information Systems. Springer, Berlin, Heidelberg, 2006, pp. 705–721.
173. Mell, P., and Grance, T. The NIST Definition of Cloud Computing. Tech. Rep. NIST Special Publication (SP) 800-145, National Institute of Standards and Technology (NIST), Sept. 2011.
174. Mellor, S. J., and Balcer, M. *Executable UML: A Foundation for Model-Driven Architectures*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.
175. Mellor, S. J., Scott, K., Uhl, A., and Weise, D. Model-Driven Architecture. In *Advances in Object-Oriented Information Systems* (Berlin, Heidelberg, 2002), J.-M. Bruel and Z. Bellahsene, Eds., vol. 2426 of *LNCS*, Springer, pp. 290–297.
176. Mendling, J. Event-Driven Process Chains (EPC). In *Metrics for Process Models: Empirical Foundations of Verification, Error Prediction, and Guidelines for Correctness*, J. Mendling, Ed., vol. 6 of *LNBIP*. Springer, Berlin, Heidelberg, 2008, pp. 17–57.
177. Mendling, J., Recker, J., Rosemann, M., and van der Aalst, W. Generating correct EPCs from configured C-EPCs. In *Proceedings of the 2006 ACM Symposium on Applied Computing, SAC* (Dijon, France, 2006), ACM, pp. 1505–1510.
178. Mišic, V. B., and Zhao, J. L. Evaluating the quality of reference models. In *Conceptual Modeling* (Berlin, Heidelberg, 2000), A. Laender, S. Liddle, and V. Storey, Eds., vol. 1920 of *LNCS*, Springer, pp. 484–498.
179. Moody, D. The "Physics" of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering. *IEEE Transactions on Software Engineering 35*, 6 (Nov. 2009), 756–779.
180. Morimoto, S. A Survey of Formal Verification for Business Process Modeling. In *Computational Science – ICCS 2008* (Berlin, Heidelberg, 2008), M. Bubak, G. D. van Albada, J. Dongarra, and P. M. A. Sloot, Eds., vol. 5102 of *LNCS*, Springer, pp. 514–522.
181. Murer, S., Bonati, B., and Furrer, F. J. *Managed Evolution: A Strategy for Very Large Information Systems*. Springer, Berlin, Heidelberg, 2011.
182. Mylopoulos, J. Conceptual Modelling and Telos. *Conceptual Modelling, Databases, and CASE: an Integrated View of Information System Development* (1992), 49–68.
183. Nickerson, R. C., Varshney, U., and Muntermann, J. A method for taxonomy development and its application in information systems. *European Journal of Information Systems 22*, 3 (May 2013), 336–359.
184. Object Management Group. Business Process Model and Notation (BPMN) v2.0. Tech. rep., Object Management Group, Jan. 2011.
185. Object Management Group. Model Driven Architecture (MDA) Guide rev. 2.0. Tech. rep., Object Management Group, June 2014.
186. Object Management Group. Meta Object Facility (MOF) Core Specification v2.5.1. Tech. rep., Object Management Group, Nov. 2016.
187. Object Management Group. Unified Modeling Language v2.5.1. Tech. rep., Object Management Group, Dec. 2017.
188. Olivé, A. On the design and implementation of information systems from deductive conceptual models. In *VLDB* (1989), vol. 89, Citeseer, pp. 3–11.
189. Olivé, A. *Conceptual Modeling of Information Systems*. Springer, Berlin, 2007.
190. Olivé, A., and Raventós, R. Modeling events as entities in object-oriented conceptual modeling languages. *Data & Knowledge Engineering 58*, 3 (Sept. 2006), 243–262.

