# Goal-oriented Business Process Engineering

Benjamin Nagel

A PhD thesis submitted to the
Faculty of Computer Science, Electrical Engineering, and Mathematics of the
University of Paderborn
in partial fulfillment of the requirements for the degree of
*doctor rerum naturalium (Dr. rer. nat.)*

supervised by
Prof. Dr. Gregor Engels

Paderborn, January 2015

# Danksagung

Während der Arbeit an dieser Dissertation habe ich die Unterstützung vieler Personen erfahren, bei denen ich mich an dieser Stelle gerne bedanken möchte.

Der erste Dank gilt meinem Doktorvater Prof. Dr. Gregor Engels für die Möglichkeit in seiner Arbeitsgruppe zu forschen und die wissenschaftliche Betreuung meiner Promotion. Ebenfalls möchte ich mich bei Prof. Dr. Jochen Küster für die Erstellung des Zweitgutachtens, sowie den weiteren Mitgliedern meiner Prüfungskommission Prof. Dr. Uwe Kastens, Dr. Matthias Meyer und Prof. Dr. Hans Kleine Büning bedanken.

Ich danke den (ehemaligen) Mitarbeiter der AG Engels und des s-labs für die kollegiale Arbeitsatmosphäre, sowie den vielen wertvollen Diskussionen und konstruktiven Hilfestellungen. Neben der fachlichen Unterstützung danke ich Friedhelm Wegener, Beatrix Wiechers und Sonja Saage für die technische und administrative Hilfe. Dr. Stefan Sauer danke ich für die erfolgreiche gemeinsame Projektarbeit und die Möglichkeit im s-lab zu arbeiten.

Besonderen Dank möchte ich an meine Bürokollegen und Freunde Fabian Christ und Henning Wachsmuth richten. Ich danke euch für die tolle gemeinsame Zeit, jede Menge Spaß und das ihr meine Launen so geduldig ertragen habt! Für die kurzweiligen Mittagspausen danke ich außerdem den Kicker-Freunden Jan Bals, Christian Gerth, Markus Luckey und Yavuz Sancar.

Für das Korrekturlesen und die vielen hilfreichen Kommentare und Anmerkungen danke ich meinem guten Freund Matthias Ammon.

Ein großer Dank geht an meine Familie für ihre unschätzbare Unterstützung, die weit über den Rahmen dieser Arbeit hinausgeht. Mama und Papa - mei-

ne Dankbarkeit für eure bedingungslose Liebe und die Selbstverständlichkeit Eures Rückhalts lässt sich nur schwer in Worte fassen. Meiner Schwester Pia, meinen Schwiegereltern und Schwägern danke ich für das Zuhören und Dasein in den vielen schönen und auch den schwierigen Momenten.

Mein letzter und größter Dank geht an meine geliebte, kleine Familie die mir so viel Kraft gegeben hat. Tanja, du hast mir in der Zeit der Promotion nicht nur den Rücken freigehalten, sondern uns auch ein Zuhause voller Geborgenheit geschenkt. Julius, dir danke ich dafür, dass du mich an deiner Welt teilhaben lässt und mir immer wieder zeigst, welche Dinge im Leben wirklich wichtig sind!

# Abstract

Service-oriented architectures have emerged as an architectural style for the design of business applications. Accordingly, business process models play a central role in the description of business requirements as well as in the specification of required service compositions.

The modeling of business processes is not pursued for its own sake, but contributes to the achievement of strategic concerns represented by business goals. In order to evaluate the suitability of business process models, e.g. with respect to completeness and relevance, their relations to business goals need to be considered. This comprises the initial specification of business goals and business processes as well as the preservation of consistency between evolving models.

In previous work, several goal-oriented requirements engineering approaches and business process modeling techniques have been proposed. Nonetheless, there is no integrated specification method that supports the goal-oriented specification, the systematic derivation of business process models and the assurance of quality in a sufficient manner.

In this thesis, we present an approach for goal-oriented business process engineering that provides modeling techniques and analysis capabilities to assure the overall specification quality. Based on an existing goal-modeling notation, we introduce an extended modeling approach that supports the expression of goal dependencies, relevant business context elements and the identification of composable actions. Further, we describe a systematic derivation method to ensure the consideration of this information in the business process composition. To ensure a valid and consistent specification, a quality analysis and assurance framework is introduced. Our approach is prototypically implemented and evaluated in a project, two case studies and an experiment.

# Zusammenfassung

Service-orientierte Architekturen haben sich als Architekturstil für die Entwicklung von Geschäftsanwendungen etabliert. Dementsprechend spielen Geschäftsprozessmodelle eine zentrale Rolle sowohl für die Beschreibung fachlicher Anforderungen als auch für die Spezifikation der erforderlichen Service-Kompositionen.

Die modellierten Geschäftsprozesse dienen in der Regel keinem Selbstzweck, sondern der Erreichung eines strategischen Geschäftsziels. Um die Angemessenheit der Geschäftsprozessmodelle, zum Beispiel hinsichtlich ihrer Vollständigkeit und Relevanz, zu bewerten, müssen ihre Beziehungen und Abhängigkeiten zu den Unternehmensziele berücksichtigt werden. Diese Abhängigkeiten müssen sowohl in der initialen Spezifikation von Geschäftszielen und -prozessmodellen als auch zwischen sich ändernden Modellen berücksichtigt werden.

In bestehenden Arbeiten werden verschiedene zielorientierte Requirements Engineering Methoden und Modellierungstechniken für Geschäftsprozesse beschrieben. Trotzdem gibt es keine integrierte und durchgängige Spezifikationsmethode, die die zielorientierte Spezifikation, die systematische Ableitung von Geschäftsprozessmodellen und die Sicherstellung der Qualität ausreichend berücksichtigt.

In dieser Arbeit wird ein Ansatz zur zielorientierten Spezifikation von Geschäftsprozessen vorgestellt, der Modellierungstechniken und Qualitätsanalysefunktionen beinhaltet. Basierend auf einer vorhandenen Ziebeschreibungssprache wird ein erweiterter Modellierungsansatz vorgestellt, der die Beschreibung von Abhängigkeiten zwischen Geschäftszielen und relevanten Elementen im Geschäftskontext sowie die Identifikation von komponierbaren Aktionen unterstützt. In einem systematischen Ableitungsverfahren wird beschrie-

ben, wie diese Informationen bei der Komposition eines Geschäftsprozesses berücksichtigt werden können. Die Qualität der Spezifikation wird durch verschiedene Analyseverfahren gewährleistet. Unser Ansatz wurde prototypisch implementiert und im Rahmen eines Projekts, in zwei Fallstudien und einem Experiment evaluiert.

# Contents

# Chapter 1

# Introduction

Today's enterprises compete in globalized markets with rapidly changing business environments and customer requirements. Sustainable economic success in such dynamic markets does not only require the efficient execution of business processes, but also depends on the ability to quickly react to environmental changes. Due to the increasing degree of IT-support in business processes, the adaptability of the business processes mainly depends on the flexibility of the underlying applications [WM06].

Addressing these requirements, service-oriented architectures (SOA) emerged as an architectural style for the development of highly-flexible business applications [PTDL03, Erl05, KBS05]. SOA enables a strong alignment between business processes and the implementing business applications. This alignment facilitates the evolution of business applications in response to changing business processes. Hence, business process modeling and management have become essential tasks in the engineering of business applications.

The specification of business processes is a prerequisite for the systematic engineering of service-oriented business applications. Business process models describe sequences of actions and decision points connected by edges which define valid action execution orders. The specified business processes can be implemented by the composition of technical services which are executed according to the defined control flow, e.g. by a process engine [Bon13, IBM13, SAP13]. Through the execution of a business process a certain business goal, like the processing of a customer order, shall be achieved.

As the conceptual foundation for the development of service-oriented business applications, the efficiency and appropriateness of the business process models are the key impact factors for the overall quality and effectiveness of the resulting business application. The suitability of the modeled business processes can be evaluated with respect to their contribution to the achievement of the addressed business goals. Therefore, it needs to be ensured that each business process sufficiently achieves the related business goal (completeness), but does not include irrelevant actions that are not actually required (relevance). By explicitly relating business processes to business goals, their completeness and relevance can be evaluated [Wes12].

Goal-oriented engineering of business processes comprises the elicitation of relevant business goals, the identification of required business processes and the definition of business process models that sufficiently fulfill the stated business goals. Goal models provide a well-known approach for the elicitation and specification of business goals and their refinement in a structured hierarchy. Existing goal modeling approaches like KAOS [DFvL91, DvLF93], Tropos/i* [BPG+04, Yu96, Yu97] provide notations for the specification of business goal models.

An overview of the goal-oriented engineering of business processes is depicted in Figure 1.1. High-level stakeholder objectives are typically presented in terms of unstructured and informal ideas which are not specified explicitly. To capture these objectives in terms of requirements, they need to be formulated, prioritized and structured. By using goal models, the requirements engineer defines a structured hierarchy of business goals in an explicit model. Hereby, high-level business goals are decomposed to subgoals, which are finally decomposed to operationalized goals in terms of actions [vL03, AKRU09a]. These actions describe what needs to be done to achieve the related business goal. By composing and refining the elicited actions, the process engineer operationalizes the business goals to a business process model. This business process model defines the required action execution sequence to achieve the stated business goals.

The resulting business process model provides a specification of the required service composition that needs to be implemented in the subsequent development steps of the software engineering process. To ensure that the resulting implementation is in line with the stakeholder objectives, the consistency in

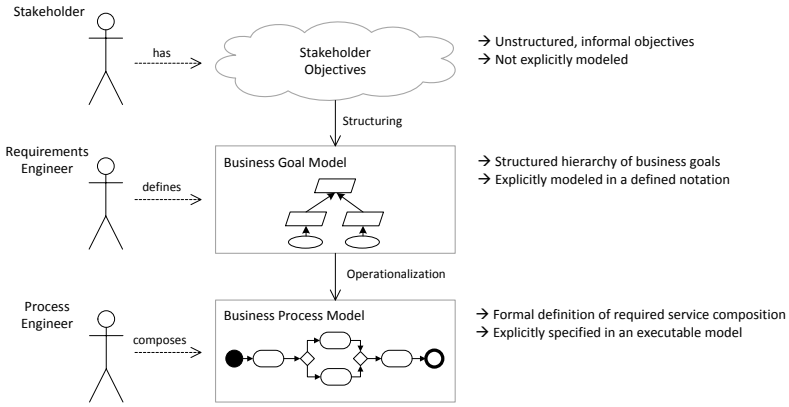


Figure 1.1: Goal-oriented Specification of Business Processes

the goal-oriented specification needs to be verified. This means that stakeholder objectives are sufficiently captured by the business goal model and the stated business goals are appropriately addressed by the derived business processes. Inconsistencies in the specification can lead to business applications performing business processes which do not sufficiently fulfill or even contradict the strategic business goals of the enterprise.

## 1.1 Problem Statement

As discussed, business process models are not only used as a foundation for communication, but also provide an operational specification of required service compositions. Figure 1.1 shows that different roles are involved in the derivation and specification of business process models. These roles have different backgrounds and use various terminologies, which makes it challenging to ensure consistency through the whole specification process. In the following, we discuss constructive specification and consistency analysis as the two main impact factors for a consistent specification of business process models.

#### Goal-oriented Specification of Business Process Models

The capturing and structuring of strategic requirements in business goal models and their operationalization to business processes have a direct impact

on the overall quality of the whole specification. To illustrate the challenging character of the composition task, Figure 1.2 shows the exemplary operationalization of a goal model excerpt. We assume that three actions ($a_1$, $a_2$ and $a_3$) have been identified in goal model $GM_1$. In the operationalization step, these actions are composed to a business process model. As depicted, the actions in $GM_1$ can be composed in various ways. In this example, we consider three possible compositions resulting in different business process models: $PM_1$, $PM_2$ and $PM_3$. Each business process model comprises the required actions specified in $GM_1$ and defines a valid composition of these actions, but obviously implements different execution logics.

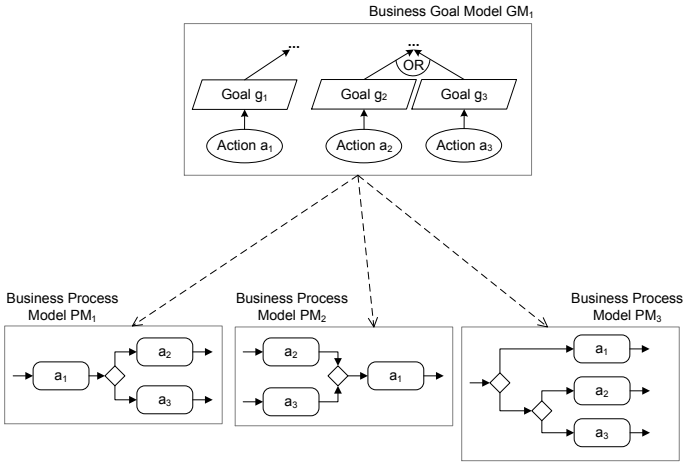


Figure 1.2: Derivation of Different Process Models from a Goal Model

To decide which composition is in line with the actual stakeholder objectives, it needs to be checked whether there are dependencies between goals that have to be considered, e.g. a specific order in which they need to be achieved. Existing approaches for goal-oriented requirements engineering [vL01] lack the ability to elicit and to specify the stakeholders' knowledge about such dependencies between goals. As a consequence, a lack of specification can lead to a missing consideration in the business process composition. For the example in Figure 1.2, this could mean that $a_1$ should always be performed before $a_2$. Accordingly, business process model $PM_1$ would be the only valid composition.

In addition to the consideration of dependencies, the composability of actions needs to be addressed as well. Actions are defined by pre- and postconditions

which need to hold before and after their execution. For the valid composition of these actions in a business process model, it needs to be ensured that the conditions of consecutive actions match correctly.

To summarize, the goal-oriented specification of business processes by existing approaches shows three weaknesses which affect the quality of the business process design. First, the goal model does not capture all relevant information from the stakeholders' knowledge. While business goals can be elicited and defined, dependencies among them are not considered sufficiently. Second, it needs to be ensured during the operationalization and specification of actions in the goal model that these actions are composable. Finally, a systematic method for the derivation of consistent business process models is required that considers the goal dependencies appropriately.

#### Consistency of Business Goals and Business Process Models

In response to environmental changes, business goal models and business process models can evolve. Elements of the model are changed, deleted or decomposed, leading to compositional changes which may affect the consistency between both models. For example, a new goal needs to be addressed by the business process, while a deleted goal could make actions obsolete in the business process model. A process designer could exchange an action in the business process, e.g. by renaming, or decomposing an action into a new subprocess. In this case, it needs to be ensured that the evolved business process still achieves the related business goals.

While business goal models are specified by using well-defined notations, the elements of the model themselves, like goals and actions, are described in natural language. Natural language is easy to use, is comprehensible to the people using and creating these models and has a high expressiveness. For the validation of the consistency between models the use of natural language has disadvantages like ambiguity and imprecision. Since people in different roles create and change the models, different understandings of terms can lead to inconsistencies in their usage.

A consistent usage of a common terminology provides an important foundation for the consistency but is not sufficient to validate the consistency. Business goal models provide an overview of the goals which need to be achieved,

while business process models describe how these goals are achieved. With respect to the given business goal model the business process model has to be *complete* and *relevant*. Completeness means that all goals are addressed sufficiently by the business process. Relevance of the business process means that each action in a business process can be linked to a business goal, expressing that it contributes to its achievement. To evaluate completeness and relevance, an explicit traceability model and a traceability analysis of the business goal model and business process models is required.

The example in Figure 1.2 illustrates derived business process models which are complete and relevant to the given business goal model. Nonetheless, the consideration of dependencies is required to ensure consistency among these models. For this purpose, the stakeholders' knowledge about dependencies among goals has to be elicited during requirements engineering and needs to be validated against the derived action compositions.

## 1.2 Solution Overview and Scientific Contributions

To address the identified problems, we propose an approach to the goal-oriented engineering of business processes. An overview of our solution is depicted in Figure 1.3. The stakeholder objectives are used as input for our approach that comprises business goal modeling, the derivation of business process models and quality analysis and assurance. The result of the outlined solution is a business process that is in line with the stated business goals.

By realizing the approach, this thesis makes the following contributions.

**Business Goal Modeling**

To elicit and specify the stakeholders' knowledge about goal dependencies we propose an extended goal modeling approach. Although goal-oriented RE approaches are widely used in research and industry, the focus of existing work lies on the specification of functional and non-functional goals and not on the identification and specification of dependencies among them. To enable a consistent specification, we propose to explicitly consider temporal

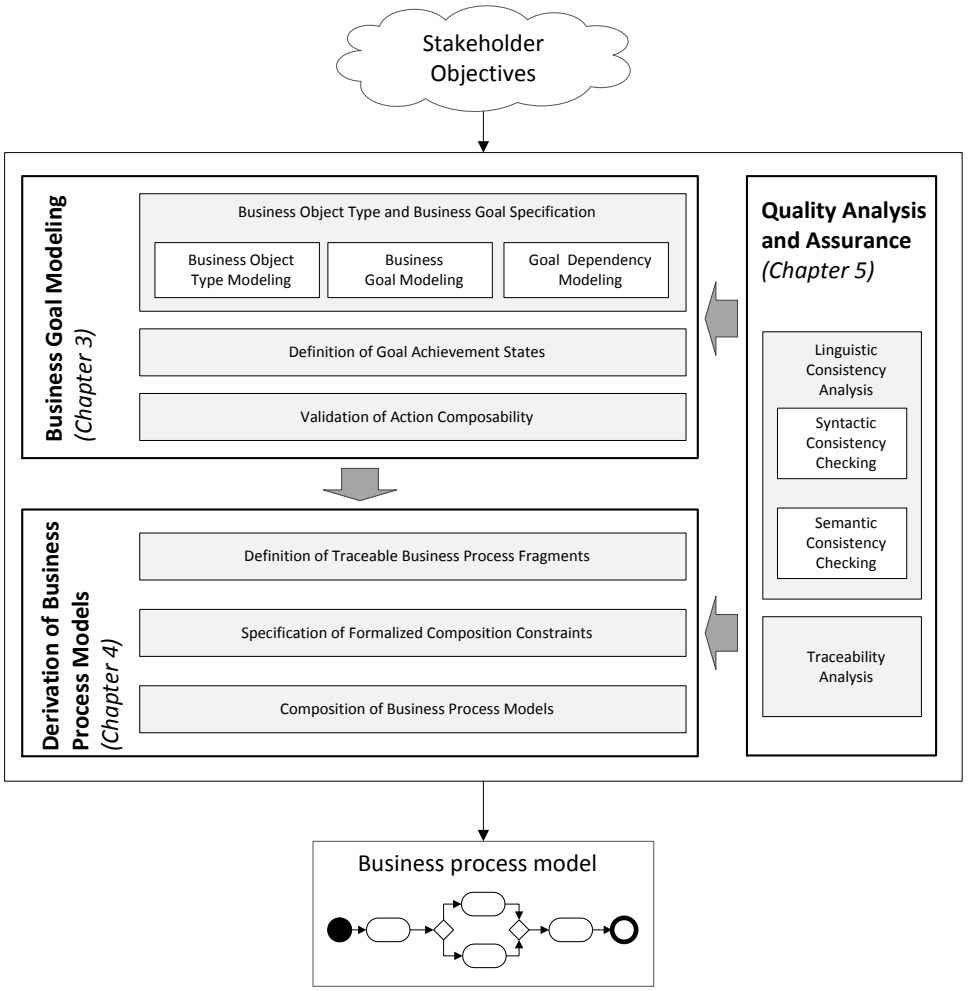


Figure 1.3: Overview about the Goal-oriented Business Process Engineering Approach

and logical dependencies in a goal model. For this purpose, a meta-model for an extended business goal modeling language is presented and goal annotation elements are defined that provide the required modeling capabilities. By using the extended modeling capabilities, the stakeholders' knowledge about dependencies between goals can be elicited and modeled in an explicit manner. The explicit specification provides the foundation for their consideration in the business process composition.

In addition to the extension of the modeling capabilities, we also present an

approach to the validation of action composability. Hereby, we ensure that the defined actions can be composed in a valid manner at an early stage of the specification process.

**Derivation of Business Process Models**

We provide a systematic derivation approach which uses a business goal model specified by the modeling techniques we have introduced. In order to enable the definition of relationships between business goals and business process elements, we present an approach to calculate and cluster business process fragments encapsulating logically related actions and to define explicit traceability links to the related business goals. Further, we demonstrate how the defined temporal dependencies among goals can be considered and validated in the derived business process model. For this purpose, an algorithm for the iterative formalization and refinement to verifiable composition constraints is presented.

Using the set of identified fragments and formalized constraints, we describe how existing composition algorithms can be integrated into our approach. Finally, we enrich the resulting business process models with context information.

**Quality Analysis and Assurance**

Inconsistencies among business goal models and business process models raise the problem of implemented processes in terms of service compositions that are not in line with the actual stakeholder objectives. Addressing this problem, we provide a quality analysis and assurance framework for automated consistency analysis. The analysis includes the validation of the linguistic consistency of the goal model and the traceability analysis to validate relevance and completeness of the derived business process model.

To support the handling of detected inconsistencies, we contribute a decision support that proposes suitable repair strategies. For this purpose, we present a catalogue of strategies and discuss their application for the semi-automatic correction of inconsistencies.

## 1.3 Publication Overview

The results presented in this thesis have been reviewed and published in the proceedings of international conferences and workshops. The following papers are directly related to the presented approach.

- Benjamin Nagel, Christian Gerth, Jennifer Post, Gregor Engels: Ensuring Consistency Among Business Goals and Business Process Models. In *Proceedings of the 17th IEEE International EDOC Conference*, pages 17-26. IEEE Computer Society, 2013.

- Fabian Pittke, Benjamin Nagel, Gregor Engels, Jan Mendling: Linguistic Consistency of Goal Models. In *Enterprise, Business-Process and Information Systems Modeling*, pages 393-407. Springer, 2014.

- Benjamin Nagel, Christian Gerth, Gregor Engels: Goal-driven Composition of Business Process Models. In *Proceedings of the 9th Workshop on Engineering Service-Oriented Applications*, pages 16-27. Springer, 2013.

- Benjamin Nagel, Christian Gerth, Jennifer Post, Gregor Engels: Kaos4SOA - Extending KAOS Models with Temporal and Logical Dependencies. In *Proceedings of the CAiSE Forum at the 25th International Conference on Advanced Information Systems Engineering*, pages 9-16. CEUR-WS.org, vol. 998, 2013.

- Benjamin Nagel, Klaus Schröder, Steffen Becker, Stefan Sauer, Gregor Engels: Kooperative Methoden- und Werkzeugentwicklung zur Cloudmigration von proprietären Anwendungskomponenten. In *Software Engineering 2015*, Fachtagung des GI-Fachbereichs Softwaretechnik, 2015 (accepted for publication).

In addition, the following papers address the applicability of our work in the field of self-adaptive systems.

- Markus Luckey, Benjamin Nagel, Christian Gerth, Gregor Engels: Adapt Cases: Extending Use Cases for Adaptive Systems. In *Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 30-39. ACM, 2011.

- Benjamin Nagel: Semi-automatische Ableitung externer Anpassungsmechanismen für selbst-adaptive Systeme. In *Proceedings of the Software Engineering 2011*, Gesellschaft für Informatik (GI) (Bonn), pages 303-308. Lecture Notes in Informatics, 2011.

- Benjamin Nagel, Christian Gerth, Enes Yigitbas, Fabian Christ, Gregor Engels: Model-driven Specification of Adaptive Cloud-based Systems. In *Proceedings of 1st International Workshop on Model-Driven Engineering for High Performance and Cloud Computing*, pages 4:1-4:6. ACM, 2012.

## 1.4 Outline of this Thesis

The remainder of this thesis is structured as follows.

- Chapter 2 lays the foundations for the presentation of our approach. We address the general concepts of model-driven software engineering and requirements engineering. Hereby, we focus on goal-oriented requirements engineering and business process modeling. Further, we discuss requirements management and different aspects of requirements quality.

- Our business goal modeling approach is described in Chapter 3. This chapter comprises a comprehensive analysis of existing goal modeling notations as well as the introduction of a new modeling technique that enables the derivation of business process models. Hereby, we consider the consistency of the goal model itself, the explicit modeling of goal dependencies and the action composability.

- In Chapter 4, we discuss our approach to the goal-oriented derivation of business process models. The derivation approach considers goal dependencies and enriches the resulting business process model with context information.

- The quality analysis and assurance of business goal and business process models is introduced in Chapter 5. Our approach considers a linguistic consistency analysis of business goal models as well as traceability analysis between business goals and business processes.

- Chapter 6 documents the implemented GooPE tool support. We describe the functionality of GooPE and present the provided user interfaces.

- In Chapter 7, the evaluation of our approach is described. We discuss the performed evaluations including a project report, two case studies and an experiment.

- We conclude this thesis in Chapter 8. We summarize the contributions of our approach and give an outlook of future work and research challenges.

- Additional information is given in the appendices. Appendix A describes a formal language grammar definition that is used for the generation of a textual editor. Applied models and results of the performed evaluations are provided in Appendix B, C and D.

# Chapter 2

# Foundations

In this chapter, we describe the required foundations and concepts for the approach presented in this thesis. First, we introduce the general concepts of model-driven software engineering in Section 2.1. In Section 2.2, we discuss fundamentals of requirements engineering, the different activities and a selection of modeling techniques which are relevant in the context of this thesis. The management of requirements throughout the software life cycle is discussed in Section 2.3. Success factors for a high-quality requirements specification and relevant quality properties are presented in Section 2.4. Finally, we introduce our running example that is used in this thesis in Section 2.5.

## 2.1 Model-driven Software Engineering

Models on various levels of abstraction are used in different phases of software projects. They provide a communication foundation for the stakeholders involved who are in different roles and have different technical backgrounds. Software engineering methods provide guidance for the systematic creation, refinement and validation of models. By defining the required steps and the actors involved, a structure for software development projects is provided. Depending on the context and the complexity of the project, different types of methods will be suitable. Agile approaches, like Scrum [Sch97], support the iterative development but also require the intensive integration of customers into the different development phases.

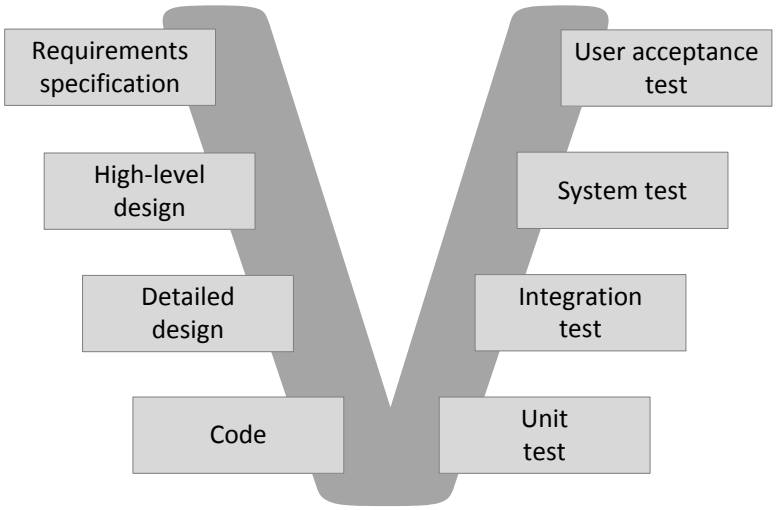


Figure 2.1: V-Model of the Software Engineering Process ([Boe84, KNR06])

A more restricted order of activities is given by traditional methods like RUP [Kru04] or the V-Model XT [Boe84, KNR06]. The phases of the V-Model are depicted in Figure 2.1. It illustrates the different steps and the interrelations among them. Starting with a requirements specification resulting from the requirements engineering phase (cf. Section 2.2), a high-level design of the software is created. This results in one or more architectural views on a high level of abstraction, which are systematically refined to a detailed design. The technical design is used as a foundation for the implementation (source code) of the software.