191. Organization for the Advancement of Structured Information Standards. Web Services Business Process Execution Language Version 2.0. Tech. rep., Organization for the Advancement of Structured Information Standards, Apr. 2007.
192. Ould, M. A., and Ould, M. A. *Business Processes: Modelling and Analysis for Re-Engineering and Improvement*, vol. 598. Wiley Chichester, 1995.
193. Övergaard, G. A Formal Approach to Collaborations in the Unified Modeling Language. In *«UML»'99 — The Unified Modeling Language* (Berlin, Heidelberg, 1999), R. France and B. Rumpe, Eds., vol. 1723 of *LNCS*, Springer, pp. 99–115.
194. Pastor, O., España, S., Panach, J. I., and Aquino, N. Model-Driven Development. *Informatik-Spektrum 31*, 5 (Oct. 2008), 394–407.
195. Peffers, K., Tuunanen, T., Rothenberger, M. A., and Chatterjee, S. A Design Science Research Methodology for Information Systems Research. *Journal of Management Information Systems 24*, 3 (Dec. 2007), 45–77.
196. Popovic, A., Lukovic, I., Dimitrieski, V., and Djukic, V. A DSL for modeling application-specific functionalities of business applications. *Computer Languages, Systems & Structures 43* (Oct. 2015), 69–95.
197. Popović, A., Luković, I., Dimitrieski, V., and Đukić, V. An approach for modeling events in information systems. In *Proceedings of the 2017 Federated Conference on Computer Science and Information Systems* (Prague, Sept. 2017), IEEE, pp. 707–710.
198. Rabe, M., Jaekel, F.-W., and Weinaug, H. Reference Models for Supply Chain Design and Configuration. In *Proceedings of the 38th Conference on Winter Simulation, WSC* (Monterey, California, 2006), ACM, pp. 1143–1150.
199. Rademacher, F., Sorgalla, J., and Sachweh, S. Challenges of Domain-Driven Microservice Design: A Model-Driven Perspective. *IEEE Software 35*, 3 (May 2018), 36–43.
200. Rask, M., and Kragh, H. Motives for e-marketplace Participation: Differences and Similarities between Buyers and Suppliers. *Electronic Markets 14*, 4 (Dec. 2004), 270–283.
201. Recker, J., Rosemann, M., Indulska, M., and Green, P. Business Process Modeling - A Comparative Analysis. *Journal of the Association for Information Systems 10*, 4 (Apr. 2009).
202. Recker, J., Rosemann, M., van der Aalst, W., and Mendling, J. On the Syntax of Reference Model Configuration – Transforming the C-EPC into Lawful EPC Models. In *Business Process Management Workshops* (Berlin, Heidelberg, 2006), C. J. Bussler and A. Haller, Eds., vol. 3812 of *LNCS*, Springer, pp. 497–511.
203. Recker, J. C., Rosemann, M., van der Aalst, W. M. P., and Mendling, J. On the Syntax of Reference Model Configuration – Transforming the C-EPC into Lawful EPC Models. In *Business Process Management Workshops* (Berlin, Heidelberg, 2005), C. Bussler and A. Haller, Eds., vol. 3812 of *LNCS*, Springer, pp. 60–75.
204. Reinhartz-Berger, I. Towards automatization of domain modeling. *Data & Knowledge Engineering 69*, 5 (May 2010), 491–515.
205. Reinhartz-Berger, I., Soffer, P., and Sturm, A. A Domain Engineering Approach to Specifying and Applying Reference Models. In *Proceedings of the 2005 Enterprise Modelling and Information Systems Architectures* (Bonn, Germany, 2005), J. Desel and U. Frank, Eds., vol. 75 of *Lecture Notes in Informatics - Proceedings*, Gesellschaft für Informatik e.V., pp. 50–63.
206. Reinhartz-Berger, I., Soffer, P., and Sturm, A. Extending the adaptability of reference models. *IEEE Transactions on Systems, Man, and Cybernetics Part A: Systems and Humans 40*, 5 (2010), 1045–1056.
207. Reinhartz-Berger, I., and Sturm, A. Utilizing domain models for application design and validation. *Information and Software Technology 51*, 8 (Aug. 2009), 1275–1289.
208. Richardson, C. *Microservices Patterns*, 1st ed. Manning Publications, Shelter Island, NY, US, Nov. 2018.
209. Riehle, D., and Gross, T. Role Model Based Framework Design and Integration. In *Proceedings of the 13th ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPLSA* (New York, NY, USA, 1998), ACM, pp. 117–133.