The structure of the V-Model makes it possible to assign the different levels of testing to the corresponding development phases. Quality assurance of the code is done by unit tests. The detailed technical design is used to apply integration tests. Based on the high-level architectural design, system tests are performed. Finally, the achievement of the specified requirements is validated by user acceptance tests.

An integrated and systematic approach for model-driven software development is provided by the Model Driven Architecture (MDA) [OMG09], introduced by the Object Management Group (OMG) in 2001. MDA focuses on forward engineering, aiming for the generation of executable code from abstract, human-elaborated models. Hereby, MDA distinguishes three model

types, providing different views on a system on varying levels of abstraction.

The computation independent model (CIM) is the most abstract representation and does not show details of the system structure. It serves as a domain model and provides a common vocabulary. Hereby, it plays an important role especially in early phases (e.g. requirements specification) to bridge the gap between domain experts and software architects [MM+03].

A platform independent model (PIM) is derived from the CIM. The PIM focuses on the description of the required functionality in a platform independent way. This means that concrete details about the way the described functionalities are realized on a specific technical platform are not specified. Such models are suitable for the definition of high-level designs in order to get a common understanding about the functionalities, without determining a concrete implementation platform.

In the last step of the MDA approach, the PIM is transformed into a platform specific model (PSM). Hereby, the generic functionality descriptions are enriched with the implementation details of a certain platform. Such details can be information about a specific framework or database that is used. The PSM can be matched to the detailed design phase of the V-Model.

To define concrete models for such types, languages are required that provide the required expressiveness in terms of modeling capabilities and are suitable for the stakeholders to work with these models. A standardized collection of modeling languages is provided by the Unified Modeling Language (UML) [OMG11b] published and developed by the OMG. The UML provides notations for the creation of models on different levels of abstraction, which are formally defined over a common meta-model. For example, use case diagrams can be used to elicit and specify functional requirements and class diagrams are suitable for the definition of relevant context elements and their relations. Encapsulating functionalities, component diagrams are widely used for the specification of the system structure. Behavioral models like activity diagrams, sequence diagrams or state charts can be used to represent the system behavior. By applying suitable UML notations, the required models in the MDA process can be defined in a precise way.

## 2.2 Requirements Engineering

In this section, we discuss the foundations of requirements engineering. This discussion includes the fundamentals of software requirements in Section 2.2.1 and the crucial activities performed in the requirements engineering process in Section 2.2.2. Goal-oriented requirements engineering approaches and business process modeling techniques are described in detail in Sections 2.2.3 and 2.2.4. Finally, we discuss domain and object modeling in Section 2.2.5.

### 2.2.1 Fundamentals of Software Requirements

Due to their different backgrounds, roles and perspectives, stakeholders involved in a project often have different understandings of a requirement. These misunderstandings are aggravated by the fact that no common definition of the term *requirement* exists [Wie03].

Brian Lawrence gives a general definition and considers everything as a requirement that leads to a design decision [Law97]. Sommerville and Sawyer discuss their variety and state that *"requirements are defined during the early stages of a system development as a specification of what should be implemented. They are descriptions of how the system should behave, or of a system property or attribute. They may be a constraint on the development process of the system."* [Law97].

For the purpose of this thesis, we adopt the widely known and applied definition provided by the IEEE Standard Glossary of Software Engineering Terminology [IEE90] and define the term requirement as follows:

**Definition 1 (Requirement)** *A requirement is defined as*

1. *a condition or capability needed by a user to solve a problem or achieve an objective.*

2. *a condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents.*

3. *a document representation or a condition or capability as in (1) or (2).*

**Classification of Requirements**

A common classification scheme for requirements distinguishes three main types of requirements: functional requirements, quality requirements and constraints [RR06, Som07, Poh10]. Another more detailed classification is given by Glinz [Gli07]. He proposes a concern-based taxonomy considering project, system and process requirements. Focusing on the system concern, he considers four different types which comprise functional requirements and constraints as well as performance and specific quality requirements.

Wiegers provides a model representing different levels of requirements and their interrelations [Wie03]. Figure 2.2 sketches an overview of this classification model. Business objectives represent strategic requirements of an enterprise that are described by a product vision or a goal model. Based on the defined project scope, user requirements are derived which define what the user will be able to do with the system. Bringing together user and system requirements, the consolidated functional requirements are defined. Functional requirements are effected by non-functional requirements. Business rules define governance and compliance guidelines or the business logic that needs to be considered. Quality requirements and constraints define additional restrictions for the implementation of the desired functionality. The result is a software requirements specification.

### 2.2.2 Requirements Engineering Activities

The process of requirements engineering comprises different activities addressing all aspects of the capturing, structuring, specification and validation of requirements [GBB+06]. As depicted in Figure 2.3, the requirements engineering process can be divided into four core activities [SS97, Wie03, ABD+04].

- **Requirements elicitation**

  This activity aims for the elicitation of all existing and new requirements at the required level of abstraction for the system to be developed [Poh10]. The process of requirements elicitation comprises five steps [ZC05]. First, an understanding of the application domain needs to be gained by investigating and examining the actual environmental context. Based on this information, sources for requirements are iden-

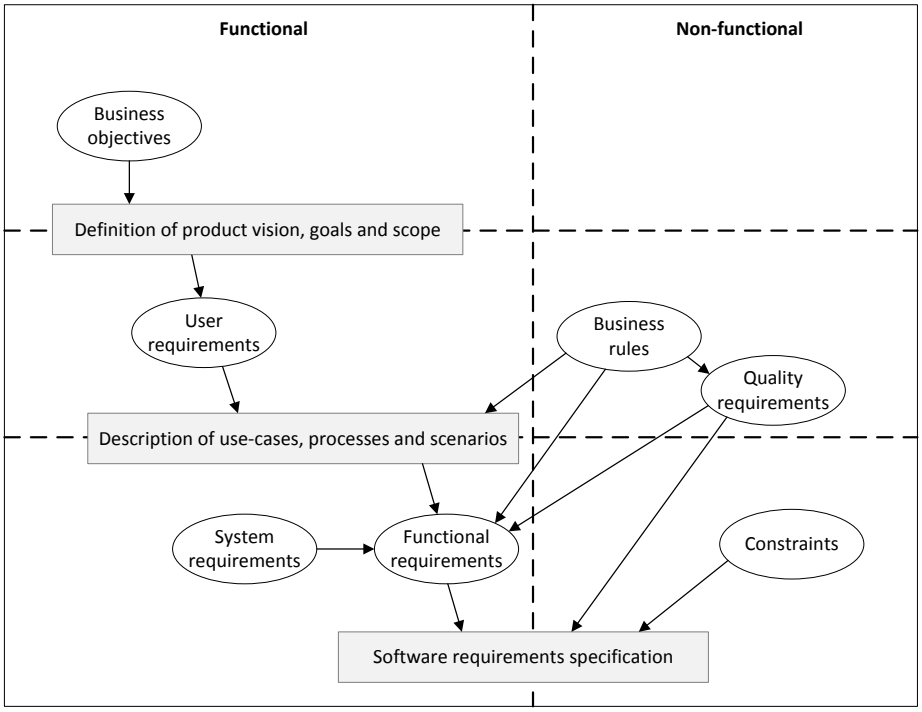


Figure 2.2: Classification and Relations of Requirements Types [Wie03]

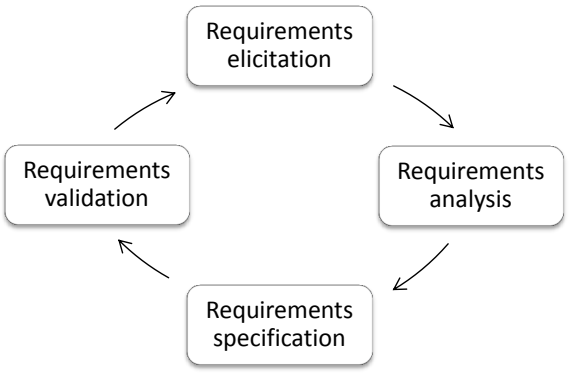


Figure 2.3: Requirements Engineering Activities

tified and the corresponding stakeholders are analyzed resulting in a list of relevant project stakeholders. According to the analysis of the project context, the available information sources and stakeholders, suitable elicitation techniques, approaches and tools are selected [HD03].

Equipped with this setup, the requirements are elicited in the last step.

- **Requirements analysis**

  Using the raw information gathered in the elicitation phase, the requirements are analyzed in-depth in the next step. Their refinement and redefinition aims to provide a common understanding for all stakeholders [Wie03]. The analysis comprises requirements classification, detection and resolution of conflicts between requirements, the evaluation of their feasibility and the negotiation of their implementation priority [BRF14]. Discussions about requirements are usually supported by conceptual models, e.g. context models, which illustrate the underlying real-world problems. Extending the conceptual modeling, a high-level system architecture is specified, usually in terms of a component diagram, which is used to allocate the requirements to components that have to satisfy them. This allocation step emphasizes the intertwined relation between requirements engineering and software architecture development [Nus01].

- **Requirements specification**

  To establish the foundation for an agreement about the requirements that are in the scope of the project and to explicitly model them for subsequent development phases (design, implementation etc.), a requirements specification is created. In the previous activities, various kinds of information have been documented, like the context information or a raw documentation of the requirements. The requirements specification is a more formal way of specifying the requirements in compliance with a set of defined specification rules and guidelines [Poh10]. The distinction between documented information, documented requirements and the actual specification of requirements is illustrated in Figure 2.4. The structure and content of a software requirements specification can vary depending on the complexity of the project. A proven standard is proposed in the IEEE Standard 830-1998 recommended practice for software requirement specifications [IEE98].

- **Requirements validation**

  Requirements validation is concerned with the process of examining the requirements specification to ensure that it defines the right software. This means a software that meets the users expectations [SK98]. For this purpose, quality criteria like completeness, correctness, consistency, unambiguousness, feasibility and comprehensibility need to be evaluated and assured. Common techniques for requirements validation are requirements reviews that validate the existing specification by inspection, static analysis of formalized requirement models or prototyping in order to evaluate the feasibility of the requirements [BRF14]. The planning of user acceptance tests based on the requirements models (cf. V-Model in Figure 2.1), is often considered as part of the requirements validation phase [Wie03].

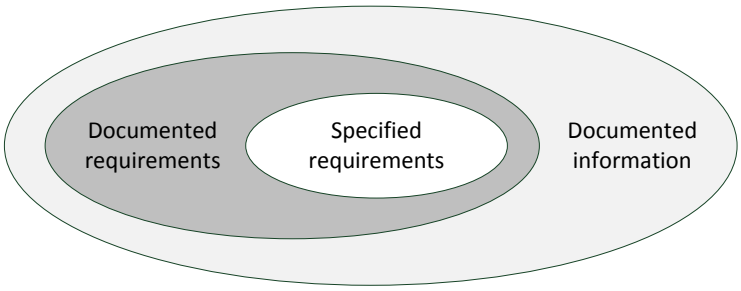


Figure 2.4: Distinction between Documentation and Specification of Information and Requirements ([Poh10])

During the described requirements engineering process, different requirements artifacts are created. Pohl [Poh10] distinguishes three kinds of requirements: goals, scenarios and solution-oriented requirements. Fernandez et.al. [Fer11, FPKB10] propose a more detailed model for structuring requirements models according to the abstraction levels and the specified modeling view. The artifacts which are relevant in the context of this thesis are highlighted in the overview sketched in Figure 2.5.

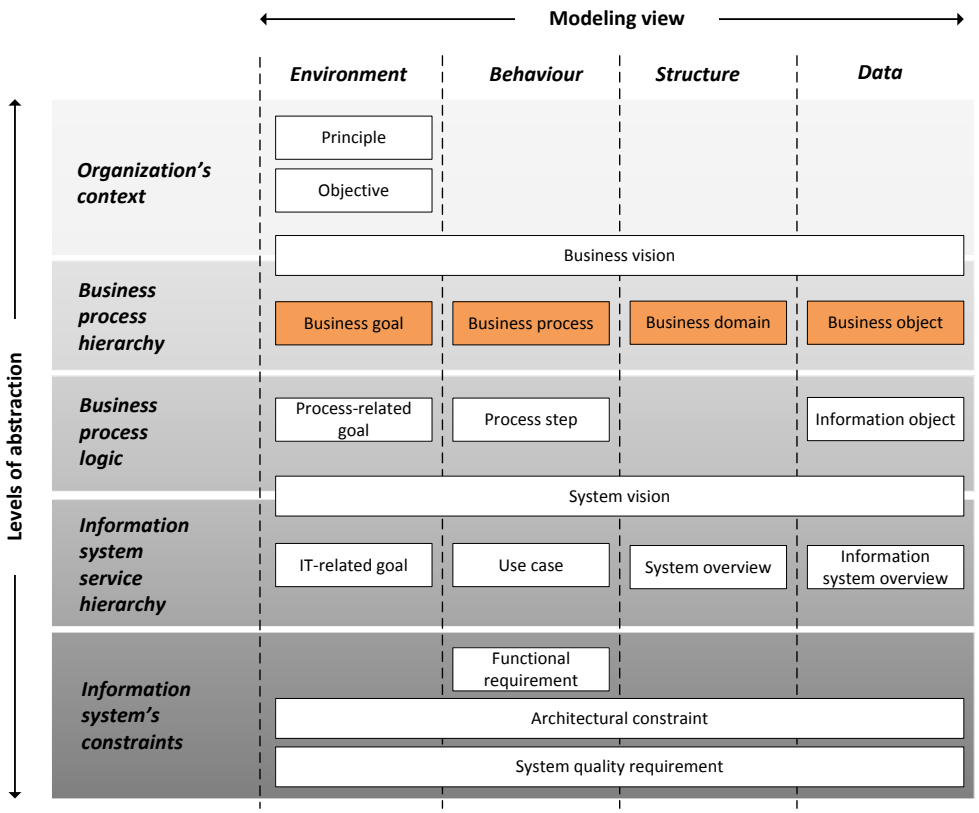


Figure 2.5: Abstraction Levels and Modeling Views for Requirements Artifacts [Fer11, FPKB10]

### 2.2.3 Goal-oriented Requirements Engineering

To enable the specification of requirements, the requirements engineers need to understand the stakeholders' intentions and objectives. Such objectives are defined and specified as goals in the requirements engineering process [Poh10]. Based on van Lamsweerde [vL01] we define a goal as follows.

**Definition 2 (Goal)** *(based on [vL01]) Goals capture, at different levels of abstraction, the various objectives the system under consideration should achieve.*

Goals can be formulated on different levels of abstraction, ranging from strategic goals on a higher abstraction level to low-level concerns that describe tech-

nical requirements. The defined goals can represent a functional property or a quality property of the system that shall be developed [KW09]. To enable the consideration of these goals, they need to be specified in an explicit goal model. Several established approaches provide notations for the specification of goal models [DvLF93, Yu96, Yu97, GKMP04]. The most popular modeling languages are KAOS, i* and the Tropos methodology. These goal modeling notations support the specification of common goal model concepts: decomposition, operationalization and responsibility.

- **Decomposition**

  Goal models are defined as a hierarchy in which a goal is refined to a set of subgoals. This refinement is termed decomposition. Most approaches consider two different types of decomposition. The AND-decomposition of a goal means that all subgoals need to be achieved in order to achieve the super goal. Further, an OR-decomposition can be defined. This decomposition expresses that the super goal is achieved if at least one of its subgoals is achieved. Hereby, different alternatives for the achievement of a goal can be expressed.

- **Operationalization**

  Following the concept of decomposition, goals within a goal model are iteratively refined. To derive functional requirements in terms of concrete actions, the leaf goals are operationalized by actions. These actions define concrete steps that need to be performed to achieve the related leaf goal. There are no certain rules whether a goal is decomposed or operationalized. A decomposition guideline is to operationalize a goal if any action can be assigned to exactly one actor. The operationalization of high level goals leads to complex actions which might make a further refinement necessary in a subsequent design step.

- **Assignment of Responsibilities**

  The third common concept is the assignment of responsibilities. Existing goal modeling approaches provide an actor concept that can be used to specify relevant actors and hierarchies of actor roles. These actors are related to the actions that they are able to execute. Hereby, goal mod-

els also provide additional value for the identification of relevant actors and their contributions to the achievement of the stated goals.

A detailed analysis and discussion of the different approaches is given in Section 3.1. Regardless of the modeling language chosen, the explicit specification of goals serves several purposes [Yue87, DFvL91, vL01].

- **Common understanding of vision and objectives**

  Goal models describe the overall vision and the related objectives. An explicit specification provides the foundation for their communication between the stakeholders involved. This facilitates the discussion and justification of the goals itself as well as inferring more concrete requirements.

- **Guidance for the elicitation process**

  The systematic definition of high-level goals and their step-wise decomposition and operationalization provides a structure for the requirements elicitation and specification process. Specified goals can also be combined with other requirements engineering techniques, like scenario-based requirements elicitation [Sut98, RSA98, MKK05]. Hereby, for each goal a set of scenarios is described to improve the understandability.

- **Consideration of alternatives**

  A goal can be achieved in different ways. The decomposition of a goal into subgoals is suitable for the expression of different possibilities on how to reach the super goal [VL09]. By making these alternatives explicit, their evaluation and the decision-making are supported.

- **Completeness of requirements**

  Following the concept of goal-oriented requirements engineering, each requirement is related to a goal. By using the goal model specification and the defined relation to requirements, it can be evaluated whether all goals are addressed sufficiently or if additional requirements need to be considered. Hereby, goal models support the evaluation of com-

pleteness.

- **Relevance of requirements**

  In addition to completeness, goal models and their relations to requirements can also be applied to check the relevance of requirements. A requirement is relevant if it contributes to the achievement of a goal. Requirements which cannot be related to a goal might be irrelevant and obsolete.

- **Stability of goals**

  Goal models provide the starting point for more concrete requirements or specification artifacts like business process models (cf. Section 2.2.4). The definition of such concrete specification artifacts depends on design decisions on how the elicited goals shall be achieved. Such design decisions can evolve during a project and result in changes of the artifacts. Hereby, goals are considered as more stable pieces of information that provide a reference for managing changes of related requirement artifacts.

### 2.2.4 Business Process Modeling

By using goal-oriented approaches, the system vision can be defined in terms of goals on different levels of abstraction and operationalized actions. To describe how the goals are achieved, the actions need to be composed specifying the desired behavior of the system. This behavior can be represented by business process models. Today, several modeling languages have emerged which can be applied for the specification of business process models.

The industrial de-facto standard for the specification of business processes is the business process modeling notation (BPMN) [OMG11a] which has been developed by the OMG. The primary goal of BPMN is to provide the ability to specify business process models, which are comprehensible and intuitive for business and technical users. To describe business process models, BPMN provides a comprehensive set of language elements. A precise description of all available notation elements is provided in the official OMG BPMN speci-

fication [OMG11a].

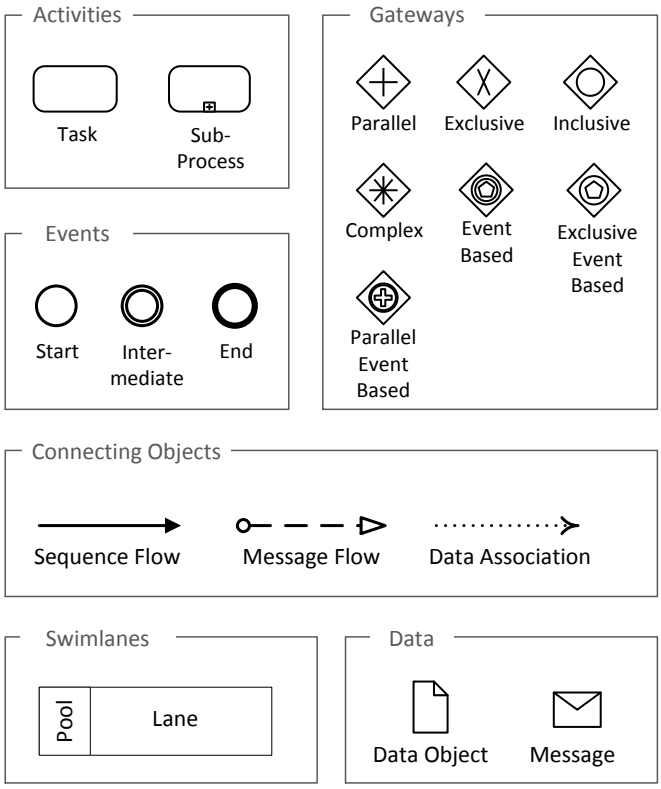


Figure 2.6: Notations for Selected BPMN Language Elements

An overview about selected notation elements is provided in Figure 2.6. The actual processing logic expressed by the business process is defined by activities. These activities can be atomic tasks describing a single step or an aggregated subprocess which encapsulates a separate sequence of tasks or further subprocesses. The application of subprocesses is suitable if an abstracted view of a certain activity is desired in the business process model.

Events are used to model relevant things which happen or can happen during the execution of the business process. Such events usually affect the control flow and are often related to a trigger that causes the event. In BPMN, three types of event are distinguished: *Start*, *Intermediate* and *End*.

Usually, the control flow of a business process cannot be defined in a simple sequence. Very complex scenarios need to consider different execution paths and alternative behaviors. For this purpose, BPMN provides gateways to de-

scribe the divergence and convergence of the control flow.

To define a coherent business process model several connecting objects can be used. The sequence flow is used to represent the actual control flow by defining the execution order of flow elements (activities, events and gateways). In addition to the sequence flow, the exchanged messages can be expressed by the message flow connection. The messages are described by a *message* element. To illustrate the data that is processed or created in the business process, the data association can be used to relate a *data object* to an activity.

Considering different roles and actors who are involved in the execution of a business process, elements can be assigned to *pools*. A *pool* represents a participant in the business process, e.g. a supplier. Hereby, it is possible to express which activities are executed by a certain participant. If a more precise assignment is required, lanes can be used to address a more specific role or function of the participant, e.g. an accounting or a logistics department.

As discussed in Section 2.1, the unified modeling language (UML) provides a set of modeling languages for several purposes [OMG11b]. Business processes can be specified by UML activity diagrams. The notation and visual representation is very similar to the presented BPMN. Sequences of actions are connected by edges that represent the control flow within a business process. A more complex control flow logic can be specified by decision nodes, separating alternative executions paths and synch states for the modeling of parallel action sequences.

While BPMN and UML activity diagrams focus on comprehensibility, the Business Process Execution Language (BPEL) has a stronger focus on executability. BPEL is an OASIS standard [OAS11] that enables the specification of executable processes by relating activities to implemented web services. For this purpose, BPEL provides different language elements to express the invocation of web services but does not provide a visual notation for the expression of the business process structure. Therefore, a BPEL specification is often combined with the modeling capabilities of BPMN.

In addition to the approaches discussed, further modeling languages exist which can be used for business process modeling, like Event-driven Process Chain (EPC) [NR02], Yet Another Workflow Language (YAWL) [VdATH05] or the XML Process Definition Language (XPDL) [Spe12]. Due to their limited

application and distribution in the academic and industrial context, we do not discuss them in detail.

### 2.2.5 Domain and Object Modeling

The analysis and explicit specification of the domain and relevant objects within a domain has been recognized as an important part of the requirements engineering process [Wie03, Poh10]. It is not possible to define requirements properly without considering the application domain of the designed system [MP84, HRH01].

Context models, as proposed in [DeM79], illustrate the data flow between a system and context "endpoints" on a very high level of abstraction. These diagrams are well-suited to identify the boundary between the system and its context. Furthermore, they make external interfaces explicit which the system needs to provide. In early phases, a context diagram also facilitates the identification of relevant stakeholders that need to be considered in the requirements engineering process.

While the context diagrams are often used to provide an externalized view, domain models are used to define objects and their interrelations on a more concrete level [DeM79]. For this purpose, the UML specification provides UML class diagrams [OMG11b]. These diagrams can be used to define relevant object types and to describe the relationships among them. The result is a vocabulary that provides a common understanding and makes communication about relevant objects much more efficient.

## 2.3 Requirements Management

In addition to the engineering of requirements, their efficient management during the software engineering process is an important task as well. In this chapter, we discuss requirements management in detail. First, we discuss the fundamentals of requirements management by presenting different viewpoints and definitions of this term. Further, we describe two essential activities: requirements traceability and change management for requirements.

### 2.3.1 Fundamentals of Requirements Management

Existing definitions and usages of the term "requirements management" vary depending on the perspective, especially with respect to the scope of management activities and the relation between requirements engineering and requirements management. Leffingwell and Widrig consider requirements engineering as an integral part of requirements management [LW00]. A broader scope is proposed by the work in [Sch01]. It defines part of the general product and project management as elements of requirements management.

The requirements engineering framework described in [Poh10] discusses a broader understanding expressed by the term *requirements management* which comprises the following three main aspects.

- **Management and observation of the system context.**

  As discussed in Section 2.2.5, the context of the planned system is relevant for the correctness and validity of the requirements. Hence, the continuous observation of the system context is crucial for the detection of relevant changes that affect the system requirements, e.g. new customer groups that need to be addressed.

- **Management of the RE process.**

  This aspect comprises the management of the single activities in the RE process (elicitation, analysis, specification and validation). Usually, a project defines a time frame and the desired result for each of these ac-

tivities before the RE process is started. Nonetheless, different external events, e.g. changes in the context, can affect the efficiency of the existing plan. During the RE process, requirements management needs to ensure that the current process is appropriate due to the current project context. In order to handle changes, the RE process or certain activities within the process need to be adapted accordingly to create an updated project plan.

- **Management of requirements artifacts**

  During the RE process, various requirements artifacts are created to document the stakeholders intentions. To ensure that these documents are always up-to-date, the requirements artifacts need to be managed as well. In addition, the evolutionary changes of requirements artifacts need to be tracked. Further, the consistency among different artifacts needs to ensured during their evolution. For this purpose, traceability between requirements artifacts needs to be established, which enables the detection of dependencies among them.

In [SK98, Wie03] the management of requirements artifacts is considered as the main challenge in requirements management. This work considers both processes in a more separate way as sketched in Figure 2.7.

As depicted, RE deals with the elicitation, analysis, specification and validation of the stakeholders' requirements. The resulting requirements specification is the transition between requirements engineering and management. Similar to artifact management described in [Poh10], a change management for requirements artifacts is established that handles external changes and their impact on the existing requirements specification.

A prerequisite for preserving consistency within an evolving requirements specification is the explicit specification of traceability relations and dependencies among the different artifacts. In the next section, we introduce the concept of traceability and discuss existing approaches.

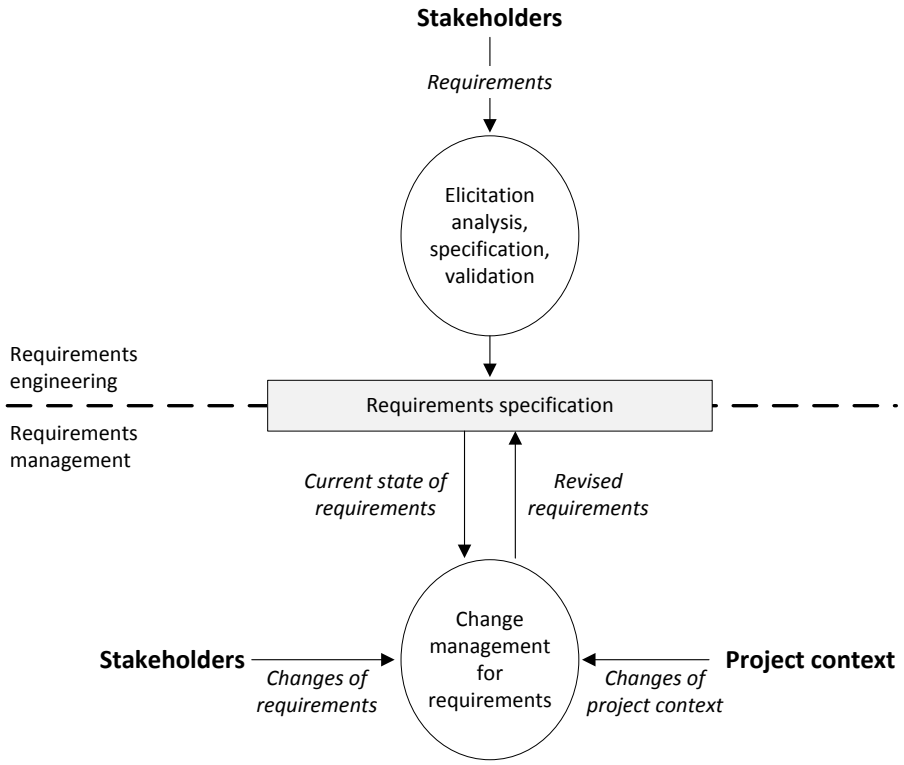


Figure 2.7: Requirements Engineering and Requirements Management ([Wie03])

### 2.3.2 Requirements Traceability

Requirements traceability is an essential aspect of requirements management. In general, traceability denotes the degree to which relationships between software development artifacts can be established [IEE90]. A requirement is traceable if its usage, achievement and consideration through the different phases of the development process is documented [HB91, IEE98]. Requirements traceability can also be established among requirements on different levels of abstraction, e.g. a defined goal and functional requirements which contribute to its achievement.

To define our understanding of the term *Requirements Traceability*, we apply the following definition given by Gotel and Finkelstein:

**Definition 3 (Requirements Traceability)** *([GF94]) Requirements traceability refers to the ability to describe and follow the life of a requirement, in both a forwards and backwards direction (i.e. from its origins, through its development and specification, to its subsequent deployment and use, and through all periods of on-going refinement and iteration in any of these phases).*

According to [Poh96, Ram98, Poh10], requirements traceability supports various activities in the development process.

**Verifiability and acceptance.** The traceability of requirements to design and implementation artifacts provides the foundation for their validation with respect to the requirements. By applying model analysis approaches, design artifacts, e.g. a software architecture specification, can be validated. Further, the user acceptance can be evaluated with respect to the stakeholder requirements [PG96].

**Change management.** Traceability of requirements supports the management of changing requirements by the identification of all artifacts which are related to the evolved requirement and may be affected. Hereby, it eases the effort prediction for the implementation of the desired changes [EH91].

**Quality assurance and maintenance.** While the change management addresses desired changes, traceability also supports quality assurance and maintenance actions. By following the established traceability relations, all artifacts, e.g. parts of the source code that are related to an erroneous artifact, can be identified. Effort estimations for bug fixing can also be done based on this information [Bro87].

**Monitoring of project progress.** Established traceability relationships are useful for the monitoring of the project progress. By estimating the coverage of requirements, i.e. the number of requirements that have already been implemented, the current progress can be determined [Zmu80]. To get a more precise and realistic estimation, requirements can also be weighted, e.g. by the application of use case points [Car05].

Over the past years, several classifications of traceability have been proposed. An overview is sketched in Figure 2.8. First, requirements traceability is differentiated into *pre-requirements specification (pre-RS) traceability* and *post-requirements specification (post-RS) traceability*. Pre-RS deals with the establishment and management of traces during elicitation and analysis of requirements until they are defined in a requirements specification. This means, the specified requirements are usually traced to informal idea sketches or objective description. The lack of maturity and structure of such early documentation makes it very challenging to establish such traces.

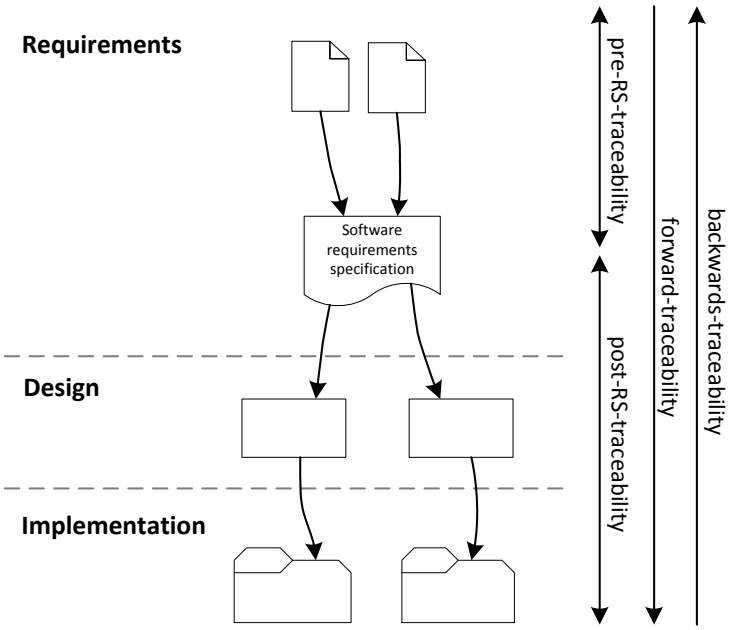


Figure 2.8: Dimensions and Directions of Traceability Links ([WP10])

Post-RS deals with the traceability of the requirements specification to artifacts in subsequent project and development phases. For example, a requirement is traced to a component in the software architecture and the corresponding source code (e.g. a package or a set of classes) that implements the component.

A second classification of traceability is the distinction between *forward- and backwards-traceability*. Forward-traceability of an artifact addresses all artifacts which are derived in subsequent phases from the considered artifact. For example, the forward-traceability link of an architectural design artifact is connected with the source code implementing the designed functional-

ity. Accordingly, backwards-traceability refers to artifacts created in previous phases. Backwards-traceability is useful to identify the sources a certain artifact is related to. An example is the backwards-traceability of a business process to the business goal that it achieves.

Furthermore, traceability can be classified in *horizontal* and *vertical*. In literature, different interpretations of this classification are defined. Following the work in [BBH+96, WP10], a horizontal traceability link connects artifacts created in the same phase or on the same level of abstraction. In [LS96], horizontal traceability considers different elements within the same model. Vertical traceability addresses the connection of different models and in different phases of the software development process.

Traceability links have a certain type that represents the semantics of the relationship. Several classifications of traceability relationship types have been proposed in [RJ01, VKP02, DP05, ELGS05, EAG06]. Aggregated overviews summarizing the proposed approaches from literature are presented in [SZ05, WP10].

We apply these classifications and distinguish eight categories of traceability relationship types. In the following, we consider two artifacts $d_1$ and $d_2$ which are connected by a traceability link and discuss the different relationship types.

- **Dependency links.** The specified artifacts are usually not independent from each other. For example, if a requirement changes, the code block implementing this requirement needs to be adapted as well. To ensure that such changes are done accordingly, dependencies among artifacts need to be defined explicitly. For this purpose, *dependency links* are established. An artifact $d_1$ *depends on* $d_2$, if $d_1$ relies on the existence of $d_2$, or if changes made in $d_2$ have to be reflected in $d_1$ as well. The application of dependency links is particularly useful for the connection of requirement artifacts like use cases and design artifacts to enable an impact analysis [vKPKH02].

- **Refinement links.** During the engineering process, artifacts are specified on different levels of abstraction by following an iterative refinement. To establish traceability through this refinement process, refine-

ment links are used to specify hierarchies of abstraction levels for related artifacts.

- **Evolution links.** Addressing changes in the context, e.g. through new customer requirements, the system and the models describing it are continuously changing in an evolutionary process. Usually, it is desirable to track those changes in terms of an history that documents earlier versions of an evolved artifact, e.g. for the purpose of a roll-back. For this purpose, evolution links can be applied. An evolutionary link between two artifacts $d_1$ and $d_2$ is read as $d_1$ evolved to $d_2$. This means, artifact $d_1$ has been replaced by $d_2$ during the development or by the evolution of the models.

- **Satisfiability links.** This type of relationship expresses that $d_1$ satisfies $d_2$. Satisfiability links are often defined between a requirement and the system component which satisfies this requirement [RJ01].

- **Overlap links.** Two artifacts $d_1$ and $d_2$ overlap, if they refer to common parts, features, functionalities or properties of the system or the relevant domain [SZ05]. For example, in [CAS03] overlap links are established between i* goal models, UML use case and class diagrams. The application for the expression of overlaps between scenario descriptions and use cases is discussed in [vKPKH02].

- **Conflict links.** In many cases, artifacts are not independent from each other. They can describe similar or common aspects of the system (e.g. overlap links) or describe contradicting aspects, e.g. requirements that cannot be achieved within the same system. Such a contradiction between two artifacts $d_1$ and $d_2$ can be marked explicitly by a conflict link. Winkler et.al. [WP10] point out that conflict links also need to include information how the identified conflict can be resolved, e.g. by suitable design alternatives.

- **Rationalization links.** The engineering process includes decision making, e.g. between alternative design solutions to realize a given requirement. Such decisions usually rely on different reasons given by the system or project context. To justify decisions, it is useful to document the

reasons describing the rationale behind these decisions. Rationalization links are associated with artifacts that describe the rationale behind its changes, evolution etc. [Let02].

- **Contribution links.** The creation and authoring of artifacts is done by stakeholders involved in the development project. To define those connections, a contribution link is established between an artifact (contribution) and the related stakeholder (contributor). According to [SZ05], the contributor can be a principal who is responsible for the consequences of the artifact, an author who formulates and organizes the content and the documentor who captures and records the content in the artifact.

### 2.3.3 Change Management for Requirements

During the system life cycle, requirements can evolve according to changes in the context. As part of the requirement management such changes need to be analyzed, evaluated and implemented while considering dependencies on other development artifacts. Changes of requirements can be differentiated into five categories [Poh10]:

- Integration of a new requirement
- Removal of an existing requirement
- Extension of an existing requirement
- Reduction of an existing requirement
- Change of an existing requirement

To support the handling of such change requests, Figure 2.9 sketches an approach for the systematic management of requirement changes.

First, an incoming change request is classified by a change manager or a change control board. An exemplary categorization is the distinction between corrective, adaptive and exceptional changes [Poh10]. Usually, different types of change request are assigned to a certain percentage of the project budget. Hence, their acceptance may be dependent on their classification.

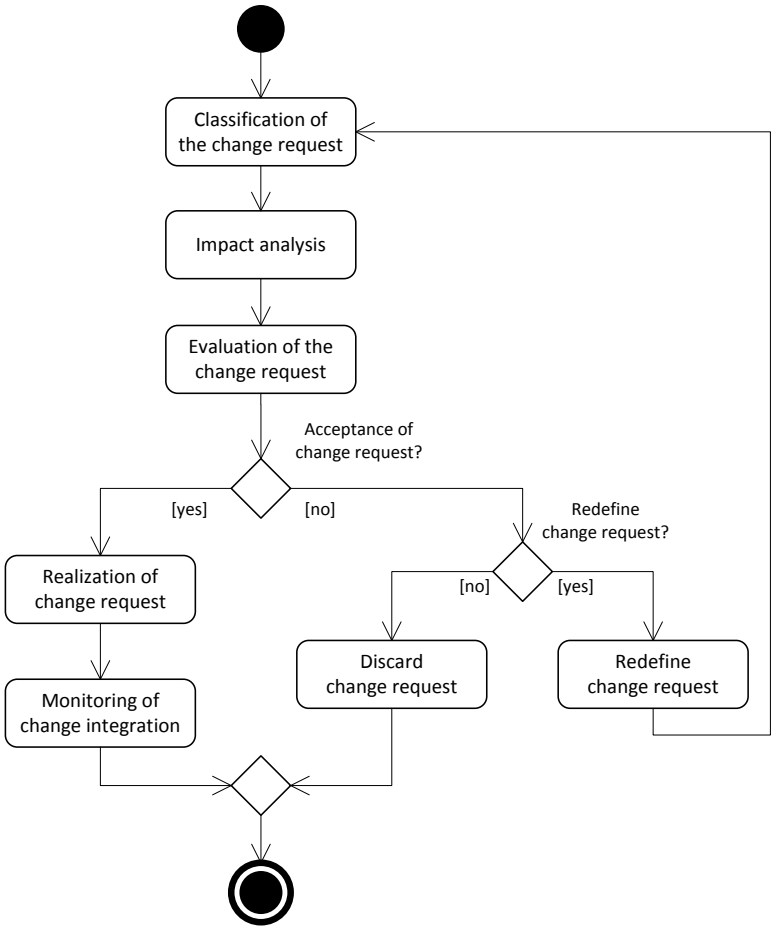


Figure 2.9: Requirements Change Management Process (Extended Version of the Work in [Poh10])

In the next step, the impact of the change request is analyzed and the necessary effort is estimated. The prediction needs to consider the adaption of the requirement artifact, the adaption of all related artifacts (e.g. design or test artifacts) and the actual implementation in the system (code artifact). For this purpose, the traceability links introduced in Section 2.3.2 are applied to identify all artifacts which need to be considered in the impact analysis. The result of this phase is an estimation of the total effort for the change request.

By using the estimated effort and evaluating it against the expected benefits of the change request, a decision about the acceptance of the change request is made. Since not all benefits can be measured in economic units (e.g. 20 percent more sales), the decision-making might require extensive discussions

and negotiations.

If the change request is accepted, it is implemented accordingly. The progress and completion according to time, budget and quality restrictions is monitored continuously. Furthermore, the actual impact and the additional value of the change request are evaluated to gather information for the management of future change requests.

In case the change request is rejected, it needs to be decided if it is discarded completely or if a redefinition is required. The redefined change request is used to restart the requirements change management process.

## 2.4 Quality of Requirements

Several studies discuss the impact and importance of requirements engineering in industrial projects. About 40% - 60% of all defects in a software product can be related to an error in the requirements engineering [LWE01, Wie03]. From these insights, we can infer that many errors can be prevented during the requirements engineering phase, e.g. by a suitable quality assurance of the requirements specification. The removal of errors in an early project phase can reduce the overall effort significantly. According to [Boe88], the effort is 50 - 200 times cheaper while McConnell argues for an effort reduction of 10 - 100 times.

Addressing the topic of requirements quality, this section discusses existing impact factors and quality properties which can be used to improve and validate the quality of requirements specifications. Critical success factors for the requirements engineering phase are described in Section 2.4.1. Concrete attributes and properties which can be applied to validate the overall quality of the requirements specifications are discussed in Section 2.4.2.

### 2.4.1 Success Factors for Requirements Engineering

According to the study results described in the Standish Group Report [The06], more than 50% of the key factors for the success of software projects can be related to the requirements engineering phase. To illustrate such success fac-

tors, the Requirements Engineering Reference Model (REM) developed by the TU München and Siemens provides an overview about critical success factors [GBB+06, BGK+07].

An important factor is the integration of domain experts during the different requirements engineering phases (cf. Section 2.2.2). Their availability and support especially in early steps, like the context definition, is crucial for the success of this phase. While the contribution of domain experts often focuses on functional requirements, non-functional requirements need to be elicited and specified as well to ensure their consideration. For this purpose, technical knowledge needs to be acquired which requires a qualified software architect that can define non-functional requirements based on his experience and know-how.

In Section 2.3 we discussed the central role and importance of requirements management. To support the different management activities, an efficient requirements management process needs to be established. Existing approaches like the Capability Maturity Model (CMMI) can be applied [Pau93]. Furthermore, the requirements engineering process needs to provide the required level of flexibility and the ability to scale with the complexity of the project. To improve the applicability of the requirements engineering and management processes, an appropriate tool support is required. Usually, the integration and adaption of the tooling is necessary to match the project requirements.

### 2.4.2 Quality Attributes for Requirements Specifications

The quality of a requirements specification can be validated in different dimensions. Existing work proposes several quality attributes [DOJ+93, Wil97, IEE98, Wie03, KEB08, Tjo08, Poh10, GFL+13]. In the following, we describe the most important and relevant quality properties. The requirements artifacts themselves and a requirements specification need to be:

- **Complete:** A requirement is complete if it considers all information that is relevant for all stakeholders. Furthermore, it needs to fulfill all rules and guidelines defined for the specification of requirements given by the application of a given standard like [IEE98]. Such standards and guidelines can also define formalities which need to be considered (e.g.

a defined pattern for the requirements identifier). According to [Poh10], a requirements artifact is complete if it has been captured completely and a requirements specification is complete if it comprises all relevant requirements.

- **Traceable:** A requirement artifact is traceable if every requirement can be traced to its source and its impact and relation to artifacts in subsequent development phases are traceable. A detailed discussion of requirements traceability has been given in Section 2.3.2.

- **Correct:** A requirement specification is considered as correct, if every requirement defines an aspect that is required to develop the system [DOJ+93, IEE98]. The correctness of a requirement is confirmed by the stakeholders involved who decide whether the requirements describe a system that matches their needs [Wil97, Poh10, GFL+13].

- **Unambiguous:** A requirements specification is unambiguous, if every documented requirement has only one possible interpretation [DOJ+93, Poh10]. In [IEE98] it is also stated that relevant characteristics of the domain or the final product need to be defined by unique terms.

- **Understandable:** A requirements specification is considered as understandable, if its content is easy to comprehend by its reader [Poh10]. The understandability of requirements can be improved by an appropriate structure and suitable guidelines. Furthermore, a concise description also improves their comprehensibility [DOJ+93].

- **Consistent:** Pohl [Poh10] considers a requirements specification as consistent if it has no contradictions. Davis et.al. [DOJ+93] explicitly distinguishes between internal and external consistency. Internal consistency means that the different parts of a requirement, e.g. its single statements, do not conflict with each other. A requirements specification is considered as externally consistent if no requirements within this specification are conflicting.

- **Verifiable:** A requirements specification is verifiable if for each requirement it can be checked whether it has been considered appropriately in

the implemented system. In [DOJ+93, IEE98] a requirement is defined as verifiable if a finite and cost-effective process exists to check whether the software meets this requirement.

- **Modifiable:** A modifiable requirements specification is defined in a structure or follows a template that makes it possible to change and modify the requirements in an easy and traceable manner. This comprises the tracking of changes and the management of different versions. Modifiability is important for two reasons: (1) the requirements continuously evolve and (2) the requirements might contain weaknesses which need to be fixed [DOJ+93, IEE98].

- **Atomic:** An atomic requirement describes a single and coherent fact. In contrast, a requirement is not atomic if it expresses different isolated or loosely coupled facts, which can be separated into several requirements [Poh10].

- **Right level of abstraction:** Addressing the different groups of stakeholders, a requirements specification can describe the requirements of a system on different levels of abstraction [Wil97]. The right level of abstraction cannot be measured in a clear-cut way. In [DOJ+93] the right abstraction level is characterized as follows: *"A requirements specification should be specific enough that any system built that satisfies the specified requirements satisfies all user needs, and abstract enough that all systems that satisfy all user needs also satisfy the requirements."*

## 2.5 Running Example - Online Book Shop

Throughout this thesis, we use a running example to illustrate the concepts introduced above. As a scenario, we consider an online book shop that is to be developed. As part of the work presented in this thesis we focus on the early requirements engineering phases and discuss which goals are intended to be achieved in this scenario and the business process that is required to achieve these goals. In the following chapters, we will show how the different models are created and refined in an iterative manner. A more comprehensive description of this scenario is provided in [Pos12].

# Chapter 3

# Business Goal Modeling

In this chapter, we present an approach for the specification of business goal models. In Section 3.1 we identify requirements for our approach and analyze related work with respect to these requirements. Applying the evaluation results, Section 3.2 presents an overview about our modeling approach. In Section 3.3, we describe the definition of relevant business object types in the business context as well as the specification of business goals and their systematic operationalization to concrete actions. The validation of action composability is addressed in Section 3.5. Finally, Section 3.6 concludes this chapter and summarizes its results. This chapter is partially based on the earlier publication [NGPE13b].

## 3.1 Requirements and Related Work

In this section, we analyze the concept of business goal modeling and elicit requirements for a suitable approach. Existing approaches are evaluated with respect to these requirements and the evaluation results are discussed. Based on the gathered results we select an existing modeling language as a foundation for our approach.

### 3.1.1 Requirements Definition

Goal-oriented requirements engineering is widely used for the elicitation and specification of requirements in early project phases. These approaches aim for the definition of high-level goals that need to be achieved [GPW06] and their explicit specification in goal models. Hereby, goals are systematically decomposed into subgoals and finally operationalized to actions. These actions define concrete functional requirements that describe how a goal is achieved.

Business goal models apply the generic concept of goal-oriented requirements engineering to a specific domain. Addressing the domain-specific context, we investigate whether existing approaches support the specification of business goals and the subsequent derivation of business processes in a sufficient way. For this purpose, we analyze the challenges of business goal modeling and state requirements for the evaluation of existing approaches.

The purpose and importance of business goals becomes obvious from the definition given in [BHL+07]. It emphasizes the direct relation between business goal achievements and the success of an enterprise or organization.

> *"What an enterprise needs to accomplish in order to maintain or achieve business success."* [BHL+07]

Figure 3.1 sketches an overview about the challenges in business goal modeling. A prerequisite for the consideration, monitoring and tracing of goals is their explicit specification in a business goal model. Such models capture and specify business goals which are relevant in a given context, e.g. for a specific enterprise. Usually, they are specified by business analysts who are familiar with requirement engineering techniques and specification languages.

As depicted, the business goal model provides the foundation for subsequent specification steps, especially the derivation of business processes. To ensure that the goals are considered appropriately, the specified business goals need to be in line with the actual goals raised by the enterprise executives. Therefore, the business goal model needs to provide a communication basis that is understandable for the modeling specialist (business analysts) and the enterprise executives (**C1**).

An approved specification of business goals provides a mutual agreement

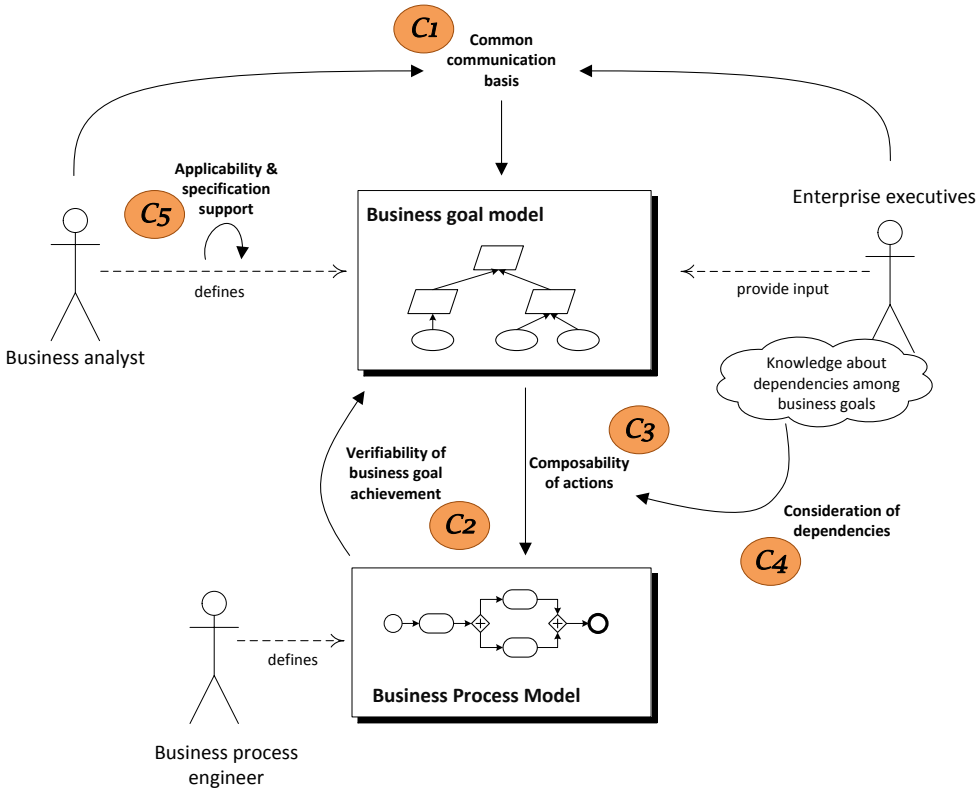


Figure 3.1: Challenges of Business Goal Modeling

concerning the desired achievements. Business processes describe an operationalized view on how the defined business goals can be achieved. To enable the derivation of business processes that sufficiently satisfy the stated business goals it needs to be verifiable whether a business goal is achieved (**C2**). Since a natural language specification is not suitable for an automated validation, the achievement of business goals needs to be expressed by a verifiable condition.

The identification and specification of the required actions and the definition of their contribution to the achievement of a certain goal is part of the business goal modeling. To define a business process that achieves the business goals, the business process engineer composes these actions. Figure 3.2 illustrates the achievement of business goal $g_1$ by the composition of actions $a_1$ and $a_2$. For the derivation of a valid composition it needs to be ensured that input $I_2$ of action $a_2$ matches output $O_1$ of action $a_1$ (composability).

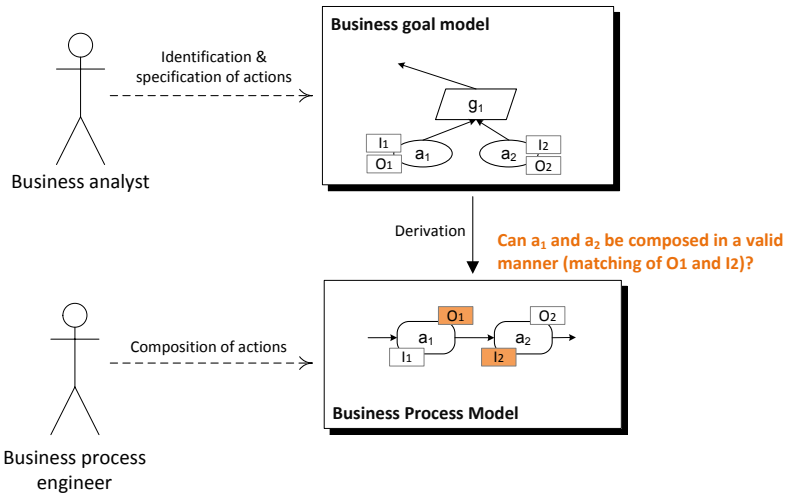


Figure 3.2: Composability of Actions

To identify valid compositions and to verify the composability of two actions, inputs and outputs representing the effect of the action execution need to be specified explicitly in the goal model. In addition to the verifiability of the composability, it needs to be decidable whether the execution of the composed actions leads to the achievement of the related business goal (**C3**).

During the derivation of a business process model, the defined actions are composed in a specific order. The resulting control flow defines an explicit order for the action execution. A valid order does not only rely on input and output matching of the actions but also needs to consider dependencies among business goals. As an example, we consider a strategic decision whether the customer has to pay the ordered products before delivery (to ensure that payment is received) or whether it is desired to present the products to the customer and to allow a later payment (to maximize the sales of products). Awareness about such dependencies is often scattered across the know-how of different executives. In order to ensure that such dependencies are considered in the composition, they need to be specified in the business goal model (**C4**).

Capturing the described aspects (**C1-C4**), business analysts elicit and specify the required information in a business goal model. A lack of quality in this specification, e.g. missing dependencies among goals, can also affect the quality of the derived business process. For this purpose, the business goal

modeling approach needs to be easy-to-use and to provide a high applicability through suitable methodical guidance and specification support (**C5**).