210. Robinson, S., Arbez, G., Birta, L. G., Tolk, A., and Wagner, G. Conceptual modeling: Definition, purpose and benefits. In *2015 Winter Simulation Conference* (Dec. 2015), pp. 2812–2826.

211. Rosemann, M., Sedera, W., and Gable, G. Critical Success Factors of Process Modeling for Enterprise Systems. *AMCIS 2001 Proceedings* (Dec. 2001).

212. Rosemann, M., and van der Aalst, W. M. P. A configurable reference modelling language. *Information Systems 32*, 1 (Mar. 2007), 1–23.

213. Rosemann, M., and vom Brocke, J. The Six Core Elements of Business Process Management. In *Handbook on Business Process Management 1: Introduction, Methods, and Information Systems*, J. vom Brocke and M. Rosemann, Eds., International Handbooks on Information Systems. Springer, Berlin, Heidelberg, 2015, pp. 105–122.

214. Sandkuhl, K., Stirna, J., Persson, A., and Wissotzki, M. *Enterprise Modeling*. The Enterprise Engineering Series. Springer, Berlin, Heidelberg, 2014.

215. Sarcar, V. *Design Patterns in C#: A Hands-on Guide with Real-World Examples*. Apress, 2018.

216. Schaefer, I., and Damiani, F. Pure delta-oriented programming. In *Proceedings of the 2nd International Workshop on Feature-Oriented Software Development, FOSD* (Eindhoven, The Netherlands, 2010), ACM Press, pp. 49–56.

217. Scheer, A.-W. *ARIS - Business Process Modeling*. Springer Science & Business Media, Dec. 2012.

218. Scheer, A.-W. *Business Process Engineering: Reference Models for Industrial Enterprises*. Springer Science & Business Media, Dec. 2012.

219. Schmidt, A., Otto, B., and Österle, H. A Functional Reference Model for Manufacturing Execution Systems in the Automotive Industry. In *Wirtschaftsinformatik Proceedings 2011* (2011), pp. 302–311.

220. Schön, H. Role-based Adaptation of Domain Reference Models: Suggestion of a Novel Approach. In *Tagungsband Multikonferenz Wirtschaftsinformatik 2018* (Lüneburg, Germany, Mar. 2018), P. Drews, B. Funk, P. Niemeyer, and L. Xie, Eds., vol. 4, Leuphana Universität Lüneburg, pp. 1447–1453.

221. Schön, H., and Furrer, F. J. Gute Softwarearchitektur ist Business Value: Ein Ansatz zur Bewertung von SW-Architektur. *Informatik-Spektrum 41*, 4 (Aug. 2018), 240–249.

222. Schön, H., Hentschel, R., and Bley, K. Adaptation of a Cloud Service Provider's Structural Model via BROS. In *Proceedings of the 2019 Americas Conference on Information Systems* (Cancún, Mexico, Aug. 2019).

223. Schön, H., Strahringer, S., Furrer, F. J., and Kühn, T. Business Role-Object Specification: A Language for Behavior-aware Structural Modeling of Business Objects. In *Proceedings of the 14th International Conference on Wirtschaftsinformatik* (Siegen, Germany, Feb. 2019), Siegen University.

224. Schön, H., Zdravkovic, J., Stirna, J., and Strahringer, S. A Role-Based Capability Modeling Approach for Adaptive Information Systems. In *The Practice of Enterprise Modeling* (Cham, 2019), J. Gordijn, W. Guédria, and H. A. Proper, Eds., vol. 369 of *LNBIP*, Springer International Publishing, pp. 68–82.

225. Schuette, R., and Rotthowe, T. The Guidelines of Modeling – An Approach to Enhance the Quality in Information Models. In *Conceptual Modeling* (Berlin, Heidelberg, Nov. 1998), T.-W. Ling, S. Ram, and M. L. Lee, Eds., vol. 1507 of *LNCS*, Springer, pp. 240–254.