In summary, we identified five core challenges for business goal modeling. Based on the challenges identified, we define concrete business goal requirements (BGR) that need to be fulfilled by a suitable modeling approach.

- **C1** - Common communication basis
- **C2** - Verifiability of business goal achievement
- **C3** - Composability of actions
- **C4** - Consideration of dependencies
- **C5** - Applicability and specification support

#### C1 - Common communication basis

To provide a common basis of communication the business goal models need to be understandable for the involved actors who specify these models (business analysts) and for enterprise executives who articulate the goals. Therefore, we infer the following requirements for our modeling approach:

- **BGR-01** Understandability of the business goal model

  The modeling approach needs to support the specification and description of business goals and actions in an understandable way. This means, it should provide informal specification techniques which are understandable for executives without a technical background.

- **BGR-02** Different abstraction levels

  Depending on the role and background of the executives involved the stated business goals can address concerns of the overall enterprise or more specific aspects, e.g. the fulfillment of a customer order. Hence, the business goal modeling approach needs to support the definition of business goals on different levels of abstraction.

**C2 - Verifiability of business goal achievement**

Business goals are iteratively refined to actions which are composed in a business process contributing to the achievement of the related goals. In order to evaluate the achievement of a business goal, conditions for the achievement need to be verifiable. For our modeling approach, we infer the following requirements:

- **BGR-03** - Conditions for the achievement of business goals

  To enable the verification of business goals a description in natural language is not sufficient. The modeling approach needs to provide language capabilities for the precise specification of verifiable conditions under which a business goal is achieved.

- **BGR-04** - Business context modeling

  The conditions for the achievement of goals are described based on objects in the context. These conditions are defined over object attributes and attribute values. Hence, the approach needs to provide a model of relevant objects in the business context for which these conditions can be defined. To consider interrelations among the different objects they should be specified in an integrated business context model.

**C3 - Composability of actions**

The derivation of business processes from a given business goal model comprises the composition of actions to achieve the related business goals. To enable this composition, the relevant actions need to be described in a precise way. Further, it needs to be ensured that they can be composed in a valid manner. Hence, we infer the following requirements:

- **BGR-05** Precise action specification

  The execution of an action has an effect on an object in the given context. To illustrate this effect, it needs to be described how the object is changed by the action execution. Accordingly, input and output as well as pre- and postconditions of an action need to be expressible.

- **BGR-06** Validation of composability

  By composing the identified actions, a control flow is defined by connecting inputs and outputs. To ensure that inputs and outputs of the actions can be connected in a valid manner, the modeling approach needs to support the validation of action composability.

#### C4 - Consideration of dependencies

The business goals are not independent from each other. To consider such dependencies in the business process composition, their explicit specification in the business goal model is required. We state the following requirements for our approach:

- **BGR-07** Hierarchical decomposition relations

  Following the decomposition logic, the achievement of a business goal is dependent on its subgoals indicated by the decomposition relations. It needs to be expressable in which way the subgoals contribute to the achievement of a super goal, e.g. the joint achievement of all subgoals.

- **BGR-08** Temporal order dependencies

  The approach needs to provide modeling capabilities for the explicit specification of temporal dependencies between business goals expressing the order in which they need to be achieved.

#### C5 - Applicability and specification support

To make the business goal modeling applicable for business analysts and to improve the quality of the requirements specification, methodical guidance and support is required. Accordingly, we infer the following requirement for our business goal modeling approach:

- **BGR-09** Integrated specification process

  The business goal modeling approach needs to provide an integrated specification process that comprises the systematic definition of the re-

quired models.

An overview about the resulting requirements is given in Table 3.1. In the following, we use the assigned IDs to refer to these requirements.

| ID | Requirement |
|---|---|
| **BGR-01** | Understandability of the business goal model |
| **BGR-02** | Different abstraction levels |
| **BGR-03** | Conditions for the achievement of business goals |
| **BGR-04** | Business context model |
| **BGR-05** | Precise action specification |
| **BGR-06** | Validation of composability |
| **BGR-07** | Hierarchical decomposition relations |
| **BGR-08** | Temporal order dependencies |
| **BGR-09** | Integrated specification process |

Table 3.1: Requirements for Business Goal Modeling

### 3.1.2 Evaluation Results and Discussion

In the previous section, we elicited requirements for business goal modeling. Based on these requirements we evaluate the suitability of existing approaches. In the following, we discuss existing work in the areas of KAOS-related approaches, agent-oriented techniques and enterprise architecture modeling.

Goal models are used for the elicitation and specification of requirements in an early stage of the software engineering process. KAOS provides a notation for the goal-directed specification of requirements [DFvL91, DvLF93]. KAOS supports the specification of goal models by modeling capabilities for the expression of goals and actions as well as the decomposition by AND-/OR-branches. Temporal dependencies between goals are not considered in KAOS. Goals and actions can be specified by a comprehensive set of predefined annotations including the definition of conditions for the achievement of goals. Relevant context elements can be modeled by using a domain model, but KAOS does not provide a meta-model for the specification of these models.

In the domain of agent-oriented software development, goal models are well-known as a requirements modeling technique. The most common notations are Tropos and i*. Tropos has been introduced in [BPG+04, GKMP04]. In contrast to KAOS, the central concept of Tropos models is an *Actor*. These *Actors* are modeled as entities with strategic goals. Goals are refined by AND-/OR-decompositions. In addition, Tropos provides simple dependency modeling by the definition of contribution links. These links express that a goal contributes positively or negatively to the achievement of another goal. To enable formal analysis techniques, Tropos provides formal goal definitions as goal annotations [GMS05]. An extended version of Tropos presented in [ADG09] supports the definition of variation points that enable the definition of conditions for OR-decomposition branches. [KPR04b] proposes an extended Tropos notation with the ability to express dependencies between actors in terms of conditions that need to be held or achieved.

The actor-centric concept of the i* modeling framework [Yu93, Yu97] is similar to Tropos. i* does not support AND-/OR-decompositions with the same expressiveness as KAOS or Tropos. In i* models, *Task-Decomposition-Links* refine tasks and specify mandatory elements for the achievement of a task. By using *Means-End Links*, goals can be related to different tasks comparable to an OR-decomposition. This relationship cannot be applied to the expression of dependencies among goals. In [LY04], i* models are applied to the modeling of strategic business objectives and their operationalization. In this approach an extended concept for logical dependencies is presented. [OSB11a] extends the goal modeling with an additional preference graph that expresses preferences for alternative goals. In [LLJM11], a framework for goal-based behavioral customization is presented. To constrain the order in which goals have to be achieved, precedence links are introduced. These links express that one goal has to be achieved before another.

Existing work in [EQJvS11] provides an enterprise architecture modeling approach for business goals and requirements. This approach distinguishes requirements domain and stakeholder domain modeling. The requirements domain focuses on the definition of goals, requirements and use cases. While the specification of goals is well-supported, the operationalization is only partially fulfilled, since the definition of concrete actions is not considered. The stakeholder domain enables the representation of an actor-centric domain modeling but does not include business object modeling in terms of non-actor

| Approach | BGR-01 | BGR-02 | BGR-03 | BGR-04 | BGR-05 | BGR-06 | BGR-07 | BGR-08 | BGR-09 |
|---|---|---|---|---|---|---|---|---|---|
| [DFvL91, DvLF93] | + | + | + | + | + | - | o | - | - |
| [BPG+04, GKMP04] | + | o | + | o | + | - | o | - | - |
| [GMS05] | + | o | + | o | + | - | o | - | - |
| [ADG09] | + | o | + | o | + | - | o | - | - |
| [KPR04b] | + | + | + | o | + | - | o | - | - |
| [Yu93, Yu97] | + | + | + | - | + | - | o | - | - |
| [LY04] | + | + | + | o | + | - | o | - | - |
| [OSB11a] | + | o | + | o | + | - | o | - | - |
| [LLJM11] | + | o | + | o | + | - | - | o | - |
| [EQJvS11] | o | + | - | o | - | - | + | - | - |

Table 3.2: Evaluation Results for Business Goal Modeling

elements like artifacts.

The evaluation overview depicted in Table 3.2 shows that none of the existing approaches sufficiently fulfills all stated requirements. To express the degree of fulfillment, we use a metric with three values: supported (+), partially supported (o), not supported (-).

The provision of a common communication basis by understandable goal specification and their definition on different levels of abstraction is supported by most of the investigated approaches. Conditions for the achievement of goals are also specifiable based on provided business context models. Basic concepts for hierarchical decomposition relations, like the refinement of goals by AND-/OR-decompositions are provided by most of the existing goal modeling techniques. The modeling of temporal constraints between goals is not supported sufficiently by existing work. Another drawback of the evaluated approaches is the lack of consideration of guidance and methods supporting a consistent goal specification. None of the approaches provides an integrated specification process that considers the systematic decomposition of goals as well as the iterative modeling of required business objects including the validation of the composability.

Based on the results of the evaluation, we pick the KAOS approach [DFvL91, DvLF93] as the foundation for our business goal modeling approach. In the next section, we perform a detailed analysis of KAOS identifying deficits with

respect to the stated requirements.

### 3.1.3 Detailed Analysis of KAOS

In this section, we investigate which concepts of KAOS can be adopted with respect to the elicited requirements. Further, we analyze the deficits and point out required extensions that need to be provided by our approach.

#### Support by existing KAOS concepts

The evaluation results depicted in Table 3.2 show that KAOS fulfills the stated requirements partially. Figure 3.3 illustrates an example of a KAOS goal model. As depicted, goals can be defined on different levels of abstractions ($g_{1-5}$). Following the decomposition logic, a goal is achieved by the achievement of its subgoals. The hierarchical relations can be specified with the logical operators AND/OR. An AND-decomposition defines that each subgoal needs to be fulfilled in order to achieve the super goal. An OR-decomposition expresses that at least one subgoal needs to be achieved to fulfill the super goal. Using operationalization [AKRU09b, VL09], actions are identified that need to be executed in order to achieve the related goals. Following the semantics of the operationalization link, each assigned action needs to be performed in order to achieve the related goal. Considering our example the actions $a_2$ and $a_3$ need to be performed to achieve goal $g_4$.

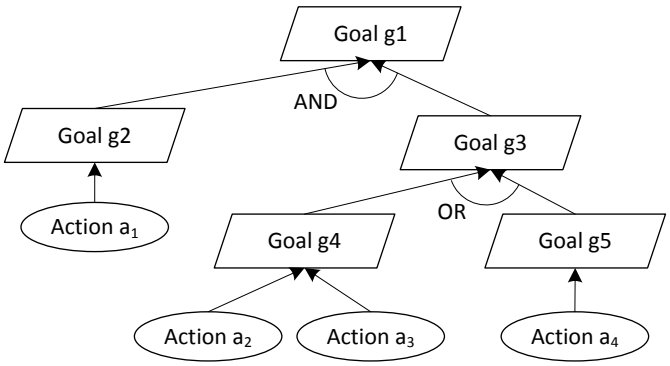


Figure 3.3: KAOS Goal Model

The goals in KAOS can be defined by referring to elements in the business

| | |
|---|---|
| **Action** | Deliver to customer |
| **Description** | The ordered books are delivered to the customer. |
| **Input** | b:BookOrder |
| **Output** | b:BookOrder |
| **PreCondition** | b.inDelivery |
| **PostCondition** | b.Delivered |

Table 3.3: Definition of Action *Deliver to customer*

context. Actions are defined by input and output parameters that express which types of objects are processed as input or returned as result. The actual transformation of the objects triggered by the execution of the action is described by pre- and postconditions. An exemplary specification of an action is illustrated in Table 3.3. Action *Deliver to customer* processes objects of type *BookOrder*. The precondition expresses that the object needs to be in state *inDelivery* and after execution of the action, the state of the object is changed to *Delivered*.

To summarize, the modeling capabilities of KAOS sufficiently fulfill the stated requirements GMR-01 - GMR-05 with respect to the specification of goals and actions. Requirement GMR-07 (hierarchical decomposition relations) is partially fulfilled by providing AND/OR decompositions.

#### Deficits of KAOS

As discussed, KAOS enables the definition of OR decompositions to express alternatives without any further restriction of the goal achievement. The need for a more precise distinction of the OR dependency is motivated by using the exemplary excerpt depicted in Figure 3.4. It considers an inclusive OR-decomposition of the goal *Payment received*. Following the semantics of inclusive OR decompositions, this goal is fulfilled by either achieving the goal *Payment via credit card* or the goal *Payment via bank transfer* or by achieving both goals. It is obvious that the joint achievement of both goals is not desired, since the customer will either pay by bank transfer or by credit card and not in both ways.

With respect to the definition of logical relations in hierarchical decompositions, we identify the deficit: **Impreciseness of logical relations in hierarchi-**

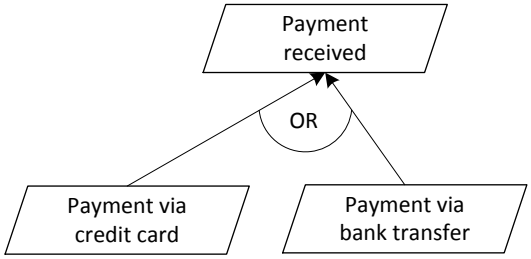


Figure 3.4: Exemplary OR-decomposition

**cal decompositions**.

Requirement BGR-08 describes the need for the expression of temporal dependencies among goals. To illustrate the shortcomings of KAOS for the explicit modeling of these dependencies we consider two goals *Payment received* and *Books delivered* as depicted in Figure 3.5. A temporal order of these business goals implies the decision whether ordered books need to be paid before they are delivered.

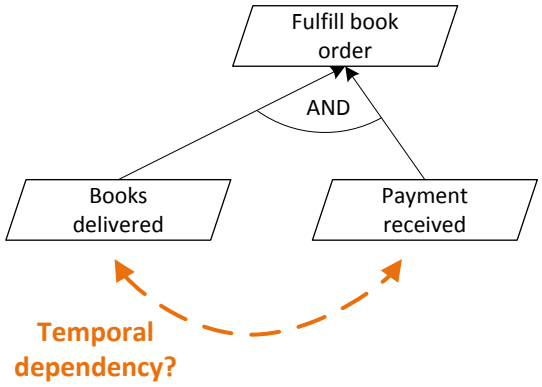


Figure 3.5: Lacking Specification of Temporal Dependencies among Business Goals

KAOS does not provide any modeling capabilities to express such temporal dependencies among goals. As a consequence, the knowledge about temporal dependencies may not be considered in subsequent development phases, e.g. the composition of business process models.

We identify the deficit: **Lacking consideration of temporal goal dependen-**

**cies**

In KAOS, actions can be specified by pre- and postconditions. An example is depicted in Figure 3.6. According to the semantics of operationalization links, both actions *Authorize credit card payment* and *Receive credit card payment* need to be performed to achieve the goal *Payment via credit card*. This means, they are composed in a certain execution order. Hereby, the pre- and postconditions need to match to enable a valid composition.

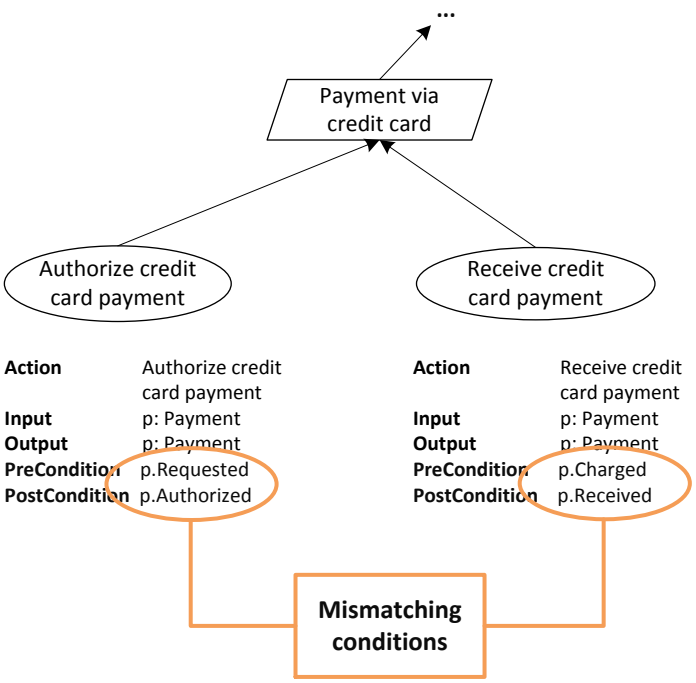


Figure 3.6: Mismatching of Pre- and Postconditions in the Action Specification

Existing capabilities in KAOS do not provide any kind of guidance or validation for the specification of actions. The depicted example in Figure 3.6 illustrates a mismatching. Based on the specified pre- and postcondition no valid order can be defined in which the precondition of one action matches the postcondition of the other action. To prevent such modeling inconsistencies an appropriate approach for the validation is required that is not provided by KAOS.

We identify the deficit: **Lacking consideration of composability in action**

**specification**

As discussed, KAOS provides modeling capabilities to specify goals by formalized conditions. These conditions are specified for goals on different levels of abstraction. Figure 3.7 sketches an exemplary excerpt from a goal model. The achievement of goal *Receipt sent* is defined by objects of type *Receipt*. The subgoals *Printed receipt sent* and *Electronic receipt sent* are defined by different object types *PrintedReceipt* and *ElectronicReceipt*. Considering this example, goal *Receipt sent* is fulfilled by achieving subgoal *Printed receipt sent* or *Electronic receipt sent*. The specification does neither reflect this relation nor does it support the validation since they are defined based on different object types. Hence, it is not decidable in which way the state *r.Sent* is related to *pr.Mailed* or *er.EMailed*.

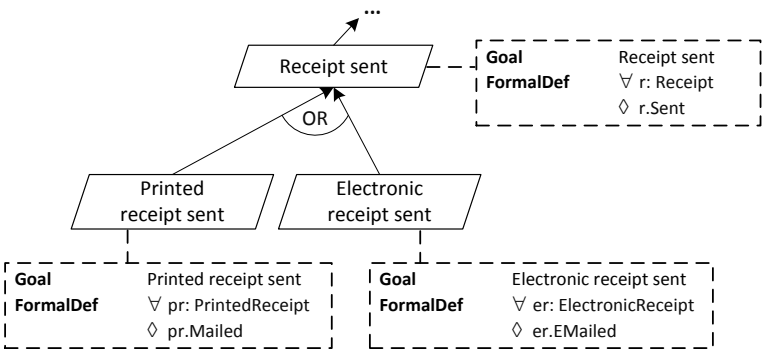


Figure 3.7: The Achievement of the Super Goal by the fulfillment of its Subgoals is not verifiable

Although KAOS provides the required modeling capabilities, it does not provide a methodical or analytical approach to ensure a consistent formalization.

We identify the deficit: **Lacking verifiability of achievement conditions in goal hierarchies**.

To improve the applicability of a business goal modeling approach a methodical guidance for business analysts supports the improvement of the overall specification quality. Such an approach needs to cover all modeling capabilities as defined in the requirements in Section 3.1.1. Since KAOS does not consider all aspects, the modeling support provided by KAOS is not sufficient.

We identify the deficit: **Lack of methodical support and guidance**

Summarizing the deficits of existing KAOS capabilities, our approach needs to extend the existing KAOS goal modeling technique by additional features. In the next section, we provide an overview about our approach and the contributions it makes.

## 3.2 Approach Overview

In the previous section, we elicited requirements for a suitable business goal modeling approach and evaluated existing techniques. Based on the selected KAOS approach we identified deficits with respect to the stated requirements. This section gives an overview about our business goal modeling that addresses these deficits. Figure 3.8 depicts an overview about the deficits and contributions of our approach to resolving these.

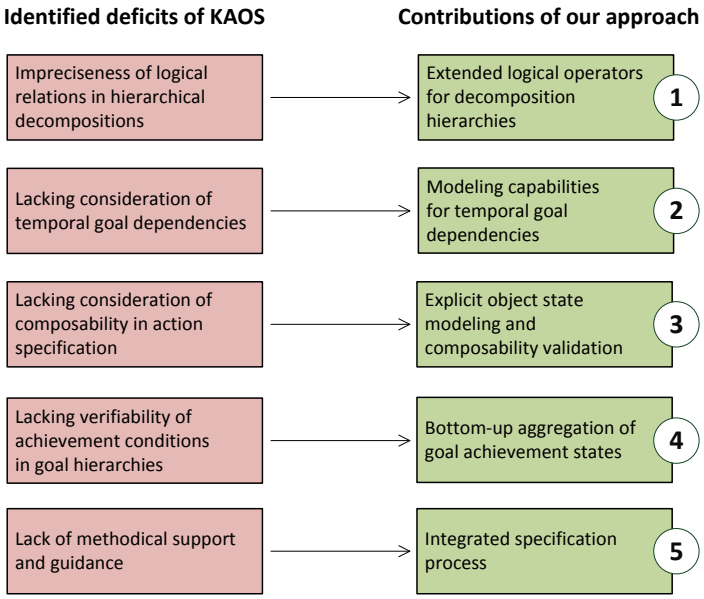


Figure 3.8: Contributions of our Business Goal Modeling Approach

To overcome the impreciseness of logical operators in hierarchical relations, we introduce a new logical operator for the specification of links in decomposition hierarchies (1). The missing consideration of temporal dependencies

between goals has been pointed out as a second major modeling drawback. Our approach provides annotation capabilities that enable the explicit definition of temporal dependencies in the business goal model (2). To support and validate the composability of actions, we introduce the explicit modeling of object states in object life cycle models (3). By using these models the composability of actions can be validated.

The modeling of verifiable achievement conditions for goals on different levels of abstraction is not considered and supported in KAOS. We propose a technique that systematically aggregates goal achievement conditions enabling their verifiability (4). The lack of guidance and methodical support in the existing KAOS approach is addressed by an integrated specification process (5).

An overview about the different steps of our approach, the related contributions and the resulting structure of this chapter is depicted in Figure 3.9.

The first step involves executives introducing their goals and business analysts that specify them in a systematic manner. To provide a common understanding and a foundation for the discussion and justification of business requirements, two models are created in this step. Elicited business goals themselves as well as dependencies and relationships among them are specified in the *business goal model*. The *business object type model* specifies relevant elements in the business domain in terms of business object types including their attributes and interrelations. Details about this step are given in Section 3.3.

In the second step, the specification of business goals and actions is extended by more precise definitions. The actions are defined by pre- and postconditions that need to hold before and after their execution. These conditions are expressed by states of business object types. Each business goal is precisely defined by a state that needs to be achieved in order to fulfill the goal sufficiently. Achievement states for business goals are systematically defined by the aggregation according to the hierarchical decomposition relations in the business goal model. The specification of goal achievement states and their aggregation in a bottom-up manner are described in Section 3.4.

Based on the precise action specification, the composability is validated in order to provide the foundation for the derivation of business process models. Valid states of business object types as well as transitions among them are defined in an object life cycle model. For each object type a valid life cycle is

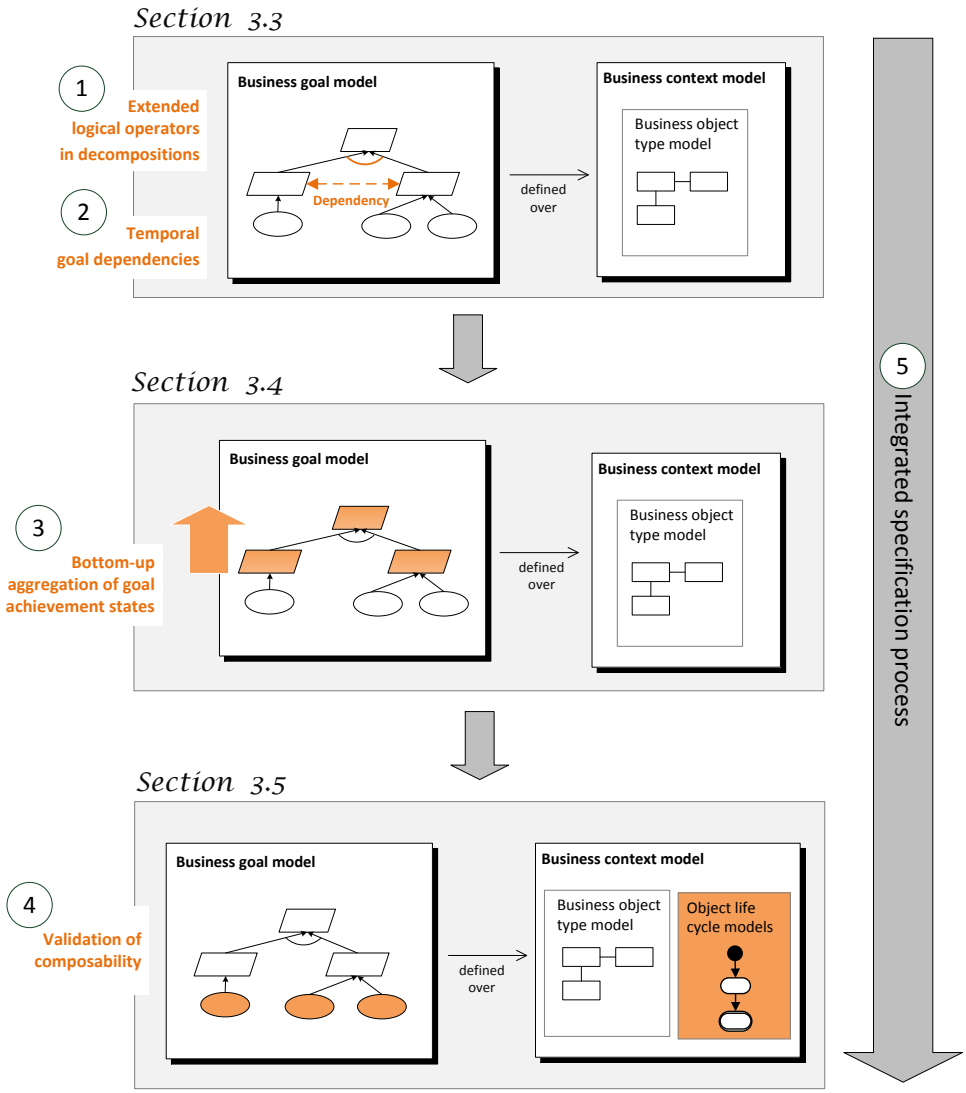


Figure 3.9: Business Goal Modeling Approach

generated. Section 3.5 describes how these models are used to validate the composability of the actions.

## 3.3 Business Object Type and Business Goal Specification

In the first step, we specify relevant business object types in the given domain and elicit and specify the business goals. For this purpose, we introduce modeling elements for the definition of business object types and business goal models. For this purpose, we leverage the KAOS goal modeling language [DvLF93] as the foundation for our approach and extend it by additional modeling capabilities raised from the requirements analysis described in Section 3.1.

### 3.3.1 Business Object Type Model

Business object type models define relevant elements in the business context. This modeling comprises the description of relevant business objects in terms of an object type definition as well as the consideration of relations among them. Hereby, the object type model can also be used as a glossary for the given business domain. To precisely scope and state our understanding we define a business object type model as follows.

**Definition 4 (Business object type model)** *A business object type model* $\mathcal{C} = (O, B)$ *consists of a finite set* $O$ *and a binary relation* $B \subseteq (O \times O)$*, such that*

- $O$ *is a finite set of business object types*
- $B$ *is a finite set of business object type relations*

The explicit specification of business object types in a separate model provides additional value by fulfilling several purposes [Cas97]. Defining a common vocabulary facilitates the communication between people with different backgrounds and roles, e.g. enterprise executives, business analysts and business process designers [Wes99]. The specification of business object types enables the precise definition of them and their characteristics. An explicit model facilitates the tracing and tracking of evolutionary changes [HIR02, ZCPT05].