226. Schütte, R. *Grundsätze ordnungsmäßiger Referenzmodellierung: Konstruktion konfigurations- und anpassungsorientierter Modelle*. Springer, July 2013.

227. Schütz, C., and Schrefl, M. Customization of Domain-Specific Reference Models for Data Warehouses. In *Proceedings of the 18th IEEE International Enterprise Distributed Object Computing Conference, EDOC* (Ulm, Germany, 2014), IEEE, pp. 61–70.

228. Schütze, L., and Castrillon, J. Analyzing State-of-the-Art Role-based Programming Languages. In *Proceedings of the International Conference on the Art, Science, and Engineering of Programming - Programming '17* (Brussels, Belgium, 2017), ACM Press, pp. 1–6.

229. Schütze, L., and Castrillon, J. Efficient late binding of dynamic function compositions. In *Proceedings of the 12th ACM SIGPLAN International Conference on Software Language Engineering - SLE 2019* (Athens, Greece, 2019), ACM Press, pp. 141–151.
230. Seidewitz, E. What Models Mean. *IEEE Software 20*, 5 (Sept. 2003), 26–32.
231. Selic, B. The pragmatics of model-driven development. *IEEE Software 20*, 5 (Sept. 2003), 19–25.
232. Selic, B., and Gérard, S. *Modeling and Analysis of Real-Time and Embedded Systems with UML and MARTE: Developing Cyber-Physical Systems*. Morgan Kaufmann, Waltham, MA, USA, 2014.
233. Sendall, S., and Kozaczynski, W. Model transformation: The heart and soul of model-driven software development. *IEEE Software 20*, 5 (Sept. 2003), 42–45.
234. Sharp, H., and Robinson, H. Three 'C's of Agile Practice: Collaboration, Co-ordination and Communication. In *Agile Software Development: Current Research and Future Directions*, T. Dingsøyr, T. Dybå, and N. B. Moe, Eds. Springer, Berlin, Heidelberg, 2010, pp. 61–85.
235. Sheng, Y., Zhu, H., Zhou, X., and Hu, W. Effective Approaches to Adaptive Collaboration via Dynamic Role Assignment. *IEEE Transactions on Systems, Man, and Cybernetics: Systems 46*, 1 (Jan. 2016), 76–92.
236. Silva, F. S., Soares, F. S. F., Peres, A. L., De Azevedo, I. M., Pinto, P. P., and De Lemos Meira, S. R. A reference model for agile quality assurance: Combining agile methodologies and maturity models. In *Proceedings of the 9th International Conference on the Quality of Information and Communications Technology, QUATIC* (Guimaraes, Portugal, 2014), IEEE, pp. 139–144.
237. Silva, V., Garcia, A., Brandão, A., Chavez, C., Lucena, C., and Alencar, P. Taming Agents and Objects in Software Engineering. In *Software Engineering for Large-Scale Multi-Agent Systems* (Berlin, Heidelberg, 2003), A. Garcia, C. Lucena, F. Zambonelli, A. Omicini, and J. Castro, Eds., vol. 2603 of *LNCS*, Springer, pp. 1–26.
238. Singh, Y., and Sood, M. Model Driven Architecture: A Perspective. In *2009 IEEE International Advance Computing Conference* (Patiala, India, Mar. 2009), IEEE, pp. 1644–1652.
239. Snoeck, M., and Dedene, G. Generalization/specialization and role in object oriented conceptual modeling. *Data & Knowledge Engineering 19*, 2 (June 1996), 171–195.
240. Soffer, P., Reinhartz-berger, I., and Sturm, A. Facilitating Reuse by Specialization of Reference Models for Business Process Design. In *Proceedings of the 19th International Conference on Advanced Information Systems Engineering, CAiSE* (Trondheim, Norway, 2007).
241. Sonnenberg, C., and vom Brocke, J. Evaluations in the Science of the Artificial - Reconsidering the Build-Evaluate Pattern in Design Science Research. In *Design Science Research in Information Systems* (Las Vegas, NV, USA, 2012), K. Peffers, M. Rothenberger, and B. Kuechler, Eds., Springer, pp. 381–397.
242. Stark, J. *Using Secondary Notation to Influence the Model User's Attention*. Dissertation, Technische Universität Dresden, Dresden, Germany, 2018.
243. Stark, J., and Esswein, W. Rules from Cognition for Conceptual Modelling. In *Conceptual Modeling* (Berlin, Heidelberg, 2012), P. Atzeni, D. Cheung, and S. Ram, Eds., vol. 7532 of *LNCS*, Springer, pp. 78–87.
244. Steimann, F. On the representation of roles in object-oriented and conceptual modelling. *Data and Knowledge Engineering 35*, 1 (2000), 83–106.
245. Steimann, F. A Radical Revision of UML's Role Concept. Tech. rep., University of Hannover, Hannover, 2000.
246. Steimann, F. Role = Interface : A Merger of Concepts. *Journal of Object Oriented Programming*, November (2001), 23–32.
247. Steimann, F. Domain models are aspect free. In *Model Driven Engineering Languages and Systems* (Berlin, Heidelberg, Oct. 2005), L. Briand and C. Williams, Eds., vol. 3713 of *LNCS*, Springer, pp. 171–185.
248. Steimann, F. The Role Data Model Revisited. *Applied Ontology 2*, 2 (2007), 89–103.