To support the specification of these models, we provide a formalized and precise language definition. First, we identify relevant aspects of business objects in order to identify the concepts that need to be considered. A business object type represents an element in the business domain which is defined by its business name and definition, attributes, relationships [Bur95, Cas97]. Each business object type is identified by a unique name and characterized by a description. The names and descriptions can be used to create a glossary describing the relevant elements in the business domain. A more comprehensive specification of the object type characteristics is expressed by a set of attributes [ZBP+12]. Accordingly, these attributes model resources that represent information objects [CST05, BGH+07, Hul08], e.g. the business object type *Receipt* can be described by attributes like *Receipt number* or *Receipt date*.

The described business domain is continuously changing, which affects the contained business objects. As part of these changes the states of affected business objects change as well. For example, a business object *bo* of type *BookOrder* can change from a state *Ordered* to a state *Delivered*. To capture valid states for a business object types these states are defined in the business object type definition.

A business object type can represent different kinds of elements. Such kinds can be artifacts like receipts, immaterial concepts like an order as well as actors who are relevant or active in the considered domain. An actor can represent a human actor as well as a system actor, like an information systems [CST05]. By assigning different roles, the actor concept can be defined more precisely [ZCPT05].

Usually, the business object types are not independent from each other. They are interrelated according to their interactions and dependencies [Gil00]. For example, we consider two business object types *Payment* and *Receipt*. The receipt is sent to the customer to confirm a received payment. Both business object types are related to each other by a relationship labeled *confirms*. To model such dependencies, different types of business object type relations need to be expressible in an object type definition.

Considering the described characteristics of business objects, we introduce a modeling language for the specification of business object types. The presented specification language provides a set of predefined elements for the definition of business object types, but also considers the applicability to a

| | |
|---|---|
| **Business object type** | BookOrder |
| **Description** | A BookOrder describes an order of books in the web shop. |
| **Attributes** | orderId, orderDate |
| **States** | Selected, Ordered, SupplOrdered, Stocked, Supplied, Available, InDelivery, Delivered |

Table 3.4: Specification of Business Object Type *BookOrder*

domain-independent modeling. The resulting meta model is depicted in Figure 3.10.

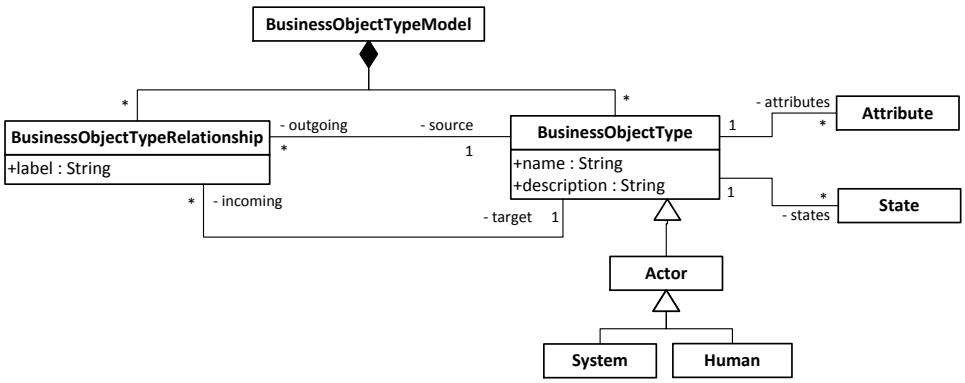


Figure 3.10: Meta Model Definition for Business Object Type Models

A business object type is identified by a name and a description of the business object type in natural language. The class `Attribute` is used to assign a set of attributes to a business object type. Valid states for objects of the defined business object type can be specified by applying the class `State`. A specific kind of a business object type is an `Actor`. An actor can be a human or a system actor that refers to a whole system or a system component. The class `BusinessObjectTypeRelationship` enables the specification of a relationship between two business object types and labels this relation to describe its type.

An example for a business object type model is depicted in Figure 3.11. As a concrete syntax for the representation of this model we apply the notation of UML class diagrams [OMG11b].

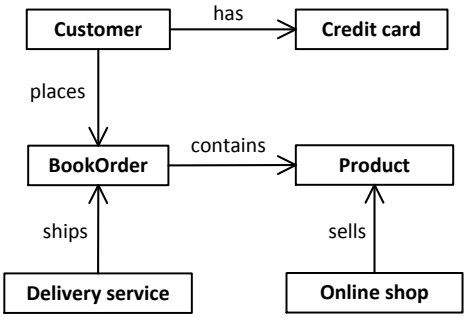


Figure 3.11: Expression of Business Object Type Relations

To summarize, we introduced an approach for the specification of business object types. By using a generic approach its application is not restricted to a specific domain. In the following, we show how the business object type model is used as a foundation for the precise specification of business goals.

### 3.3.2 Elicitation of Business Goals

To enable the specification of business goals in an explicit model, they need to be elicited in a systematic manner. The elicitation process comprises the acquisition of information from enterprise executives and managers. In [Mai13], requirements elicitation is characterized as a creative task by pointing out the importance of information search and idea discovery for the successful acquisition of relevant requirements. Several approaches and techniques are available for the acquisition of requirements [ZC05, DDH+06] that can be applied to the elicitation of business goals.

A well-known and widely used approach for requirements acquisition is the usage of interviews with stakeholders aiming at the elicitation of their needs [AT90, ME98]. To elicit business goals, the requirements engineer or business analyst performs interviews structured by questionnaires [FF94, SO01] with executives. By interviewing managers responsible for specific departments, goals can be identified that address different factors of the business success, e.g. growth strategy, marketing or finance. Another approach that facilitates the elicitation process and can be combined with the techniques discussed is the development of business models by using existing techniques, like Canvas [OP10].

When executives are not available for interviews due to a lack of time or spatial separation, business analysts can apply the technique of introspection [GL93]. In this approach, the analysts identify and define business goals without the direct involvement of stakeholders. This approach comprises market and industry sector analysis, benchmarks of comparable companies and consultancy knowledge and experience gained from previous projects. Due to the obvious disadvantages resulting from the missing consideration of executives' knowledge, this approach is only applicable if the requirements engineer is very familiar with the enterprise as well as the whole industrial domain.

Bottom-up approaches for the elicitation of business goals are ethnographic methods [SRS+93, BO00]. Hereby, the requirements engineer participates in the daily work of stakeholders to gather information. Depending on the industry, this can include the analysis of production or logistic processes as well as the participation in meetings and workshops. This technique involves employees on different levels of the organizational hierarchy. It provides specific value for the elicitation of business goals resulting from shortcomings in the daily business, e.g. weaknesses of internal processes.

In most cases, hybrid approaches combining several of the proposed techniques are used for the elicitation of business goals depending on the project context. In order to identify appropriate elicitation techniques, the ACRE method [MR96] provides a systematic decision support. Furthermore, the empirical results presented in [DDH+06] provide an overview about the effectiveness of the different techniques.

Applying the techniques discussed above or a combination of them, business goals can be elicited in a systematic way. In the following, we present a modeling approach for the specification of business goals in an integrated model.

### 3.3.3 Specification of Business Goals

By using the techniques discussed in the previous section, business goals can be elicited from the enterprise executives. To support the suitable specification of these goals, this section describes their definition in a business goal model. This comprises the definition of the business goals themselves, their operationalization to concrete actions and the identification and explicit spec-

ification of dependencies among goals. To clarify our understanding of a business goal model, we define it as follows.

**Definition 5 (Business goal model)** *A business goal model* $\mathcal{G} = (G, A, D, L, T)$ *consists of two finite sets* $G$, $A$, *a binary relation* $D \subseteq (G \times G)$, *a binary relation* $L \subseteq (G \times A)$ *and a binary relation* $T \subseteq (G \times G)$, *such that*

- $G$ *is a finite set of business goals*
- $A$ *is a finite set of actions*
- $D$ *is a finite set of decomposition relations*
- $L$ *is a finite set of operationalization links*
- $T$ *is a finite set of temporal dependencies*

Following the results of the analysis discussed in Section 3.1, we apply the KAOS goal modeling notation. Illustrating the specification and the applied modeling notation elements, Figure 3.12 sketches an exemplary business goal model. As a starting point for the creation of the business goal model, we determine a hierarchy among the elicited business goals and iteratively add more business goals in order to create a model comprising all relevant business goals.

Business goals are formulated by a condition that needs to be achieved in order to fulfill the goal [GPW06]. For example, the name of business goal *Books delivered* indicates that this goal is achieved when the ordered books are delivered successfully. An informal specification of this condition can be given by an extended description of the business goal.

The exemplary specification of business goal *Receipt sent* is shown in Table 3.5. Business goal definitions are based on business object types, e.g. a receipt that needs to be sent to a customer. Such object types are defined in a business object type model as described in Section 3.3.1. The relation between a business goal and object types is expressed by *Concerns* that indicate a connection between a business goal and one or more business object types [DFvL91, DvLF93].

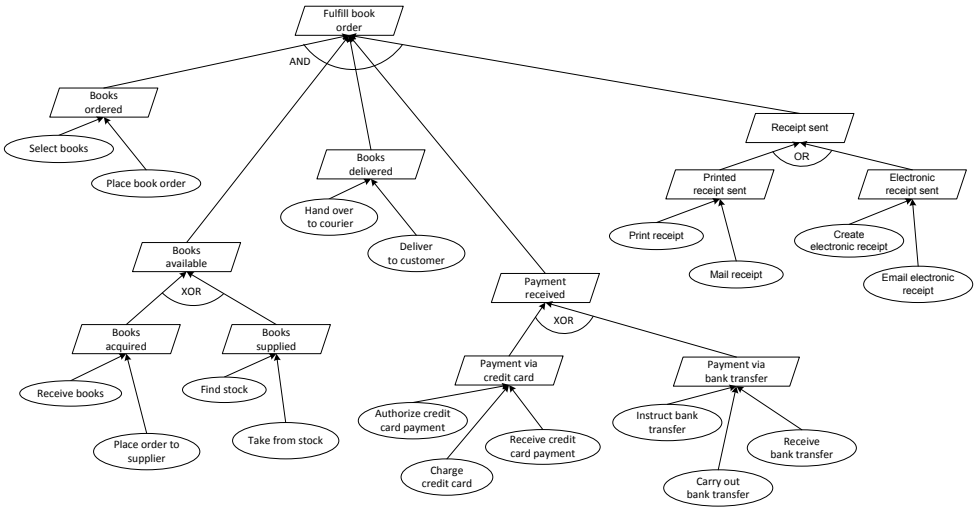


Figure 3.12: Exemplary Business Goal Model

| | |
|---|---|
| **Goal** | Receipt sent |
| **Description** | A receipt of the received payment is sent to the customer. |
| **Concerns** | Receipt, Payment |

Table 3.5: Specification of Business Goal *Receipt sent*

The assigned *Concerns* are related to the business objects types defined in the corresponding model, as described in Section 3.3.1. It is possible that not all required object types are defined appropriately. In this case, the missing object types need to be created. Figure 3.13 shows a process for the stepwise extension of the business object type model.

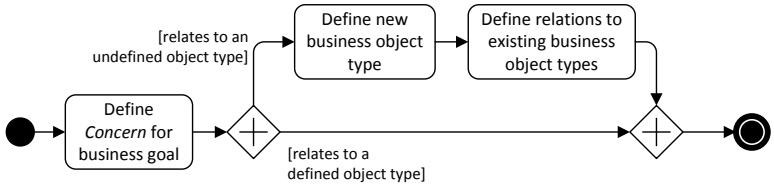


Figure 3.13: Update and Extension Process for Business Object Type Model

We consider the example of business goal *Receipt sent* and the *Concerns* for *Receipt* and *Payment*. Figure 3.14 shows how the business object type model is extended. Currently, it defines the object type *Payment* but lacks the specification of *Receipt*. To complete the specification a new object type *Receipt* with its attributes is defined. The relations to existing object type are defined and labeled according to the existing interrelations.

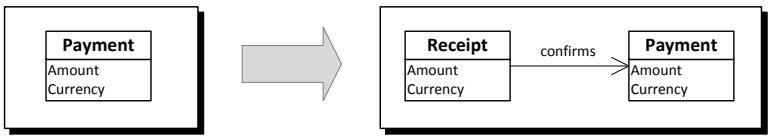


Figure 3.14: Extension of Business Object Type Model

The specification of business goals and the extension of the business object type model are performed in a top-down manner. Hereby, the business goals are iteratively refined to subgoals as shown in the example in Figure 3.12. Hierarchies in the business goal model are modeled by decomposition relations among a super goal and a set of subgoals. Distinguishing different types of logical relations, our approach supports the definition of AND, inclusive OR and exclusive OR decompositions.

AND decompositions define that each subgoal in the decomposition needs to be achieved in order to achieve the super goal. In our example the business goal *Fulfill book order* is achieved by the joint achievement of all subgoals. An AND decomposition is modeled by labeling a decomposition with the logical operator `AND`. Considering a goal $g_1$ that is AND-decomposed into subgoals $g_2$ and $g_3$, the relation can be defined by the expression ( $g_2 \wedge g_3$ ).

The concept of inclusive OR decompositions is well-known from diagram types illustrating behavior, like UML activity diagrams [OMG11b]. OR-gateways [BRU00, FESVDS07, OMG11a] in behavior diagrams are used to define inclusive decisions that are used to model alternative or parallel execution paths. The OR decomposition link is used to express alternatives to achieve a super goal without excluding the parallel achievement of subgoals. Considering the example in Figure 3.12 business goal *Receipt sent* is fulfilled by either achieving *Printed receipt sent* or *Electronic receipt sent* or by achieving both goals. It can be useful to explicitly allow the achievement of both subgoals. In our example, we express that a customer gets a direct response by

| | |
|---|---|
| **Action** | Print receipt |
| **Description** | The receipt is printed out. |
| **Actor** | OnlineSeller |

Table 3.6: Specification of Action *Print receipt*

an electronic receipt and also requires a manually signed printed receipt. To summarize, an inclusive OR decomposition in subgoals $g_2$ and $g_3$ can be formalized by ( $g_2 \vee g_3$ ).

To enable a more precise expression of OR decompositions, we introduce the exclusive OR-decomposition (XOR). An exclusive OR decomposition is fulfilled if exactly one goal is achieved. To provide an intuitive definition of the XOR dependencies, the exclusive OR is defined in the same way as existing AND-/OR-decompositions. An exclusive OR-decomposition is labeled with the keyword `XOR`. The semantics of a XOR decomposition with the subgoals $g_2$ and $g_3$ is formally defined as ( $g_2 \vee g_3$ ) $\wedge \neg$ ( $g_2 \wedge g_3$ ).

The goal refinement is performed until a business goal can be defined based on a single *Concern*. Such business goals are termed *leaf goals*. A leaf goal is not refined any further, but operationalized by actions. Hereby, concrete actions are defined that need to be executed to achieve the related leaf goal. The connection between a leaf goal and the related actions is expressed by operationalization links. Considering the example depicted in Figure 3.12, the business goal *Printed receipt sent* is operationalized to two actions *Print receipt* and *Mail receipt*. The semantics of this assignment are that both actions need to be performed to achieve the related business goal.

The purpose of an action can also be described more comprehensively, as illustrated by the example in Table 3.6. Actions are defined by a unique name and an additional description. Further, our approach provides the ability to assign an actor. Actors are modeled as a business object type and represent a system or a human person, as described in Section 3.3.1. If no suitable actor is defined, the object type model needs to be extended by applying the process described in Figure 3.13.

### 3.3.4 Temporal Business Goal Dependencies

In the previous section, we have shown how hierarchical relationships between business goals can be expressed by decomposition relations and logical operators related to this decomposition. These links are not intended to define temporal dependencies among business goals. However, the elicitation and explicit specification of the stakeholders' knowledge about these dependencies is a crucial requirement for their consideration in the derivation of business process models. Temporal dependencies are used to prescribe the temporal order in which business goals shall to be achieved. Such dependencies can be specified in two different ways.

The *preceding goal* order dependency describes that if goal $g_1$ shall be achieved, goal $g_2$ needs to be achieved beforehand. It does not matter if other goals are achieved in the meantime. To express this temporal order dependency, the goal annotation `Order.Predecessor` *goal name* is introduced. For example, if a goal $g_1$ is annotated with `Order.Predecessor` $g_2$, it demands the achievement of goal $g_2$ as a prerequisite for the achievement of $g_1$.

To express a temporal dependency in terms of a required successor, we introduce the *succeeding goal* dependency that is defined by using the annotation `Order.Successor` *goal name*. The annotation `Order.Successor` $g_2$ for $g_1$ expresses that goal $g_2$ needs to be achieved some time later than goal $g_1$. It does not matter, if other goals are achieved in between.

To illustrate the modeling capabilities to specify temporal dependencies, Figure 3.15 gives an overview about two examples. Figure 3.15 (a) shows an exemplary preceding goal dependency that expresses that before the goal *Books delivered* can be achieved the goal *Books available* needs to be achieved. A successor dependency is shown in Figure 3.15 (b), expressing that after goal *Books delivered* the goal *Payment received* needs to be achieved. This dependency ensures that after the ordered goods have been paid for, the order is actually delivered.

The definition of temporal dependencies between goals can lead to invalid statements about their order, e.g. goal $g_a$ has predecessor $g_b$ and $g_b$ has predecessor $g_a$. Obviously, this would lead to an invalid dependency definition that cannot be fulfilled. To avoid the definition of such invalid dependencies, we define a set of restrictions for the definition of temporal dependencies. An

overview about these rules is sketched in Table 3.7.

| Type of Dependency | Unassignable Goals |
|---|---|
| Predecessor | The goal itself,<br>All child-goals in the subtree under the goal,<br>All super goals,<br>All succeeding goals |
| Successor | The goal itself,<br>All child-goals in the subtree under the goal,<br>All super goals,<br>All preceding goals |

Table 3.7: Overview of Restrictions for Goal Dependencies

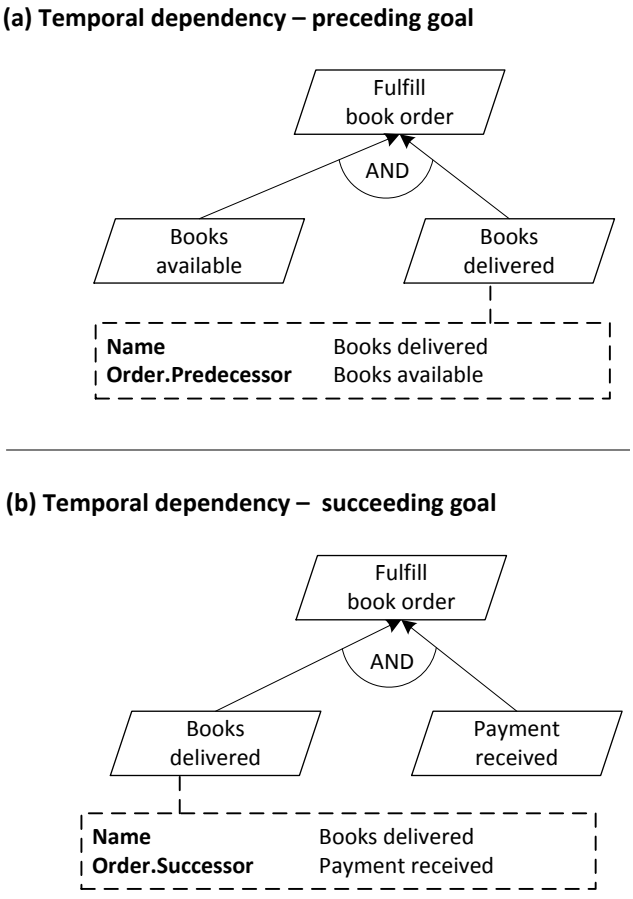


Figure 3.15: Definition of Temporal Order Dependencies

## 3.4 Definition of Goal Achievement States

In the previous section, the identification and description of business goals as well as the modeling of dependencies among them has been discussed. Business goals are defined by a name and a description that informally describes under which condition a goal is achieved. The expression of such conditions for the achievement of goals is understandable for the involved actors. However, the specification in natural language has disadvantages with respect to preciseness, verifiability and traceability of the goal achievement. To overcome these shortcomings, this section presents an approach for the precise specification of goal achievement states based on the defined business object types. Furthermore, we describe the consistent definition of these states through the hierarchy of the business goal model.

### 3.4.1 Specification of Leaf Goal Achievement States

As input for the specification of goal achievement states, we consider a business goal model and a set of business object type definitions. For illustration purposes, we describe the following steps by applying them to the exemplary leaf goal *Books delivered* and the related actions depicted in Figure 3.16.

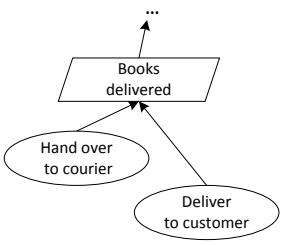


Figure 3.16: Model Excerpt for Business Goal *Books delivered*

A business goal is specified by a unique name and a description. Relations to business object types are defined by the *Concerns* attribute. By refining the *Concerns* concept, we aim for the definition of achievement conditions for business goals in terms of precise state definitions called *achievement states*. First, we define these achievement states for all leaf goals in the business goal model. The aggregation of these states to the super goals is described in the next section.

We assign a state definition based on the *Concerns* of a goal. It defines that a business goal is achieved if one or more objects of a certain business object type are in the defined state. Following existing categorizations and defini-

| | |
|---|---|
| **Goal** | Books delivered |
| **Description** | The ordered books are delivered to the customer. |
| **Concerns** | BookOrder |
| **AchievementState** | $(\forall\ b : BookOrder)$ $\Diamond\ b.Delivered$ |

Table 3.8: Definition of Business Goal *Books delivered*

| | | |
|---|---|---|
| **Action** | Hand over to courier | Deliver to customer |
| **Actor** | OnlineSeller | Courier |
| **Input** | b:BookOrder | b:BookOrder |
| **Output** | b:BookOrder | b:BookOrder |
| **PreCondition** | b.Available | b.InDelivery |
| **PostCondition** | b.InDelivery | b.Delivered |

Table 3.9: Definition of Actions *Hand over to courier* and *Deliver to customer*

tions of goal types [DvLF93], our approach applies the concept of *achieve goals*. An *achieve goal* defines that all objects of a specific type need to be in a certain state at some point in time.

Expressing the achievement state in a formalized way, we use the temporal operator *Finally* expressed by the symbol $\Diamond$. Achievement states follow the pattern where $a$ defines an object of type $AType$ and $AchState$ defines the state that needs to be achieved.

$$(\forall\ a : AType)\ \Diamond\ a.AchState$$

Applying this to our example, the achievement state for leaf goal *Books delivered* is specified as depicted in Table 3.8. We define the achievement state over object type *BookOrder*. The business goal is achieved if all objects of type *BookOrder* are in the state *Delivered* at some point in time.

An action is described by changes of object states. Object types received as input as well as object types produced as output are defined in the action specification. To specify the change operations, pre- and postconditions are defined for input and output object types. Exemplary specifications for the actions *Hand over to courier* and *Deliver to customer* are shown in Table 3.9.

### 3.4.2 Aggregation of Achievement States

In the previous step, the specification of achievement states for leaf goals has been described. The defined states for leaf goals are aggregated to the higher level goals as depicted in Figure 3.17. Achievement states are aggregated based on the hierarchical relations within a business goal model. We consider two business goals $g_1$ and $g_2$ with achievement states $AchState_{g_1}$ and $AchState_{g_2}$. These states are aggregated to define the achievement state of their super goal $g_3$. The states are aggregated based on the logical operators related to the decomposition relations. Table 3.10 shows the achievement state aggregation depending on the logical operators (AND, OR, XOR).

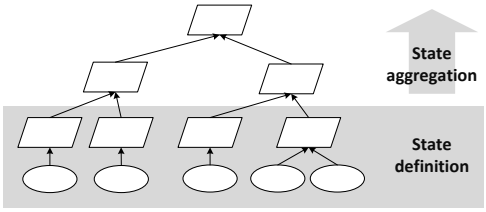


Figure 3.17: State Definition and Aggregation

The aggregation is performed bottom-up through the hierarchy of the business goal model. The presented techniques provide precise specifications of business goals and actions in terms of states that need to be achieved and of performed changes of object type states.

| **Logical Operator** | **Aggregated Achievement State** |
|---|---|
| $g_1\ AND\ g_2$ | $AchState_{g_1} \wedge AchState_{g_2}$ |
| $g_1\ OR\ g_2$ | $AchState_{g_1} \vee AchState_{g_2}$ |
| $g_1\ XOR\ g_2$ | $(AchState_{g_1} \wedge (\neg AchState_{g_2})) \vee ((\neg AchState_{g_1}) \wedge AchState_{g_2})$ |

Table 3.10: Aggregation of Achievement State Definitions

## 3.5 Validation of Action Composability

As discussed, the business goal model provides the foundation for the derivation of a business process that is executed to achieve the stated goals. This business process is derived through the composition of the identified and specified actions. A prerequisite for the successful derivation is the composability of these actions. This means, the actions can be composed without any mismatches of their pre- and postconditions.

For this purpose, we introduce an approach to validate whether the specified actions can be composed. Identified mismatches are resolved through changes and extensions of the action specifications. Section 3.5.1 introduces the concept of object life cycle models. These models are used to explicitly specify states and state transition for certain business object types. Applying object life cycle models, Section 3.5.2 presents our approach for the validation of action composability.

### 3.5.1 Object Life Cycle Models

Actions describe object state changes defined by their pre- and postconditions. To provide the foundation for the validation of the action composability, we specify valid states and state transitions in an explicit model. For this purpose, we leverage the concept of object life cycle models. Object life cycles are used to model allowed object states as well as transitions among them [EE97, AB01, SS02]. Based on the work of Küster et.al. in [KRG07], we define an object life cycle model as follows.

**Definition 6 (Object life cycle model (OLM))** *([KRG07]) Given an object type o, its object life cycle model* $OLM_o = (S, s_a, S_\Omega, \sum, \delta)$ *is defined by a finite set of states S, where* $s_\alpha \in S$ *is the initial state and* $S_\Omega \subseteq S$ *is the set of final states; a finite set of events* $\sum$ *; a transition function* $\delta : S \times \sum \rightarrow P(S)$.

Figure 3.18 depicts an exemplary OLM. It defines all states for a given object type and defines valid transitions among these states. Starting from the initial state, the different states are connected to each other by transitions that are related to an event triggering the corresponding transition. State $s_6$ indicates

the final state for each object of the defined object type.

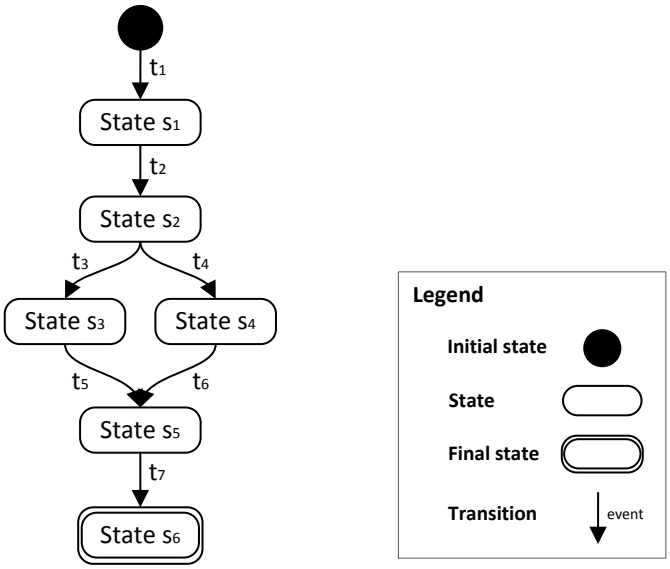


Figure 3.18: Object Life Cycle Model

### 3.5.2 Validation of Composability

To validate the composability of the specified actions we use a two-step approach. First, we analyze whether the actions of a leaf goal can be composed in a valid manner while achieving the operationalized leaf goal (local composability). Second, we consider that actions which handle the same object type can be related to different leaf goals. In order to ensure that actions operationalizing different goals can be composed to a business process, their composability is validated as well (global composability).

#### Validation of Local Composability

First, we validate the composability of all actions for a certain leaf goal. *Local composability* for a leaf goal $g$ and a set of actions $A$ operationalizing $g$ is given if the following conditions are fulfilled.

- All actions $a \in A$ can be composed in a sequence such that for all consecutive actions $a_i$, $a_j$ the postcondition of $a_i$ matches the precondition

of $a_j$.

- The postcondition of exactly one action $a_i$ matches the achievement state of $g$ and there exists no action $a_j$ that matches the postcondition of $a_i$ with its precondition.

The first step is the specification of *local object life cycle models* (lOLM). Applying the general concept of object life cycle models, lOLMs consider object states and transitions related to a leaf goal. A local object life cycle model $lOLM_{o,g}$ defines the object life cycle model for an object type $o$ in the context of a leaf goal $g$. Accordingly, the $lOLM_{o,g}$ defines all valid states and transitions for an object of object type $o$ that can be achieved by performing the actions related to leaf goal $g$.

The four steps for the construction of lOLMs are sketched in Figure 3.19. Starting from the final state given by the achievement state of the leaf goal, the algorithm defines states and transitions according to the actions that operationalize the leaf goal. The result is a connected and coherent object life cycle model that defines all states as well as valid transitions among them that lead to the final achievement state of the business goal. If such a lOLM can be constructed, a valid sequence order can be derived for the action composition based on the order of the defined transitions. In the following, the different steps are discussed in detail and illustrated by the exemplary generation of a lOLM for business goal *Books delivered*. To generate all lOLMs this algorithm is executed for all leaf goals in the business goal model.

**Step 1: Set final state**

The aim of the first step is to identify and define the final state of the lOLM. Starting from the business goal model the relevant business objects are identified by the object types referred to in the *Concerns* of the goal. Considering the example depicted in Figure 3.20 we create a lOLM of object type *BookOrder* for business goal *Books delivered* abbreviated as $\text{lOLM}_{BookOrder,BooksDelivered}$.

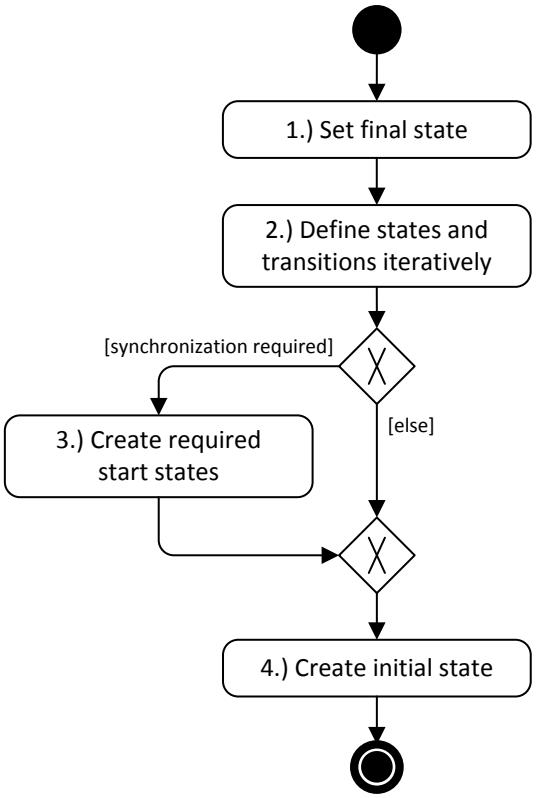


Figure 3.19: Generation of local Object Life Cycle Models (lOLM)

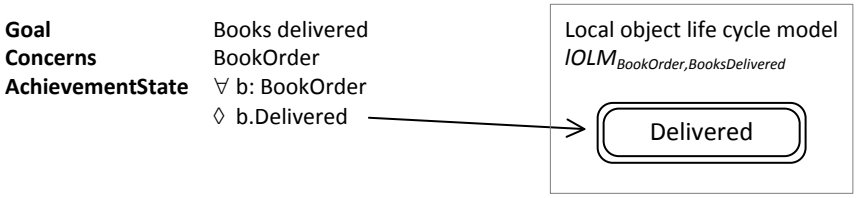


Figure 3.20: Final State Definition of Object Type *BookOrder* for Business Goal *Books delivered*

The business goal *Books delivered* is defined by the achievement state *b.Delivered* expressing that each object of type *BookOrder* needs to be in state *Delivered* at some point in time. We define this achievement state as final state for the lOLM. It is important to notice that this describes a final state in a local context, defined by the addressed business goal and does not necessarily represent the global final state for objects of type *BookOrder*.

**Step 2: Define states and transitions iteratively**

Starting from the final state identified in Step 1, the preceding states and transitions for a lOLM are derived from the defined actions and their pre- and postconditions. The iterative definition of states and state transitions is described in Algorithm 1 and explained in the following. Input for the construction step is an existing lOLM including a valid final state that has been defined as described in Step 1. The addressed leaf goal is also required as a precondition for the algorithm.

**Algorithm 1** Creation of States and Transitions

**Precondition:** Leaf goal $g$, lOLM $\mathcal{J}$ with a valid final state
**Postcondition:** Updated lOLM $\mathcal{J}$

```
unassignedActions ← g.getActions()
currState ← J.getFinalState()
while ((unassignedActions.size() ≥ 0) ∧ (¬initReq)) do
    for all a ∈ unassignedActions do
        if a.getPostCondition() = currState then
            matchingActions.add(a)
    if matchingActions.size() = 1 then                          ▷ Scenario 1
        unassignedActions.remove(a)
        if (a.getPreCondition() ≠ initial) then
            s ← createNewState(a.getPreCondition())
            J.addState(s)
            J.createNewTransition(a.getName(), s, currState)
            currState ← s
        else
            if unassignedActions.size() = 0 then
                ExecuteStep4
            else
                RaiseException(IncompleteComposition)
    else if matchingActions.size() = 0 then                     ▷ Scenario 2
        RaiseException(NoMatchingActionFound)
    else                                                        ▷ Scenario 3
        RaiseException(MultipleMatchingActionsFound)
ExecuteStep3
```

First, all actions operationalizing the addressed leaf goal are marked as unassigned actions. The final state is marked as the current state that needs to be reached. To identify matching actions, the algorithm iterates through the list of all unassigned actions of the business goal. A matching action is found if its postcondition matches the currently selected state in the lOLM. For il-

lustration purposes, we consider the example sketched in Figure 3.21. The postcondition has to match the state *Delivered*. As depicted, the postcondition of action *Deliver to customer* matches this state and is selected accordingly.

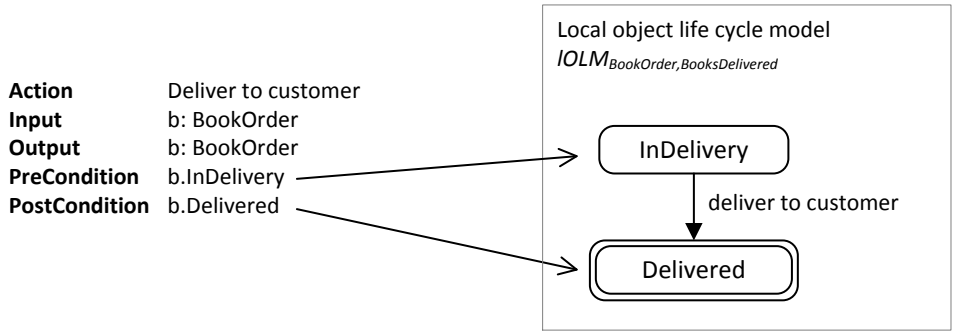


Figure 3.21: Matching of State and Action Precondition

The identification of matching actions can result in different scenarios.

**Scenario 1: Exactly one matching action is identified.**

The matching of exactly one matching action is the desired scenario. The action is removed from the list of unassigned actions. In case the matching action has a precondition, a new state is created according to the defined precondition and a new transition is defined to connect it with the current state. The transition is labeled with the name of the action. An example is depicted in Figure 3.21. The new state *InDelivery* is created and the transition *deliver to customer* connects both states. If the identified action has no explicit precondition, this implies that it is connected to an initial node in the next step. Due to completeness it needs to be checked whether unassigned actions are left. If no actions are left, the construction can be continued with the definition of an initial state in step 4. Unassigned actions raise an *Incomplete Composition* exception. Table 3.11 proposes resolution strategies to overcome this exception.

**Scenario 2: No matching action is identified**

If no matching action can be identified, it is not possible to construct a coherent lOLM. Hence, it is not possible to create a composition of actions with respect to their pre- and postconditions. In this case, the *No Matching Actions Found*

| Exception | Trigger | Resolution Strategies |
|---|---|---|
| Incomplete composition | The constructed composition does not consider all actions | • Deletion of the unconsidered actions<br>• Redefine pre- and postconditions of actions to integrate the unconsidered actions |
| No matching action found | An identified state is not matched by the postcondition of an unassigned action | • Definition of a new action that matches the state with its postcondition<br>• Adapting the postcondition of an unassigned action to match the state |
| Multiple matching actions found | An identified state is matched by more than one action | • Delete one or more actions with identical postcondition<br>• Redefine postcondition of one or more actions in multiple matching |

Table 3.11: Exceptions resulting from an incorrect Action Specification

exception needs to resolved (cf. Table 3.11)

**Scenario 3: Multiple actions match the state**

The identification of more than one matching action would lead to the creation of different transitions (and states according to the preconditions). This means, an action composition that is consistent to this lOLM would not allow the execution of each action in a sequence. This contradicts the semantics of an operationalization which define that all actions need to be performed in order to achieve the goal. The resulting *Multiple Matching Actions Found* exception can be resolved as described in Table 3.11.

The further processing of the construction algorithm depends on the precondition of the last action. If the action does not have an explicit precondition, the lOLM is completed by performing step 4. If the action has a precondition which cannot be matched, step 3 closes this gap.

**Step 3: Create required start states**

In the previous step, we described the stepwise creation of lOLMs by connecting pre- and postconditions of actions. The iterative extension of the lOLMs is continued as long as an action matches a given state with the postcondition and a new state can be created as indicated by the precondition. This means, if the last available action has a precondition, this precondition cannot be matched by another action. Nonetheless, the required state needs to be considered to create a coherent lOLM.

For this purpose, we introduce the concept of required start states. A required start state defines a state that is not achieved by an action related to the addressed leaf goal, but is required as a prerequisite for the actual achievement of the addressed business goal. This means that the state needs to be achieved through an action related to another business goal that is not considered in the local context of the lOLM. The matching of the required start states with the corresponding actions is part of the validation of the global composability.

To illustrate this concept, an example is shown in Figure 3.22. We consider the action *Hand over to courier* as the last action added to the lOLM. The postcondition matches the state *InDelivery* and it has a precondition *Available*. To define a valid transition for this action, a state *Available* needs to be defined as well, although no action matches it by its postcondition. We define a required start state *Available* that allows us to create the transition in a coherent manner.

**Step 4: Create initial state**

Finally, the lOLM is completed by the creation of an initial state. According to the previous processing, we distinguish two scenarios for the final step which are slightly different. If step 4 is performed directly after step 2 we execute step 4(a). After the definition of required start states in step 3, step 4(b) is performed.

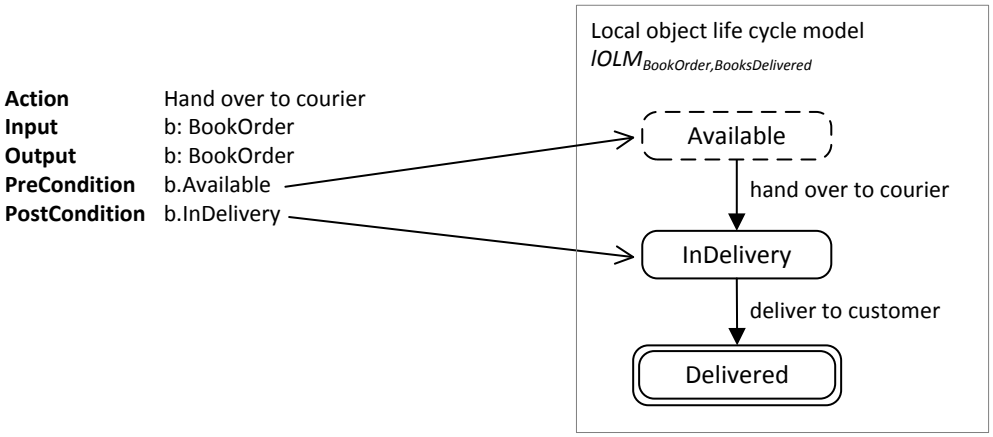


Figure 3.22: Creation of Required Start States

**Scenario 4(a): No required start state** This scenario defines the initial state creation if no required start state is defined. Figure 3.23 shows an example illustrating the creation for a transition *select books*. We consider that the action *select books* has no precondition. That means no specific state of object *BookOrder* needs to be achieved beforehand. To complete the lOLM we create an initial state node and connect it to the state *Selected* by the transition *select books*.

**Scenario 4(b): Required start state** If a required start state has been created in step 3, this state needs to be connected to an initial state resulting in a connected model. Figure 3.23(b) depicts the initial state creation for the required start state *Available* defined in the previous step. An initial state node is created and a transition is defined to connect it with the required start state. This transition is intended as a placeholder which is replaced during the glueing of the lOLMs by the actual transition resulting in state *Available*.

The 4-step algorithm is executed for all leaf goals. To provide a global view on the object life cycles and to enable the validation of state consistency, we generate global object life cycle models in the next step.

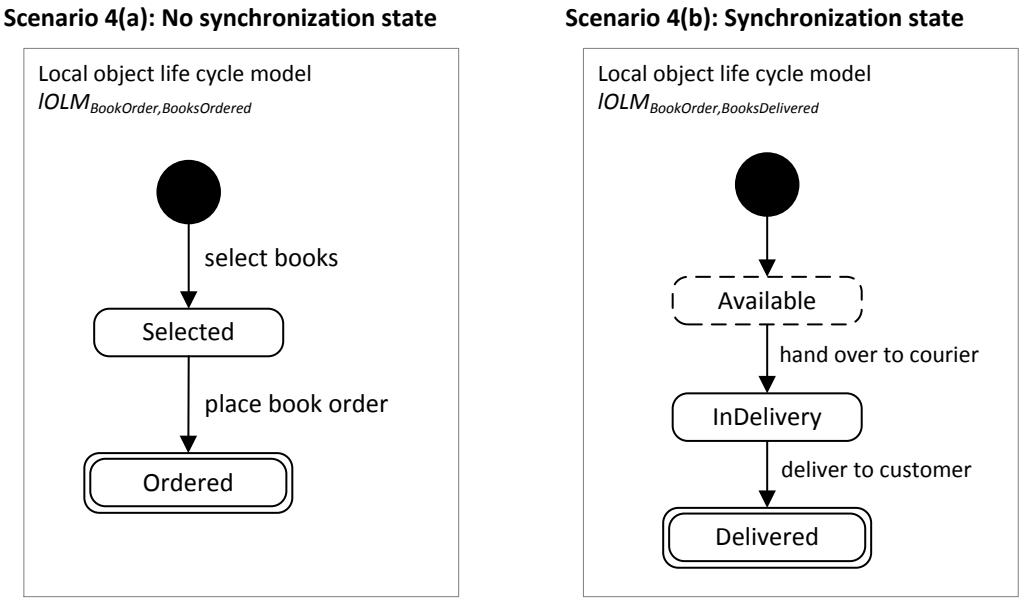


Figure 3.23: Scenarios for Initial State Creation

### Validation of Global Composability

Following the algorithm presented, lOLMs are generated for each leaf goal resulting in a set of multiple lOLMs for each object type. The local composability ensures that the actions can be composed and that they fulfill the related leaf goal. Considering the derivation of business process models, the composability needs to be validated on the level of the complete business goal model as well. For this purpose, the composability of all actions processing the same business object types is validated in this step.

We compose the lOLMs to an integrated object life cycle model as sketched in the conceptual overview in Figure 3.24. For illustration purposes, we consider the object type *BookOrder* as a running example. Figure 3.25 sketches four lOLMs that have been derived from the goal model shown in Figure 3.12 by using the algorithm introduced above. Three lOLMs contain required start states that need to be matched by the states and transitions of the other lOLMs in order to construct a coherent and consistent object life cycle model.

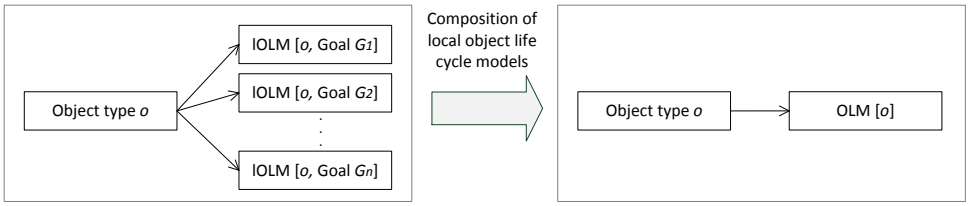


Figure 3.24: Composition of lOLMs

Enabling the composition of an integrated object life cycle model, Algorithm 2 describes the steps of our approach. First, we identify the starting point for the derivation of the global OLM. That means, all lOLMs are selected that do not include a required start state. The resulting selection contains lOLMs that can be connected to the initial state. Considering our example, the lOLM in Figure 3.25 (a) fulfills this requirement. The other models are not selected because they contain a required start state.

In our example, only one lOLM is selected. It is also possible that multiple lOLMs are selected in the algorithm which need to be merged. Picking two lOLMs the algorithm distinguishes four different scenarios based on the comparison of first and final states. For a set of three or more models, the pairwise model comparison is performed by merging two models and afterwards merging the resulting model with the next unmerged lOLM. For each possible scenario a merging pattern is provided. The first scenario is applied if two models are merged that have identical first and final states. Following the depicted pattern in Figure 3.26 a combined model is created by merging the first and final states. The intermediate sequence of states is integrated through divergent paths.

Figure 3.27 shows the pattern for merging two models with different first and final states. Both sequences of states are aligned in divergent paths in the merged model. The transitions to the first states are connected to the common initial state.

**Algorithm 2** Creation of Global OLM

**Precondition:** A Set $J$ of lOLMs
**Postcondition:** Merged global OLM

```
if (J contains at least one lOLM without required start state) then
    for all (lOLMs without required start state) do
        switch MergingScenario comparing lOLM_A,lOLM_B do
            case (sameFirstState ∧ sameFinalState)
                ApplyMergingPattern(P1)
            case (¬sameFirstState ∧ sameFinalState)
                ApplyMergingPattern(P2)
            case (sameFirstState ∧ ¬sameFinalState)
                ApplyMergingPattern(P3)
            case (¬sameFirstState ∧ ¬sameFinalState)
                ApplyMergingPattern(P4)
else
    RaiseException(NoInitialState)
repeat
    for all Final states in OLM do
        FindMatchingRequiredStartStates()
        ApplyMergingPattern(P5)
until No more matching states found
if (there exists an unassigned lOLM) then
    RaiseException(UnboundRequiredStartState)
```

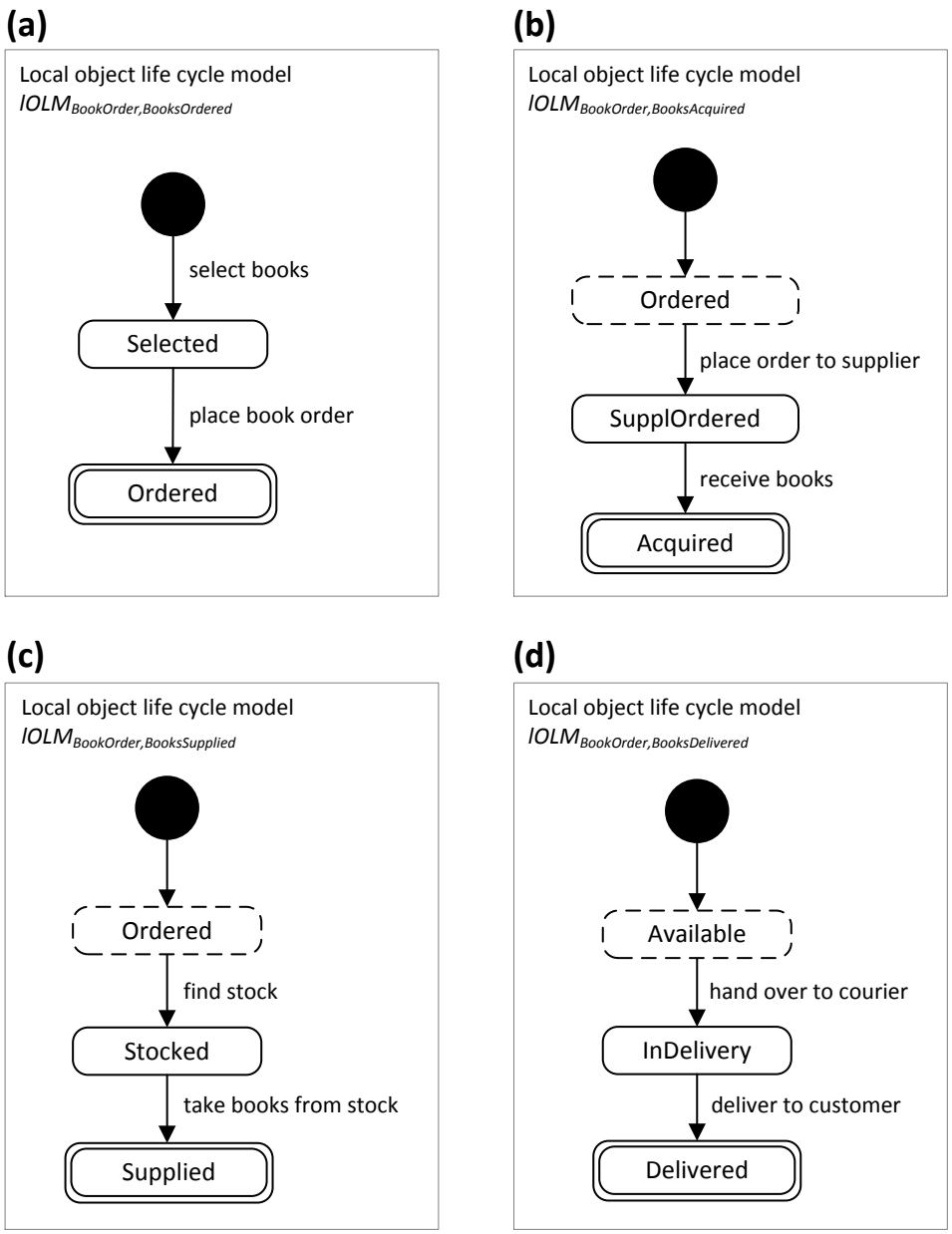


Figure 3.25: Local Object Life Cycle Models for Object Type *BookOrder*

Scenario 3 addresses the merging of lOLMs with same first and different final states as depicted in Figure 3.28. The merged model is composed by the common first state and divergent paths of the succeeding states. In the depicted pattern the following states are final states. If the corresponding lOLMs con-

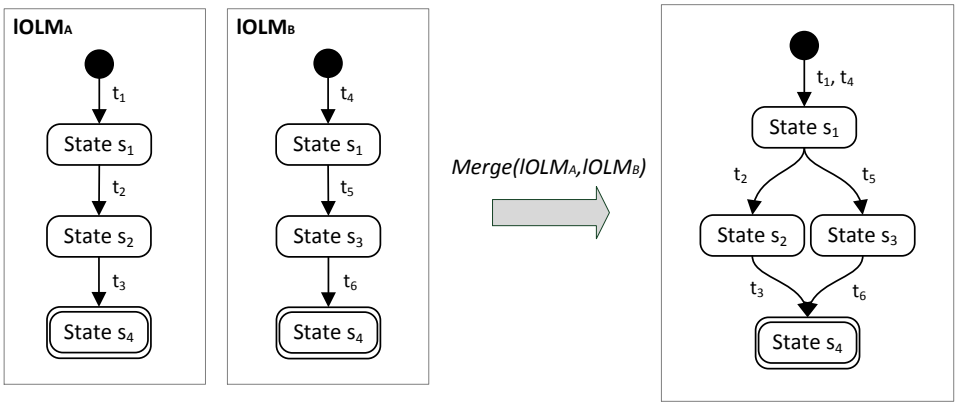


Figure 3.26: Merging of lOLMs with same First States and same Final States (Merging Pattern P1)

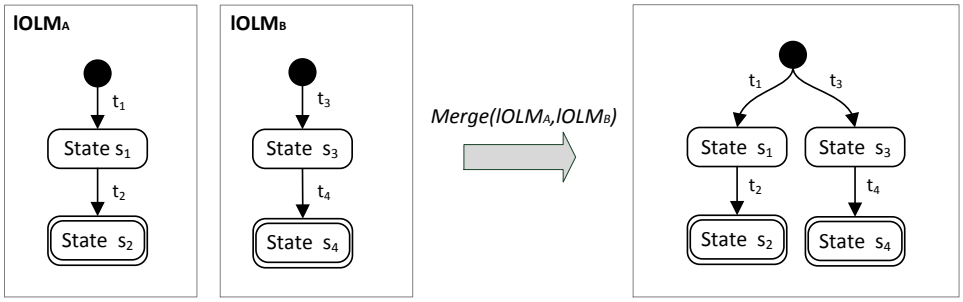


Figure 3.27: Merging of lOLMs with different First States and different Final States (Merging Pattern P2)

sist of more than two states, the complete sequences are inserted accordingly.

Figure 3.29 shows the merging of lOLMs with different first and same final states. Starting from the initial state two divergent paths are created for the different intermediate states. Both paths are merged in the common final state.

Following the described patterns, lOLMs without a required start state can be merged and connected to the initial node of the composed OLM. The successful application of these patterns requires the availability of lOLMs that do not have a precondition in terms of a required start state. If no lOLM is defined

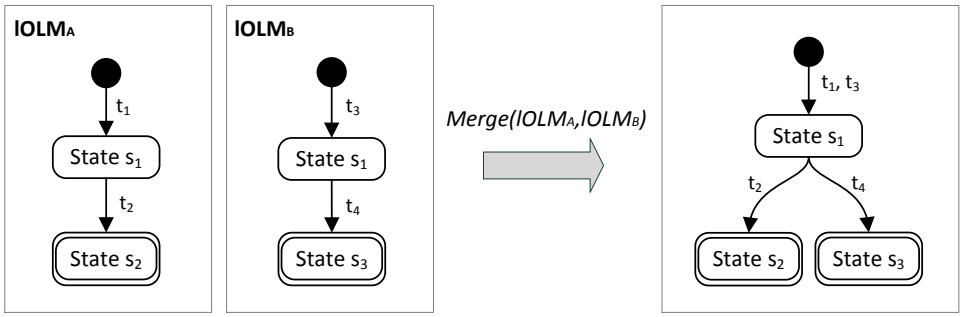


Figure 3.28: Merging of lOLMs with same First States and different Final States (Merging Pattern P3)

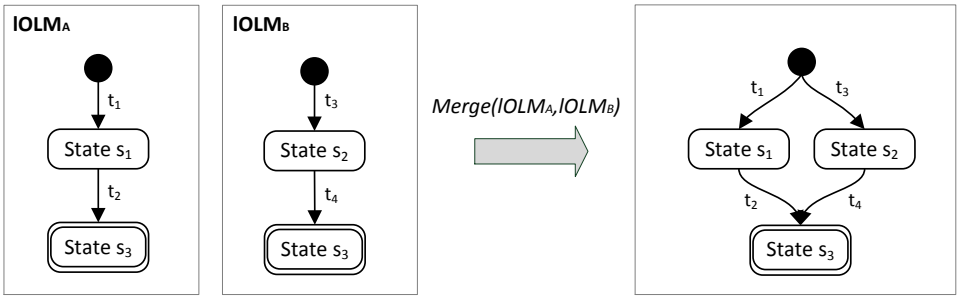


Figure 3.29: Merging of lOLMs with different First States and same Final States (Merging Pattern P4)

that matches this requirement, the algorithm raises the exception *NoInitialState*.