249. Stirna, J., Zdravkovic, J., Grabis, J., and Sandkuhl, K. Development of Capability Driven Development Methodology: Experiences and Recommendations. In *The Practice of Enterprise Modeling* (Cham, 2017), G. Poels, F. Gailly, E. Serral Asensio, and M. Snoeck, Eds., vol. 305 of *LNBIP*, Springer International Publishing, pp. 251–266.
250. Strahringer, S. Im Zentrum neuer Konzepte: Die Änderbarkeit von Software. *HMD-Praxis der Wirtschaftsinformatik (231)* (2003).
251. Taing, N., Springer, T., Cardozo, N., and Schill, A. A Dynamic Instance Binding Mechanism Supporting Run-time Variability of Role-based Software Systems. In *Companion Proceedings of the 15th International Conference on Modularity* (Málaga, Spain, 2016), MODULARITY Companion 2016, ACM, pp. 137–142.
252. Taing, N., Springer, T., Cardozo, N., and Schill, A. A Rollback Mechanism to Recover from Software Failures in Role-based Adaptive Software Systems. In *Proceedings of the International Conference on the Art, Science, and Engineering of Programming - Programming '17* (Brussels, Belgium, 2017), ACM Press, pp. 1–6.
253. Tešanovic, A. What is a pattern. Technical Report, Linköping Univeristy, Linköping, Sweden, 2004.
254. Teschke, T., and Ritter, J. Towards a Foundation of Component-Oriented Software Reference Models. In *Generative and Component-Based Software Engineering* (Berlin, Heidelberg, 2001), G. Butler and S. Jarzabek, Eds., Springer, pp. 70–84.
255. Thomas, O. Understanding the Term Reference Model in Information Systems Research: History, Literature Analysis and Explanation. In *Business Process Management Workshops* (Berlin, Heidelberg, Sept. 2005), vol. 3812 of *LNCS*, Springer, pp. 484–496.
256. Tirtarasa, S., and Zarriess, B. Projection in a Description Logic of Context with Actions. In *Proceedings of the 32nd International Workshop on Description Logics* (Oslo, Norway, June 2019), M. Šimkus and G. Weddel, Eds., vol. 2373 of *CEUR*.
257. Turnitsa, C., Padilla, J. J., and Tolk, A. Ontology for Modeling and Simulation. In *Proceedings of the 2010 Winter Simulation Conference* (Dec. 2010), pp. 643–651.
258. Valerio, A., Succi, G., and Fenaroli, M. Domain analysis and framework-based software development. *ACM SIGAPP Applied Computing Review 5*, 2 (Sept. 1997), 4–15.
259. van der Aalst, W. M. P., Barros, A. P., ter Hofstede, A. H. M., and Kiepuszewski, B. Advanced Workflow Patterns. In *Cooperative Information Systems* (Berlin, Heidelberg, 2000), P. Scheuermann and O. Etzion, Eds., vol. 1901 of *LNCS*, Springer, pp. 18–29.
260. van der Aalst, W. M. P., and ter Hofstede, A. H. M. YAWL: Yet another workflow language. *Information Systems 30*, 4 (June 2005), 245–275.
261. van der Straeten, R., Mens, T., Simmonds, J., and Jonckers, V. Using Description Logic to Maintain Consistency between UML Models. In *«UML» 2003 - The Unified Modeling Language. Modeling Languages and Applications* (Berlin, Heidelberg, 2003), P. Stevens, J. Whittle, and G. Booch, Eds., vol. 2863 of *LNCS*, Springer, pp. 326–340.
262. van der Zee, D.-J., Kotiadis, K., Tako, A. A., Pidd, M., Balci, O., Tolk, A., and Elder, M. Panel discussion: Education on conceptual modeling for simulation - challenging the art. In *Proceedings of the 2010 Winter Simulation Conference* (Dec. 2010), pp. 290–304.
263. van Hentenryck, P., and Deville, Y. The Cardinality Operator: A New Logical Connective for Constraint Logic Programming. Tech. Rep. CS-90-24, Brown University, Department of Computer Science, Oct. 1990.
264. van Renssen, A. *Semantic Information Modeling Methodology*. Lulu.com, Nov. 2015.
265. Verelst, J. The Influence of the Level of Abstraction on the Evolvability of Conceptual Models of Information Systems. *Empirical Software Engineering 10*, 4 (Oct. 2005), 467–494.
266. Verhagen, M. Drawing TimeML Relations with TBox. In *Annotating, Extracting and Reasoning about Time and Events*, vol. 4795 of *LNCS*. Springer, Berlin, Heidelberg, 2007, pp. 7–28.
267. Voelter, M. *DSL Engineering - Designing, Implementing and Using Domain-Specific Languages*. dslbook.org, 2013.
268. Voelter, M., Stahl, T., Bettin, J., Haase, A., and Helsen, S. *Model-Driven Software Development: Technology, Engineering, Management*. John Wiley & Sons, June 2013.

269. vom Brocke, J. Design principles for reference modeling: Reusing information models by means of aggregation, specialisation, instantiation, and analogy. In *Reference Modeling for Business Systems Analysis*, P. Fettke and P. Loos, Eds. IGI Global, Hershey, PA, USA, 2007, pp. 47–76.
270. Wachsmuth, G. Metamodel Adaptation and Model Co-adaptation. In *ECOOP 2007 - Object Oriented Programming* (Berlin, Heidelberg, 2007), E. Ernst, Ed., vol. 4609 of *LNCS*, Springer, pp. 600–624.
271. Wand, Y., and Weber, R. Research Commentary: Information Systems and Conceptual Modeling - A Research Agenda. *Information Systems Research 13*, 4 (Dec. 2002), 363–376.
272. Watahiki, K., Ishikawa, F., and Hiraishi, K. Formal verification of business processes with temporal and resource constraints. In *2011 IEEE International Conference on Systems, Man, and Cybernetics* (Oct. 2011), pp. 1173–1180.
273. Weber, B., Reichert, M., and Rinderle-Ma, S. Change Patterns and Change Support Features - Enhancing Flexibility in Process-aware Information Systems. *Data Knowl. Eng. 66*, 3 (Sept. 2008), 438–466.
274. Weidlich, M. *Behavioural profiles: a relational approach to behaviour consistency*. PhD thesis, Universität Potsdam, Potsdam, Germany, Dec. 2011.
275. Weissbach, M., and Springer, T. Coordinated execution of adaptation operations in distributed role-based software systems. In *Proceedings of the Symposium on Applied Computing - SAC '17* (Marrakech, Morocco, 2017), ACM Press, pp. 45–50.
276. Werner, C., and Assmann, U. Model Synchronization with the Role-oriented Single Underlying Model. In *Proceedings of the MODELS 2018 Workshops* (2018), pp. 62–71.
277. Werner, C., Schön, H., Kühn, T., Götz, S., and Assmann, U. Role-Based Runtime Model Synchronization. In *Proceedings of the 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)* (Aug. 2018), IEEE, pp. 306–313.
278. Wieringa, R., de Jonge, W., and Spruit, P. Using Dynamic Classes and Role Classes to Model Object Migration. *Theory and Practice of Object Systems 1*, 1 (1995), 61–83.
279. Winter, R., and Fischer, R. Essential Layers, Artifacts, and Dependencies of Enterprise Architecture. *Journal of Enterprise Architecture*, May (2007).
280. Winter, R., and Schelp, J. Reference modeling and method construction. In *Proceedings of the 2006 ACM Symposium on Applied Computing, SAC* (Dijon, France, 2006), ACM, pp. 1561–1562.
281. Wohed, P., van der Aalst, W. M. P., Dumas, M., ter Hofstede, A. H. M., and Russell, N. On the Suitability of BPMN for Business Process Modelling. In *Business Process Management* (Berlin, Heidelberg, Sept. 2006), S. Dustdar, J. L. Fiadeiro, and A. P. Sheth, Eds., vol. 4102 of *LNCS*, Springer, pp. 161–176.
282. Wortmann, J., Hammer, D., Goossenaerts, J., and Aerts, A. On the relation between Business, Business Model, Software, and ICT Platform Architectures. *Proceedings of ICT-Architecture 99* (1999).
283. Wutzler, M., Springer, T., and Schill, A. RoleDiSCo: A Middleware Architecture and Implementation for Coordinated On-Demand Composition of Smart Service Systems in Decentralized Environments. In *2017 IEEE 2nd International Workshops on Foundations and Applications of Self* Systems (FAS*W)* (Tucson, AZ, USA, Sept. 2017), IEEE, pp. 39–44.
284. Ye, X., Zhou, J., and Song, X. On reachability graphs of Petri nets. *Computers & Electrical Engineering 29*, 2 (Mar. 2003), 263–272.
285. Zhao, L. Designing Application Domain Models with Roles. In *Model Driven Architecture* (Berlin, Heidelberg, 2005), U. Assmann, M. Akşit, and A. Rensink, Eds., vol. 3599 of *LNCS*, Springer, pp. 1–16.
286. Zhu, H., Hou, M., and Zhou, M. Adaptive Collaboration Based on the E-CARGO Model. *International Journal of Agent Technologies and Systems (IJATS) 4*, 1 (Jan. 2012), 59–76.