**Resolve Exception(NoInitialState)** The exception *NoInitialState* is raised if no lOLM can be found for the initial composition step. To resolve this exception we propose different strategies. First, it is possible to redefine the precondition of an action related to the required start state of a lOLM. The precondition needs to be set to initial, which means that no explicit precondition is required. The redefined precondition makes the required start state obsolete and leads to a lOLM that can be processed by applying the described approach.

In some cases the redefinition of an existing action is not appropriate or would

lead to an inconsistent modeling. Alternatively, a new action is created that matches a required start state by its postcondition and has no explicit precondition. The defined action is added to the corresponding lOLM. Hereby, it closes the gap between the initial state and the required start state. The result is a lOLM that can be processed following the defined procedure.

The first step of the algorithm provides a composition considering all lOLMs that do not have a required start state. Based on this composition, lOLMs with required start states are connected iteratively. A lOLM can be merged if its required start state matches one of the final states. For each matching state the pattern depicted in Figure 3.30 is applied. A common state is defined by replacing the final and required start state. *Transition b* connects *State A* and *State B*, merging both models in a coherent manner.

The consideration of only one event in this pattern is sufficient due to restriction for the definition of preconditions. As described, a valid precondition contains one required state for each relevant object type. Since the events are derived from the preconditions of these actions and the lOLM considers exactly one object type, only one event can be defined in a lOLM.

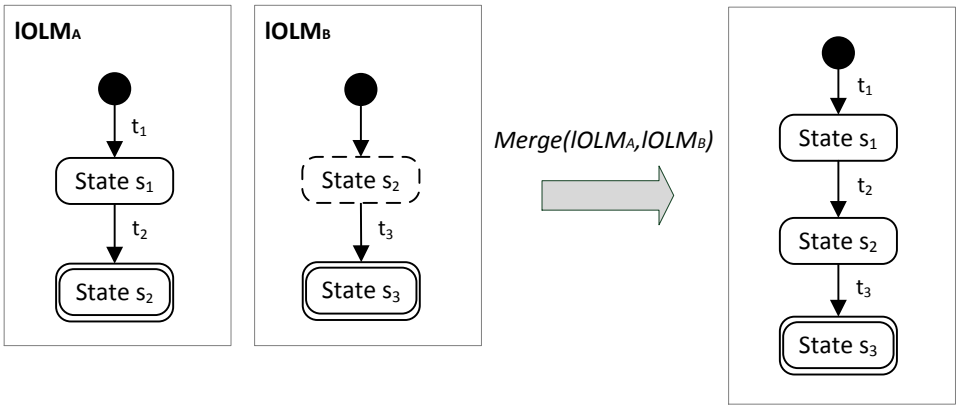


Figure 3.30: Merging of lOLMs with Matching Required Start State (Merging Pattern P5)

To merge required start states, pattern P5 is executed until no further matching lOLMs are available. If all lOLMs are merged, a complete and coherent OLM is generated that considers all relevant states and valid transitions

among them. An unmerged lOLM raises an *UnboundRequiredStartState* exception.

**Resolve Exception(UnboundRequiredStartState)**

An *UnboundRequiredStartState* exception is raised if a required start state cannot be matched, since no state is defined that matches the required start state. Figure 3.31 depicts an *UnboundRequiredStartState* exception for the running example. The lOLM cannot be merged by the provided patterns, since the required start state *Available* is not matched by any state.

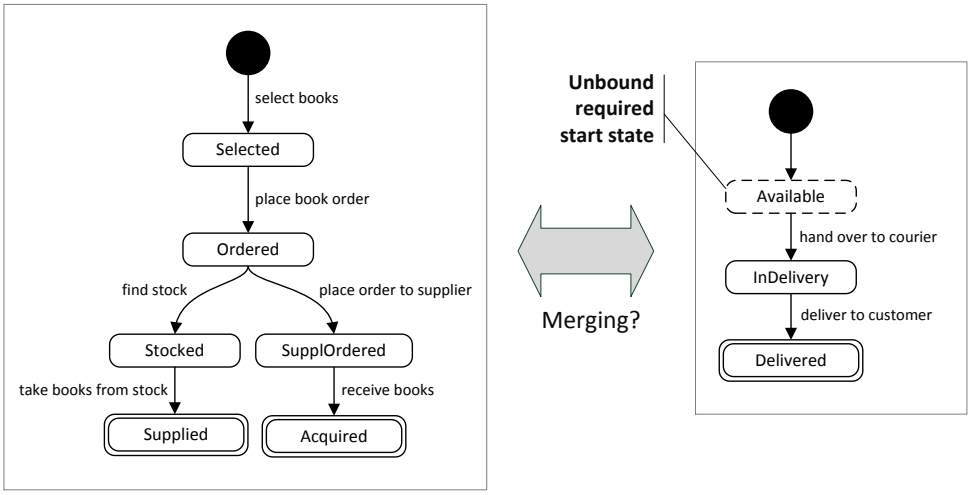


Figure 3.31: Unbound Required Start State

An unbound required start state can be resolved in different ways. To illustrate the different possibilities Figure 3.32 depicts the decision process for the selection of an appropriate resolution strategy. Binding an unmatched required start state can be resolved either by the adoption of an existing action or by the definition of a new action.

An existing action can be adapted by the redefinition of pre- or postconditions. Considering the example in Figure 3.31, this can be applied to the redefinition of postconditions for *take books from stock* or *receive books*. A redefined postcondition with *Available* creates a new final state that matches the required start state. Alternatively, the precondition for action *hand over to courier* can

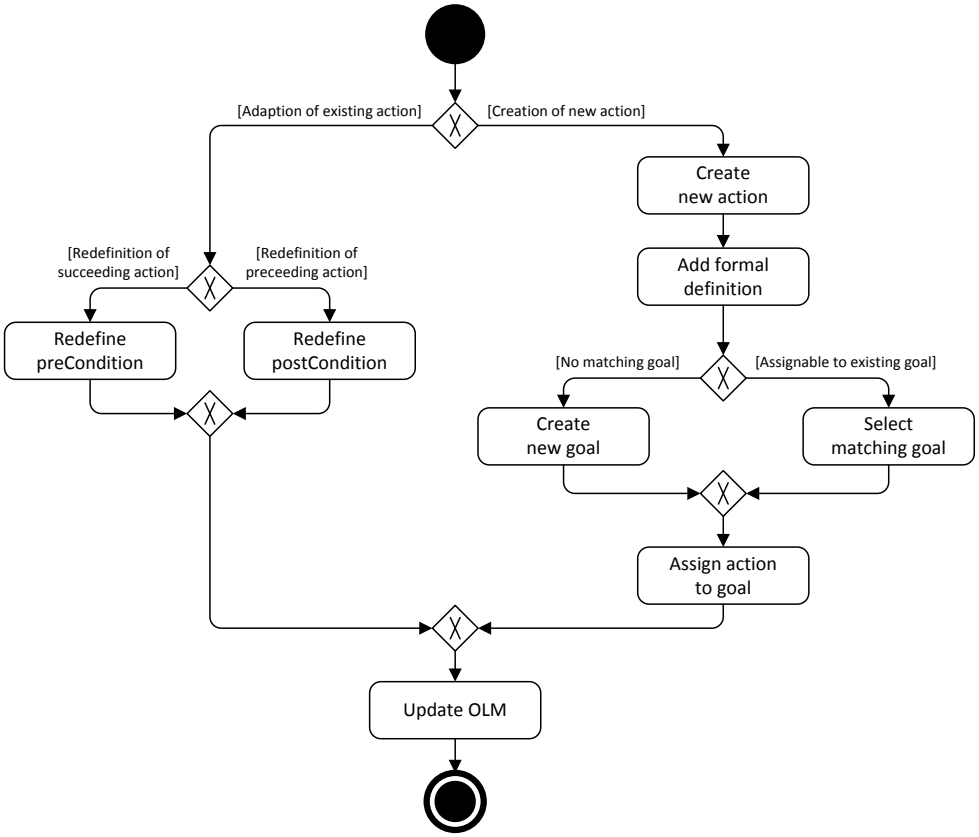


Figure 3.32: Resolve Exception *Unbound required start state*

be adapted leading to a changed required start state that matches an existing state definition or the initial state.

If an existing action cannot be modified appropriately, a new action needs to be created to match the unbound required start state. The formal definition of this action is specified, matching the required start state. To ensure a coherent modeling the action is assigned to a business goal. If an appropriate assignment is not possible or suitable, a new business goal is created and integrated into the goal model. By resolving all unbound required start states, the merging algorithm finalizes the OLM definition. The result is an integrated, connected and consistent OLM for each object type.

## 3.6 Summary and Discussion

In this chapter, we elicited requirements for business goal modeling and performed an analysis of related work with respect to these requirements. Addressing the stated requirements, we presented an approach for business goal modeling based on the existing KAOS language.

The modeling approach developed here enables the specification of business goals and operationalized actions on different levels of abstraction. Logical relations between goals as well as the expression of temporal dependencies among goals are considered by suitable modeling capabilities. The precise specification of business goals and actions is defined based on business object types. To define these object types, we proposed a domain-independent modeling approach for the specification of relevant elements in the business context. An integrated meta model considering all language elements is depicted in Figure 3.33.

In addition to the contribution of a formal language definition, our approach also provides methodical support and guidance. We described the iterative specification of business object types and business goals. The systematic aggregation of achievement states ensures consistency among goals on different levels of abstraction and improves the overall quality of the specification. The construction of object life cycle models and the semi-automated validation of action composability provides the foundation for the derivation of business process models. By providing an integrated specification process, the approach supports requirement engineers and business analysts in the creation of a business goal specification. In the next chapter, we describe how the elicited and specified information, especially the defined goal dependencies, are considered in the derivation of a business process that achieves the stated business goals.

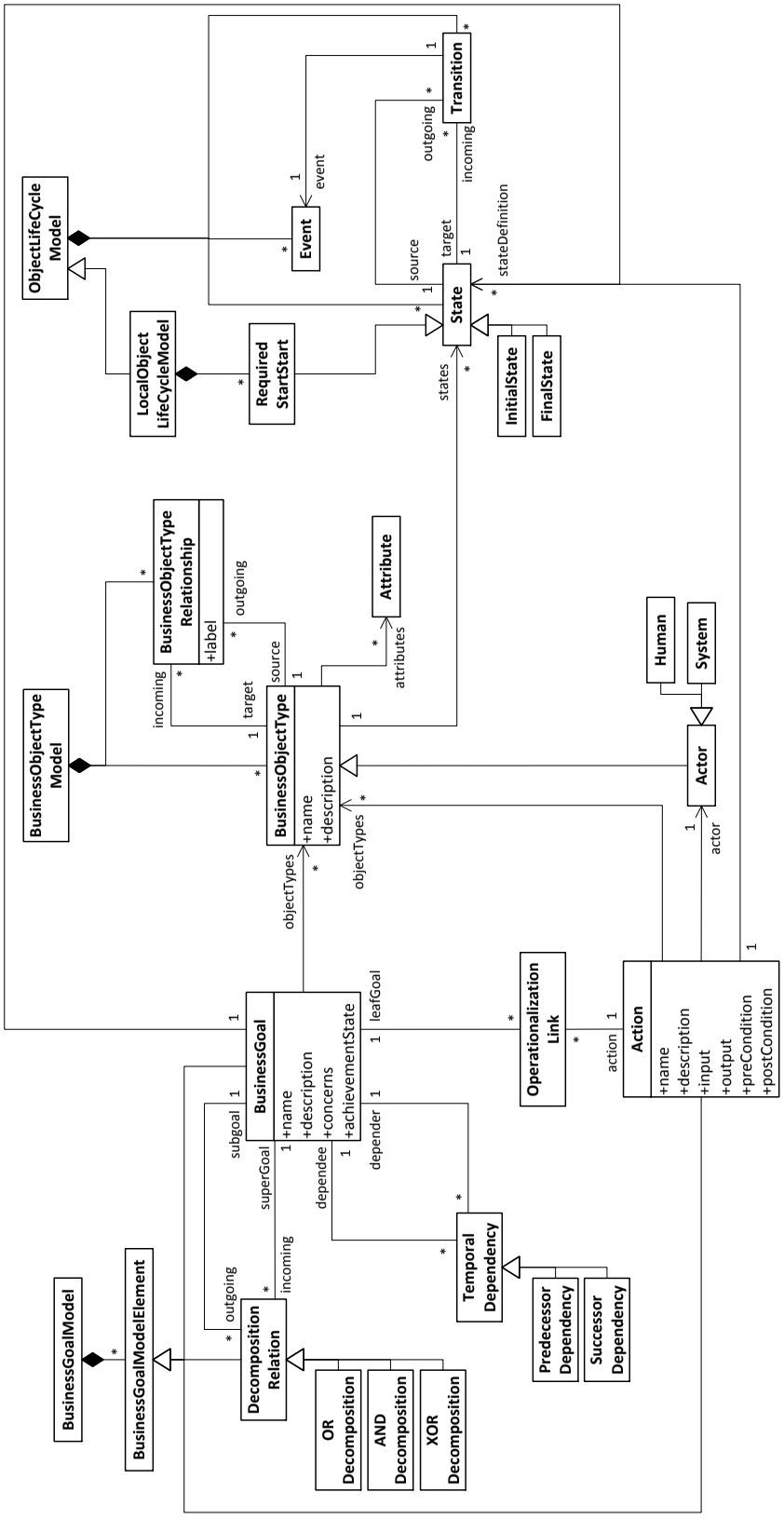


Figure 3.33: Integrated Meta-Model for our Business Goal Modeling Approach

# Chapter 4

# Derivation of Business Process Models

For the derivation of a business process model based on a specified business goal model, a composition of the identified actions needs to be calculated. To ensure consistency to the business goal model, the logical and temporal dependencies among goals need to be considered. Furthermore, the derived business process model has to achieve the addressed business goals in a sufficient manner. In this chapter, we introduce a systematic method for the goal-oriented derivation of business process models.

In Section 4.1, we elicit requirements for the derivation of business process models and evaluate existing work with respect to these requirements. Addressing the stated requirements, we give an overview about our approach in Section 4.2. In Section 4.3, we describe the identification and definition of business process fragments to provide the foundation for the business process composition. To compose the identified fragments, we describe the specification of formalized composition constraints in Section 4.4. In Section 4.5, we illustrate how the identified fragments are composed by applying the formalized constraints. Finally, we conclude this chapter in Section 4.6 and summarize the results. This chapter is partially based on the earlier publications [NGPE13a, NGE14].

## 4.1 Requirements and Related Work

In the following, we elicit requirements for the goal-oriented derivation of business process models. Based on these requirements we evaluate existing approaches with respect to their suitability and discuss the results of this analysis.

### 4.1.1 Requirements Definition

As a starting point for the analysis of requirements, we consider a given business goal specification that is defined following the notation presented in Chapter 3. Such specifications comprise a business goal model and a business context definition consisting of a business object type model and the related object life cycle models. Based on a business goal specification, the derivation step aims at the composition of a business process that sufficiently achieves the stated business goals and is consistent with respect to the restrictions and dependencies.

Figure 4.1 gives an overview about the relevant models and their interrelations that need to be considered in order to derive a consistent business process model. The resulting process derivation requirements (PDR) are discussed in the following.

The defined business goals express stakeholder objectives on different levels of abstraction. Deriving business process models from these goals aims for an operational view, described by a sequence of actions which need to be executed to achieve these goals. To validate whether a business goal is achieved, it needs to be traceable which actions or subprocesses in a business process model contribute to the achievement of a certain business goal. Establishing explicit links between related elements is the prerequisite for any kind of traceability analysis (cf. Section 2.3.2). Hence, we state the definition of traceability links and the consideration of goal achievement traceability as an integral part of the business process derivation step as requirement **PDR-01**.

- **PDR-01** Traceability of goal achievement

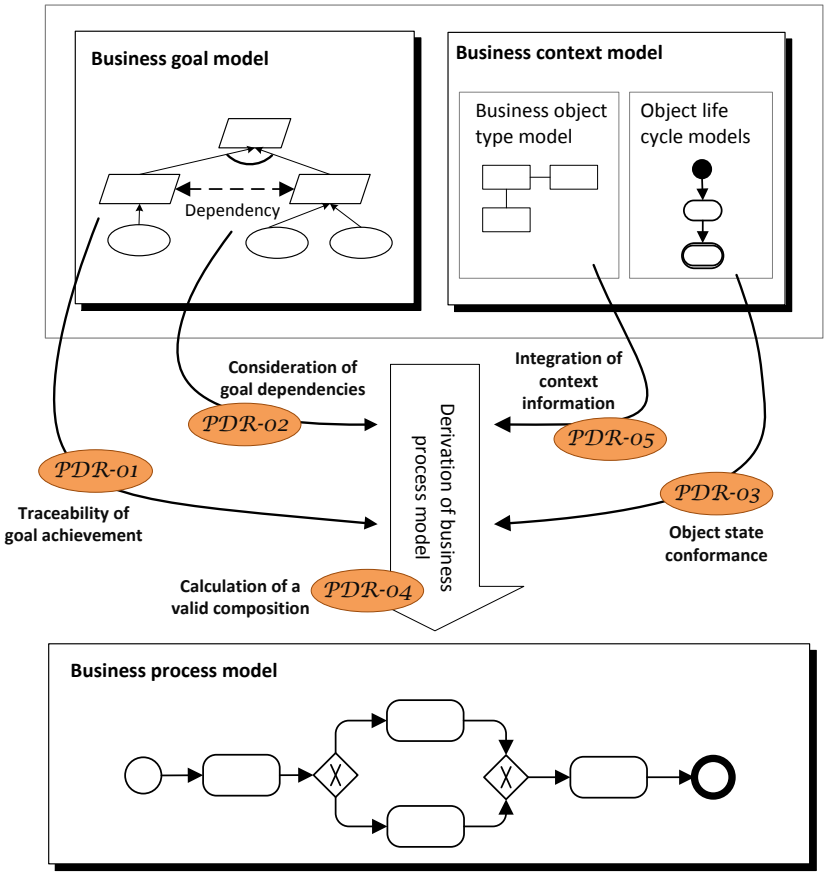


Figure 4.1: Requirements for Goal-oriented Business Process Derivation

As discussed, business goals are not independent from each other. They are related to each other in a hierarchical order expressed by decomposition links between super and sub goals in the business goal model. These decomposition relations are specified by logical operators (AND-/OR-/XOR-decomposition). For example, in an XOR-decomposition a super goal is achieved if exactly one of its subgoals is achieved. Such dependencies derived from logical operators need to be considered in the resulting business process model.

In addition to logical dependencies, we introduced the concept of temporal dependencies among business goals in Section 3.3.4. Such dependencies need to be considered in the resulting business process as well. A lack of consid-

eration can lead to inconsistencies between strategic concerns expressed by these dependencies and their realization in the modeled business processes. To avoid those inconsistencies, we state the consideration of goal dependencies in the derivation step as the second requirement.

- **PDR-02** Consideration of goal dependencies

Following our business goal modeling approach, relevant business object types are defined as part of the business context model (cf. Section 3.3.1). For each object type an object life cycle model is constructed that defines the valid object states and the transitions among them. Consistency between the business context definition and the derived business process models with respect to states and transitions is raised as the third requirement.

- **PDR-03** Object state conformance

Considering goal dependencies and object state conformance, the identified actions need to be composed to a coherent business process model. The composition has to include all required actions while fulfilling the constraints indicated by goal dependencies and object state restrictions. The resulting requirement is stated as the ability to calculate a valid composition with respect to a set of given constraints, achieving the related business goals.

- **PDR-04** Calculation of a valid composition

Each action is related to one or more business object types. This means that they are related to an actor who performs them during the business process execution and to the data objects that are processed by the execution of actions. The changes of object states are expressed by input and output objects of the corresponding actions. To ensure that this information is specified in the business process model, it needs to be considered in the derivation step. While requirement **PDR-03** addresses consistency with respect to valid states and transitions, requirement **PDR-05** addresses the integration and explicit specification of context information in the business process derivation.

- **PDR-05** Integration of context information

An overview about the resulting requirements is depicted in Table 4.1. In the following, we analyze existing approaches with respect to these requirements.

| ID | Requirement |
|---|---|
| **PDR-01** | Traceability of goal achievement |
| **PDR-02** | Consideration of goal dependencies |
| **PDR-03** | Object state conformance |
| **PDR-04** | Calculation of a valid composition |
| **PDR-05** | Integration of context information |

Table 4.1: Requirements for Goal-oriented Business Process Derivation

### 4.1.2 Evaluation of Existing Approaches

In the previous section, we elicited requirements for a suitable business process derivation approach. In the following, we discuss these requirements with regard to existing work and point out the weaknesses. As depicted in Table 4.2, our evaluation includes five composition approaches. Their suitability with respect to the stated requirements is indicated by a metric with three values: fulfilled (+), partially fulfilled (o) and not fulfilled (-).

The work of Koliadis, Ghose et.al. [KVB+06b, KVB+06a, KG06] presents an approach to explicitly relate goal-oriented requirements models and business process models. A methodology termed GoalBPM is introduced that applies informal techniques and language constructs to define relationships between elements in different model types. GoalBPM supports the definition of explicit traces, but does not provide integration into the actual business process composition step.

[KVB+06a] provides methodical guidance for the systematic enrichment of BPMN business process models with context information from a related goal model. It also supports the handling of evolutionary changes with respect to this context information. To summarize, this approach offers assistance and support for traceability and contextual consistency but does not provide any

| Approach | PDR-01 | PDR-02 | PDR-03 | PDR-04 | PDR-05 |
|---|---|---|---|---|---|
| Koliadis, Ghose et.al. [KVB+06b, KVB+06a, KG06] | o | - | - | - | + |
| Mylopoulos et.al. [LYM07, YLL+08, MPMG08] | - | o | - | o | - |
| Soffer et.al. [SW04, SW07, SGP10] | o | - | - | - | o |
| Astro project [TP04, KPR04a, TBBG07] | o | - | - | + | - |
| Oster et.al. [OSB11a, OSB11b, OAS+12] | - | o | - | + | - |

Table 4.2: Evaluation Results for Business Process Derivation

capabilities for the actual composition of business process models in a goal-oriented manner.

Mylopoulos et.al. describe the derivation of more concrete specification models from Tropos goal models. The general approach presented in [YLL+08, MPMG08] addresses the engineering of late requirements and system design models from goal-oriented requirement specifications. Based on a set of patterns, structural component diagrams as well as behavioral models represented as state charts can be derived. These patterns are used to consider the logical relationships among goals (AND-/OR-decompositions). In [LYM07], this approach has been applied to the generation of business process models. The presented solution enables the semi-automated generation of BPEL process models from a given Tropos goal model.

Soffer et.al. [SW04, SW07, SGP10] present a formal process modeling framework termed Generic Process Model (GPM). Traceability can only be provided on a high level of abstraction by relating whole business processes to goals. The framework provides support for the identification of relevant context elements and the enrichment of business processes with these information. Although, this approach provides a solid theoretical framework it lacks methodical guidance in terms of concrete algorithms for the composition of business process models.

In the Astro research project an approach for the composition of business pro-

cesses has been developed [TP04, KPR04a, TBBG07]. Following the approach presented there, atomic actions can be composed in an automated manner. Dependencies between goals and consideration of context information are not supported by this approach.

Oster et.al. propose a goal-oriented service composition framework [OSB11a, OSB11b, OAS+12]. This framework enables the automated calculation of valid compositions taking into account logical relations between goals. Traceability is not considered explicitly and the enrichment of the resulting composition with contextual information is not addressed sufficiently.

### 4.1.3 Discussion of Evaluation Results

To summarize, the analysis of related work shows that none of the investigated approaches fulfills all stated requirements for the goal-oriented derivation of business process models. Requirement PDR-01 describes the definition of explicit traces between elements in the business goal and business process model. The investigated approaches do not provide a sufficient solution enabling the definition of such traces as an integral part of the derivation step.

The consideration of goal dependencies, especially with respect to temporal dependencies, is not supported sufficiently and object state conformance is not addressed at all. Existing approaches lack the ability to derive formalized, verifiable constraints from a set of defined dependencies to enable their validation during the business process composition.

The calculation of valid compositions based on a set of given constraints is supported by two existing approaches (Astro project, Oster et.al.). The algorithms for the compositions can be applied, extended and integrated for the purpose of the goal-oriented business process composition. Nonetheless, the specification of constraints expressing goal dependencies which need to be considered is not supported by these approaches.

The topic of context information and its consideration in a business process model is addressed in the approach of Koliadis, Ghose et.al. Concerned object types of goals are related to elements in the derived business process model. However, it does not describe how this information (especially information about object states) is represented explicitly in the business process model.

Addressing the identified shortcomings, the next section provides an overview about our approach that fulfills the stated requirements and is built upon the business goal modeling approach presented above.

## 4.2 Derivation Approach Overview

In this section, we give an overview about our approach for the goal-oriented derivation of business process models. Addressing the requirements defined in the previous section our approach enables the composition of business processes based on a given business goal specification (defined according to the specification in Chapter 3). Figure 4.2 illustrates our approach, emphasizes its contributions and sketches the resulting structure for the remainder of this chapter.

The first requirement addresses the explicit consideration and integration of traceability in the derivation step. Therefore, related elements in the business goal and the business process model need to be connected by traces. To define such traces, elements or groups of business process elements need to be identified and clustered in order to specify their contribution to the achievement of a certain goal. For this purpose, we apply the concept of business process fragments. These fragments define a subgraph of the business process model which is related to a business goal. In Section 4.3, we describe how business process fragments can be derived from a given business goal model. Using the identified fragments, explicit traceability links to the related goals are created to fulfill requirement PDR-01.

In the next step, business process fragments and their explicit relations to business goals are used for the consideration of goal dependencies (PDR-02) and the assurance of object state conformance (PDR-03). Dependencies among goals can be defined on different levels of abstraction and are specified in an informal manner. We present a systematic approach to formalize them to verifiable constraints and show how these constraints can be refined to the level of business process fragments. Constraints for valid states and state transitions are refined to business process fragment accordingly. The individual steps for the specification of formalized composition constraints are described in Section 4.4.