Glossary

Adaptation The retrieval of an application model from a reference model by using an adaptation method.

Adaptation Implementation The application and execution of a particular adaptation method for a given reference model.

Adaptation Mechanism A type of an atomic change of a structural model element.

Adaptation Method A set of possible adaptation mechanisms together with the specification of instructions, rules, and guidelines for their application.

Application Model A user-generated model, derived from the reference model, tailored by an adaptation for an (enterprise-)specific domain.

Background Process A business process, behavior specification, or (implemented) algorithm, defining the detailed execution of the actual program or system, which serves as a source of events in a Business Role-Object Specification model.

Business Logic The enterprise's individual and characteristic solution to a (functional domain) problem in terms of executed activities and states.

Business Object A non-technical entity of a domain.

Business Process "A set of one or more linked procedures or activities executed following a predefined order which collectively realize a business objective or policy goal, normally within the context of an organizational structure defining functional roles or relationships." [48, p. 126]

Business Requirements Functional (non-technical) requirements that are driven by the given business logic.

Context "Context is any information that can be used to characterise the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves." [64, p. 2]

Compound Object An object including all roles and events attached to it. (derived from [159])

Concept “An idea or mental image which corresponds to some distinct entity or class of entities, or to its essential features, or determines the application of a term (especially a predicate), and thus plays a part in the use of reason or language.” [https://www.lexico.com/definition/concept, acc. 03.2020]

Conceptual Model A formal description of an aspect of a system for the purposes of understanding and communicating the semantics of the system. (derived from [182])

Domain “An area of knowledge that uses common concepts for describing requirements, problems, capabilities, and solutions.” [207, p. 1275]

Domain Model “A product of domain analysis that provides a representation of the requirements of the domain.” [133, p. 146]

Enterprise Architecture “A coherent whole of principles, methods, and models that are used in the design and realization of an enterprise’s organizational structure, business processes, information systems, and infrastructure.” [155, p. 3]

Enterprise Modeling The process of describing the current or future state of an enterprise regarding the commonly shared enterprise knowledge of the stakeholders. The resulting enterprise model consists of a number of related models, each focusing on a particular aspect of the enterprise. (derived from [214])

Entity A unique model element, which can have a state, is able to perform behavior, and can be connected to other elements through relationships.

Event A contextual modeling construct for specifying the temporal behavior of roles. It occurs at a specific instant of time, with possible re-occurrences. An event may have causes to define the entities that may trigger the event and determines the object fulfillment of roles and start and end of scenes.

Fulfillment The relationship that assigns a role to an object.

Granularity The choice of quality, width, depth, abstraction, and pragmatism of a model to satisfy a stakeholder’s requirements. (derived from [164])

Methodology “A system of methods used in a particular area of study or activity.” [https://www.lexico.com/definition/methodology, acc. 04.2020]

Model Concept A coherent set of model elements combined with their application purpose to express model statements.

Model Element Any separable part of the model, based on a certain metamodel type and part of a language concept.

Model Statement A declaration about a model state or content that can be *true* or *false*.

Object The central modeling construct for representing business objects within a given business domain. The object has its own identity and is uniquely identifiable among all other objects. It contains all necessary information to represent the business object with its universally valid properties, behavior, and relationships within the system.

Owner A person or unit who is responsible for development and maintenance of the reference model.

Player An object instance that uses a fulfillment towards a role instance at runtime.

Progressive Adaptation The Business Role-Object Specification adaptation method of a reference model towards an application model by using the given objects as a guideline for the adaptation design decisions.

Reference Model "Describes a standard decomposition of a known problem domain into a collection of interrelated parts, or components, that cooperatively solve the problem." [178, p. 484]

Regressive Adaptation The Business Role-Object Specification adaptation method of a reference model towards an application model by using the targeted (behavior) environment and context as a guideline for the adaptation design decisions.

Relationship A unique model element, which describes the connection between entities.

Role A contextual modeling construct with state and behavior that is fulfilled by an object or its roles to represent it in the user's context and extend or change its corresponding specifications and interactions. However, a role does not modify identity: when either permanently or temporarily fulfilling a role instance by an object instance, the object instance's identity remains unchanged. (see [220])

Role Fulfillment An object's instance is able to play a deep role without an own fulfillment towards the deep role if, and only if, the same object already fulfills another role that fulfills the deep role.

Scene An instantiated, temporary collaboration context of roles and events that are related to the same business logic part.

Software Architecture "Fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution." [132, p. 2])

Software Engineering "The systematic application of scientific and technological knowledge, methods, and experience to the design, implementation, testing, and documentation of software to optimize its production, support, and quality." [133, p. 418]

User A person who adapts a reference model to create the individual application model.

Index

Declaration of Authorship

I hereby certify that I have authored this Dissertation entitled

> *Role-based Adaptation of Business Reference Models to Application Models – An Enterprise Modeling Methodology for Software Construction*

independently and without undue assistance from third parties. No other than the resources and references indicated in this thesis have been used. I have marked both literal and accordingly adopted quotations as such. They were no additional persons involved in the spiritual preparation of the present thesis. I am aware that violations of this declaration may lead to subsequent withdrawal of the degree.

Hendrik Schön

Dresden, April 2020 *Hendrik Schön*