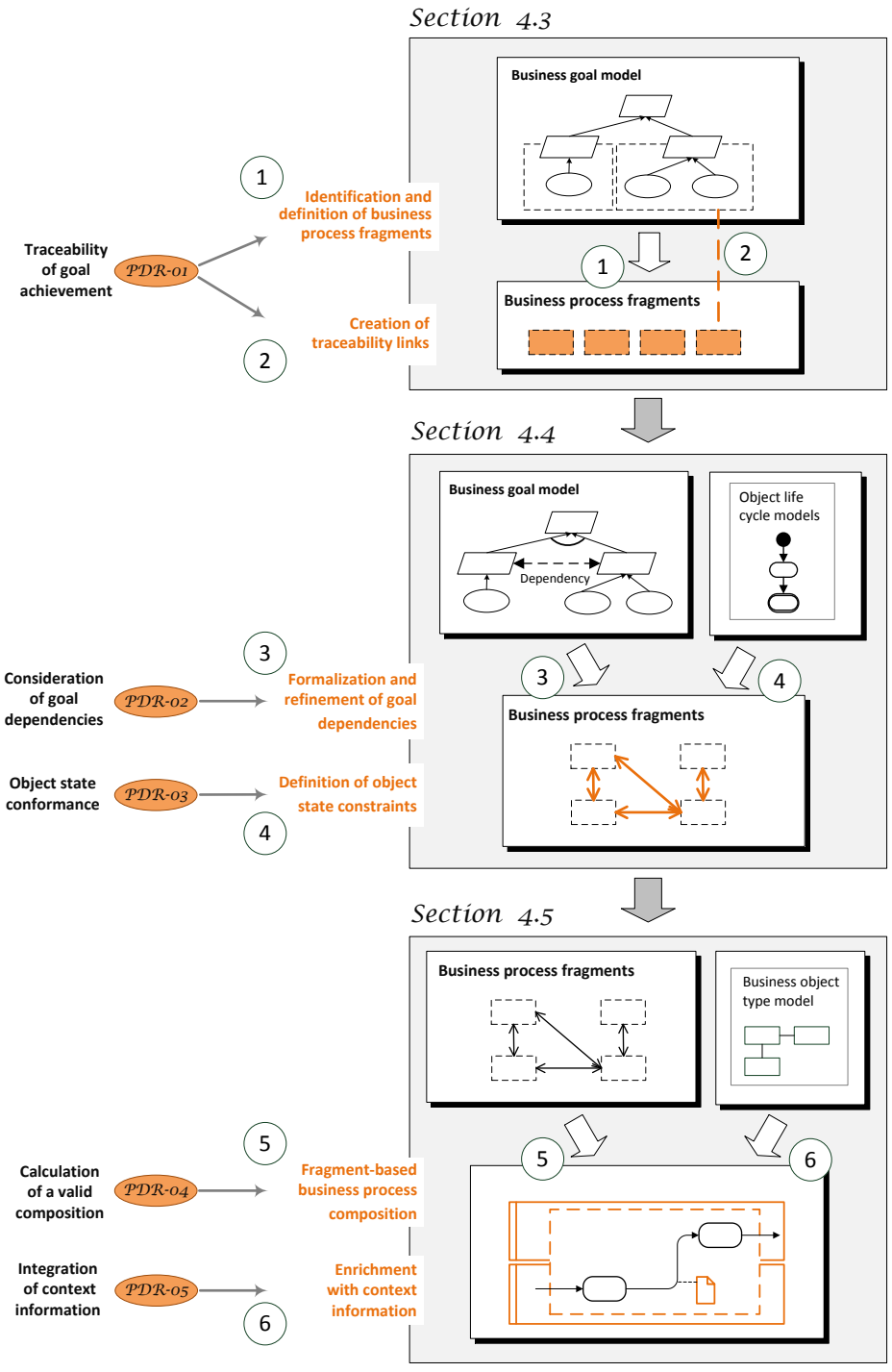


Figure 4.2: Business Process Derivation Approach

Section 4.5 describes the calculation of valid fragment compositions that are consistent to the defined constraints (PDR-04). For this purpose, we integrate an existing algorithm for service compositions and apply it to our fragment-based approach. Finally, the resulting business process model is enriched with context information described in the business object type model (PDR-05).

Addressing the stated requirements, our approach enables the systematic and semi-automated derivation of a valid and traceable business process model which is consistent with respect to the dependencies specified in the business goal model. In the following sections, the different steps of our approach are described in detail.

## 4.3 Definition of Traceable Business Process Fragments

The consideration of traceability as an integral part of the business process composition has been identified as a crucial requirement. To enable the analysis of traceability, the contribution relations between elements in the business process model and addressed business goals need to be expressed by explicit links. For this purpose, we need to identify collections of related actions and compose them in distinct groups which can be related to goals. Such groups of elements in a business process model can be defined in subgraphs termed business process fragments [JPP94]. To derive a coherent business process model, the identified business process fragments are composed to a business process in a subsequent step.

The application of business process fragments for the calculation of valid compositions is also feasible with respect to performance concerns. The decomposition of a business process model into fragments enables the expression of constraints based on these fragments, not on the level of atomic actions. By using a higher level of abstraction, the validation of the constraints is performed on a smaller number of elements. Hence, the computational complexity of the validation can be reduced.

In this section, we introduce an approach to derive business process fragments from a given business goal model in a traceable manner. Figure 4.3 depicts an overview about the different steps which are described in the following.

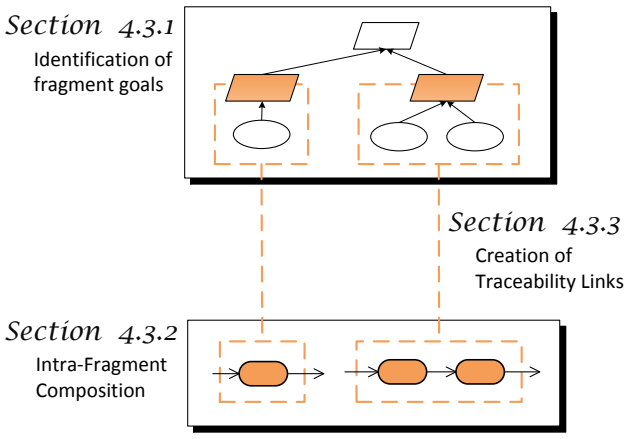


Figure 4.3: Steps for the Definition of Traceable Business Process Fragments

### 4.3.1 Identification of Fragment Goals

A prerequisite for the definition of business process fragments is the identification of the business goal that is achieved by the fragment. We term such business goals *fragment goals*. The definition of very abstract fragment goals can result in complex fragments that comprise a large number of actions. We consider all leaf goals as fragment goals. This selection has two advantages.

First, the set of actions composed within the fragment can be identified in an easy and unambiguous manner. Logical or temporal dependencies are defined between goals but not between a leaf goal and its actions or between different actions. Hence, such dependencies do not need to be considered in the action composition within a fragment. Second, by selecting all leaf goals as fragment goals, it can be ensured that all actions are considered in the resulting composition. We iterate through the business goal model and mark each leaf goal as a fragment goal. To illustrate, we apply this to the running example depicted in Figure 3.12. Table 4.3 shows the resulting list of fragment goals.

According to the construction approach for lOLMs (cf. Section 3.5.1) each leaf goal is related to exactly one lOLM. These object life cycle models can be applied in different ways. They can be used to calculate a valid composition of the actions according to the defined states and state transitions. Furthermore, the object state conformance can be validated with respect to these models as

| Fragment goal ID | Name of fragment goal |
|---|---|
| $fg_A$ | Books ordered |
| $fg_B$ | Books acquired |
| $fg_C$ | Books supplied |
| $fg_D$ | Books delivered |
| $fg_E$ | Payment via credit card |
| $fg_F$ | Payment via bank transfer |
| $fg_G$ | Printed receipt sent |
| $fg_H$ | Electronic receipt sent |

Table 4.3: Identified Fragment Goals

well. Each fragment goal is related to a set of actions by its operationalization links. In order to define a valid execution order of actions within a business process fragment, the next section describes the composition based on these actions.

### 4.3.2 Intra-Fragment Composition

Previously, we described the purpose of fragment goals and their identification in a business goal model. In this step, a business process fragment is derived for each fragment goal by composing the actions that operationalize the fragment goal. For the definition of business process fragments, we leverage the concept of single-entry-single-exit (SESE) fragments. Originally, SESE fragments are known from the field of compiler theory [JPP93, JPP94]. In [VVL07] SESE fragments have been applied to business process models. This approach uses SESE fragments to check the soundness of workflow graphs. To precisely state our understanding of SESE fragments, we apply the following definition.

**Definition 7 (Single-Entry-Single-Exit Fragment (SESE Fragment))** *Given a business process model $\mathcal{V}$ with distinguished nodes $Initial$ and $Final$, such that every node is on a path from $Initial$ to $Final$. Two distinct nodes $a$ and $b$ in $\mathcal{V}$ enclose a SESE fragment, if*

- *node $a$ dominates node $b$, i.e. every path from $Initial$ to $b$ includes $a$,*
- *node $b$ postdominates node $a$, i.e. every path from $a$ to $Final$ includes $b$, and*

- *every cycle containing node a also contains node b and vice versa.*

*(based on [Ger13, JPP94])*

Examples of SESE fragments in business process models are illustrated in Figure 4.4. To represent business process models, we leverage the BPMN notation [OMG11a]. Following the visualization proposed in [VVL07], the fragments are depicted as dotted boxes. The example includes six actions and three fragments. Fragment $f_A$ illustrates a simple fragment that encapsulates one action. A fragment can also include multiple actions, as illustrated by fragments $f_B$ and $f_C$. Business process fragments can be disjointed, like $f_B$ and $f_C$ or could also be nested following the definition in [JPP94].

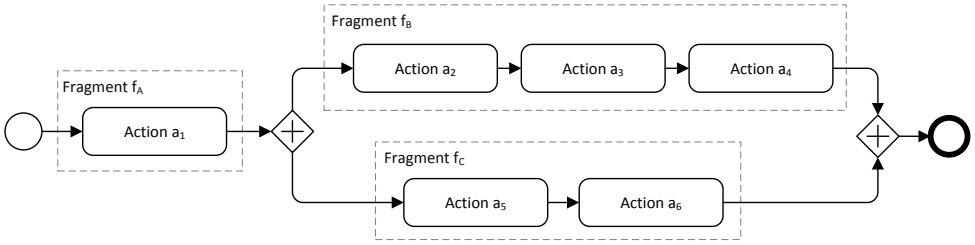


Figure 4.4: Notation for SESE Fragments in Business Process Models

The concept of SESE fragments can be applied to our approach, since all actions composed in a fragment operationalize the same leaf goal. Following the semantics of operationalization links, all actions need to be executed in order to achieve the assigned goal. For this purpose, the actions are composed in a sequence. Hence, the action composition always has exactly one incoming (single-entry) and one outgoing (single-exit) transition. While existing approaches use SESE fragments to decompose business process models, our approach applies SESE fragments for the systematic derivation of business process models. To define the actual control flow, actions within the fragments need to be composed.

To provide methodical support, Algorithm 3 describes the single steps for the creation of a valid action composition within a business process fragment. The algorithm is executed for all identified fragment goals. First, a business process fragment is created for the considered fragment goal. To provide trace-

ability between both modeling elements, a traceability link is created among them. A detailed description of the traceability link creation is given in Section 4.3.3.

**Algorithm 3** Business Process Fragment Definition

**Precondition:** Fragment goal $fg$, lOLM $\mathcal{J}$ for fragment goal $fg$
**Postcondition:** Business process fragment $f$, traceability link $l$

```
f ← createNewFragment()
l ← createNewTraceabilityLink(f, fg)
f.setCurrentAction(null)
t ← J.getFirstLabeledTransition()
while t ≠ null do
    a ← fg.getAction(t.getLabel())
    f.assign(a)
    if f.getCurrentAction() ≠ null then
        f.createEdge(f.getCurrentAction(), a)
    f.setCurrentAction(a)
    t ← J.getNextTransition()
```

To derive a valid order of actions, the algorithm uses the state transitions defined in the lOLM. Each transition is related to the corresponding action. By following these transitions the matching actions can be selected. If the fragment already comprises actions, the new action is added and an edge is created to the latest assigned action resulting in an updated, coherent action composition. These steps are repeated until all transitions of the lOLM are processed. An example for intra-fragment composition is sketched in Figure 4.5.

By applying the lOLMs for the calculation of action compositions, we ensure that a business process which composes these fragments is consistent to the lOLMs. Further, the resulting composition is consistent with respect to pre- and postconditions of the atomic actions. The outcome of this step is a set of business process fragments. Each fragment defines a composition of actions that is consistent with the related fragment goal and its actions.

### 4.3.3 Creation of Traceability Links

In this section, we describe the creation of explicit traceability links between fragment goals and the related business process fragments. To express such traceability links, relations between business goals and business process frag-

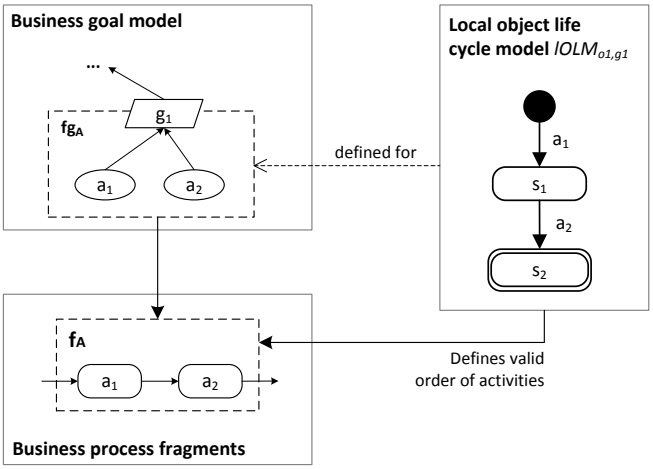


Figure 4.5: Intra-Fragment Composition

ments are specified. Figure 4.6 depicts a formal language definition for the specification of traceability links based on the introduced business goal meta model and an excerpt from a business process meta model. According to [Poh10], a traceability model comprises artifacts and traceability relationships among them.

We distinguish fragment goals and business process fragments as the modeling elements that shall be traced. To connect fragment goals and business process fragments, satisfiability links are established. The 1:1 cardinality is a valid restriction, since each fragment goal is related to one leaf goal from which exactly one fragment is derived.

In the concrete syntax, satisfiability links between a fragment goal $fg_A$ and a business process fragment $f_A$ are represented by an arrow labeled with *satisfies*. This means, a business process fragment $f_A$ satisfies a fragment goal $fg_A$. Exemplary illustrations of these links are depicted in Figure 4.7.

## 4.4 Specification of Formalized Composition Constraints

In the previous section, we described the identification and definition of business process fragments from a given business goal model. These fragments provide the foundation for the composition of a business process model. To

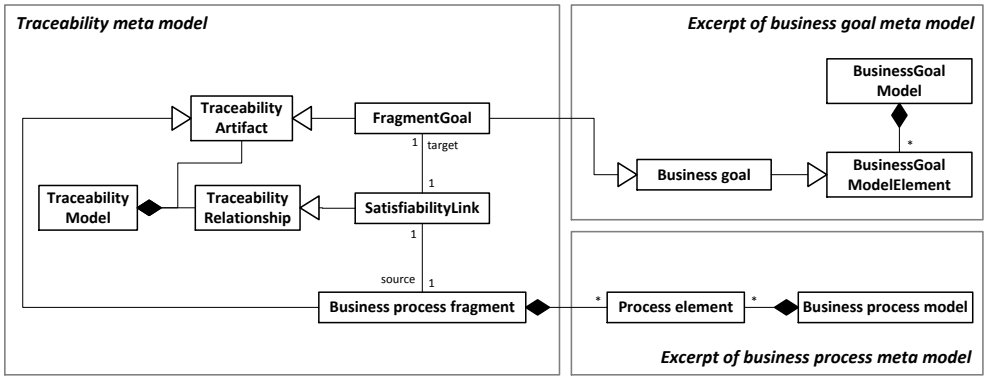


Figure 4.6: Traceability Meta Model

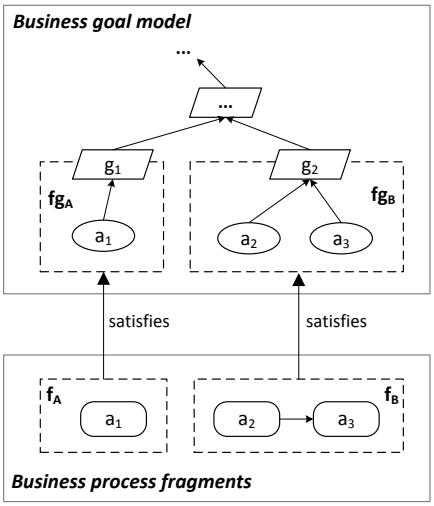


Figure 4.7: Traceability Links between Fragment Goals and Business Process Fragments

ensure consistency between the derived business process model and the business goal model, dependencies among business goals need to be considered.

The consideration of such constraints raises two prerequisites. First, the informal definition of dependencies needs to be translated into a formal representation that enables their automated validation. Second, the dependencies are defined between business goals on different levels of abstraction. To make them applicable in the composition step, the formalized constraints need to be refined to the level of fragment goals.

### 4.4.1 Formalization of Hierarchical Dependencies

Business goal models specify a hierarchy of super and subgoals described by decomposition links. These decomposition links are related to a logical operator (AND, OR, XOR) that defines how the subgoals contribute to the achievement of the super goal. To ensure that the derived business process is consistent with the business goal model, the hierarchical order and the corresponding logical operators need to be considered. Existing composition algorithms (cf. Section 4.5) support the consideration of AND-/OR-decomposition links. The introduced concept of XOR-decompositions and the resulting constraints are not supported by these algorithms. Therefore, dependencies indicated by a XOR-decomposition need to be formalized to verifiable constraints.

Illustrating the formalization, we consider the example depicted in Figure 4.8. Two goals $g_1$ and $g_2$ decompose a common super goal with an XOR operator. The resulting constraint needs to express that either $g_1$ or $g_2$ has to be achieved. To express this dependency in a formalized way the XOR-decomposition of $g_1$ and $g_2$ is expressed by the constraint $(g_1 \wedge \neg g_2) \vee (\neg g_1 \wedge g_2)$. If an XOR decomposition comprises more than two subgoals a constraint for *n* goals is defined as follows:

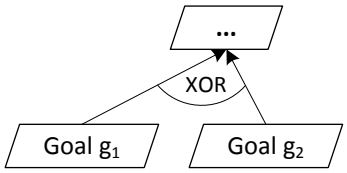


Figure 4.8: Hierarchical Dependencies with an Exclusive-OR Operator

$$(g_1 \wedge \neg g_2 \wedge .. \wedge \neg g_n) \vee (\neg g_1 \wedge g_2 \wedge .. \wedge \neg g_n) .. \vee (\neg g_1 \wedge \neg g_2 \wedge .. \wedge g_n)$$

### 4.4.2 Formalization of Temporal Dependencies

Our approach for business goal modeling does not only support the definition of logical relations but also considers the explicit definition of temporal dependencies. These dependencies define constraints for the order in which the business goals shall be achieved. Hence, the temporal dependencies need to be considered during the composition and it needs to be verifiable whether a derived business process model is in line with these dependencies. In this section, we describe the formalization of temporal order dependencies to ver-

ifiable constraints which can be processed automatically in the business process composition.

Temporal dependencies are expressed by annotations of a business goal. In this representation the temporal dependencies are only "visible" for the business goal to which the dependency is related. To make such dependencies explicit by considering both dependent goals, we apply the concept of dependency modeling [Yu96]. A temporal dependency among two business goals is defined by a *Depender*, *Dependum* and *Dependee*. A *Depender* is dependent on the *Dependee* with respect to the *Dependum*. This means, *Depender* and *Dependee* are two business goals and the *Dependum* is the temporal dependency among them. Accordingly, a temporal dependency between two business goals $g_1$, $g_2$ can be defined by a 3-tuple < $g_1$, $\delta$, $g_2$ > with $\delta \in \{hasPredecessor, hasSuccessor\}$.

All temporal dependencies in a business goal model can be expressed by such 3-tuples. To validate these dependencies on a business process model, a formalized and verifiable representation is required. For this purpose, we infer CTL constraints from the stated dependency tuples. The *hasSuccessor* dependency can be expressed by using standard CTL constructs [CES86, Eme90]. Hence, the dependency tuple $g_1, hasSuccessor, g_2$ is translated to $AG(g_1 \rightarrow AF(g_2))$. The expression of past-time constraints is not considered in CTL. Different existing approaches [BC03, MNP05] propose new notations to express constraints about previous execution paths. The main disadvantage of these notations is the lack of support by existing model-checking tools and composition algorithms. Therefore, we apply a standard CTL formula and translate the tuple < $g_1, hasPredecessor, g_2$ > to $A(\neg[\neg g_2 U g_1])$. The equivalence between the past formula and the standard CTL constraint is discussed in [LS95].

To summarize, the rules for the translation of dependency tuples to formalized CTL formulas are depicted in Table 4.4. These rules are applied to the defined dependency tuples. By providing a formalized representation in terms of a CTL formula, the temporal dependencies between goals are verifiable in an automated manner.

| **Temporal dependency** | **Formalized CTL constraint** |
|---|---|
| $< g_1, hasPredecessor, g_2 >$ | $A(\neg[\neg g_2 U g_1])$ |
| $< g_1, hasSuccessor, g_2 >$ | $AG(g_1 \rightarrow AF(g_2))$ |

Table 4.4: Translation of Temporal Order Dependencies to Formalized CTL Constraints

### 4.4.3 Refinement of Constraints

Previously, we presented the identification of fragment goals and the definition of related business process fragments which provide the foundation for the composition of the business process model. In addition, we have described the formalization of hierarchical and temporal dependencies. As discussed, the business process model is composed based on the defined fragments. To enable their consideration in the fragment composition, these constraints are refined to the abstraction level of these fragments.

In this refinement the decomposition hierarchies of goals and subgoals are considered. To illustrate the refinement of a formalized constraint according to the logical operators (AND, OR, XOR), we consider a given constraint $AG(g_1 \rightarrow AF(g_2))$. Further, we assume that goal $g_1$ is decomposed to goals $g_3$ and $g_4$.

Table 4.5 provides an overview about the different refinement scenarios. To express the execution of a refinement, we define the `Replace`($exp_1$, $exp_2$) operation. It replaces each occurrence of $exp_1$ by the expression $exp_2$ in a given CTL constraint.

For an AND decomposition the refinement results in the constraint $AG((g_3 \wedge g_4) \rightarrow AF(g_2))$. If goal $g_1$ is OR-decomposed the refined constraint is $AG((g_2 \vee g_3) \rightarrow AF(g_2))$. The most complex refinement is performed for a XOR decomposition. In this case, the constraint is refined as follows: $AG(((g_3 \wedge \neg g_4) \vee (\neg g_3 \wedge g_4)) \rightarrow AF(g_2))$.

The described refinement is performed for each goal in the constraint until a fragment goal is reached. The refinement is performed by iterating through all formalized CTL constraints. The iteration results in a set of constraints,

| **Decomposition of business goal $g_1$** | **Refinement of formalized constraint** |
|---|---|
| $g_3$ AND $g_4$ | `Replace(` $g_1$, $(g_3 \wedge g_4)$ ) |
| $g_3$ OR $g_4$ | `Replace(` $g_1$, $(g_3 \vee g_4)$ ) |
| $g_3$ XOR $g_4$ | `Replace(` $g_1$, $((g_3 \wedge \neg g_4) \vee (\neg g_3 \wedge g_4)))$ |

Table 4.5: Refinement of Formalized Constraints according to Logical Decomposition Operators

defined on the level of fragment goals.

### 4.4.4 Formalization of Object Life Cycle Constraints

The explicit consideration of object state conformance has been identified as a requirement for the business process derivation. This conformance is validated based on the states and state transitions defined in the object life cycle models. To ensure the object state conformance of the derived business process model, explicit constraints are inferred that can be verified in an automated manner. Furthermore, the formalized constraints are defined based on the identified fragments to make them applicable in the fragment composition.

Based on restrictions for states and valid transitions, we infer constraints for the order of fragments in the composition. To define these constraints for fragments, we take advantage of the relation between the identified fragments, the related fragment goal and the defined lOLM for this goal. Hereby, we can relate each fragment to a lOLM in a 1:1 relation. To restrict valid orders of lOLMs, we do not consider the single states but the glued lOLMs as illustrated in Figure 4.9. Order constraints derived from this abstract view can be related to fragments due to the 1:1 relation between lOLMs and fragments.

To infer formalized constraints from OLMs we consider four different scenarios. To apply these scenarios, we iterate through all elements in an OLM and use the matching scenario to create a formalized CTL constraint. The first scenario is applied for the initial state in an OLM. The handling of initial states

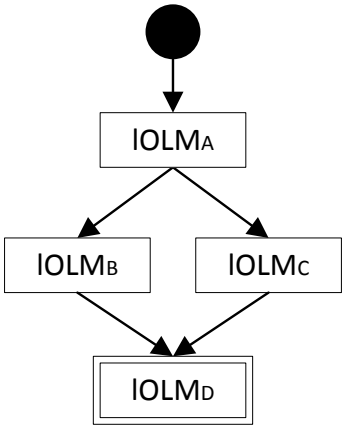


Figure 4.9: Abstracted View on an Object Life Cycle Model

in OLMs is illustrated in 4.10. In this case, it is not useful to define a constraint based on the initial state, since this is not considered in the following derivation of a business process.

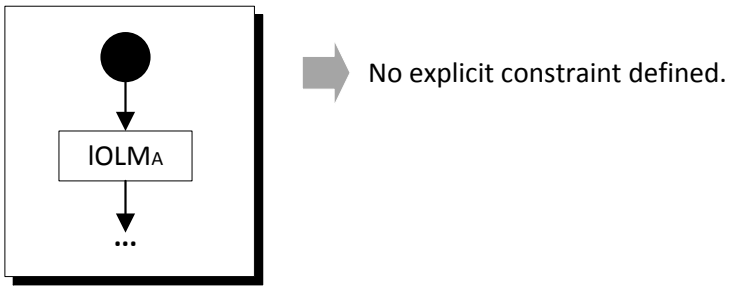


Figure 4.10: Scenario 1. Initial State

The basic structure of scenario 2 is depicted in Figure 4.11. This rule is applied for a lOLM ($lOLM_A$) which has exactly one outgoing transition to another lOLM ($lOLM_B$). In this scenario, the inferred constraint expresses that directly after the states in $lOLM_A$ the states in $lOLM_B$ have to be achieved. This constraint is formalized in CTL as follows $AG(lOLM_A \rightarrow AX(lOLM_B))$.

Scenario 3 describes a slightly different scenario. We consider a lOLM ($lOLM_A$) that is followed by more than one subsequent lOLM. An example is sketched in Figure 4.12. The example considers that $lOLM_A$ is followed by two lOLMs $lOLM_B$ and $lOLM_C$. The resulting constraint is formalized by the CTL statement $AG(lOLM_A \rightarrow AX(lOLM_B \vee lOLM_C))$.

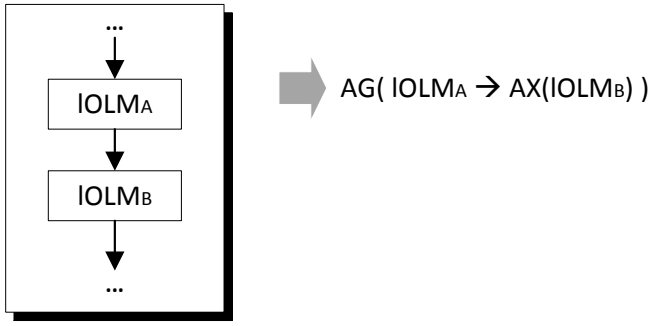


Figure 4.11: Scenario 2. Intermediate State 1:1


Figure 4.12: Scenario 3. Intermediate State 1:N

Scenario 4 is applied to the final state in an OLM. Figure 4.13 depicts an example for scenario 4 by considering a $lOLM_A$ as the final state. Since no further state can follow the final state, there are no dependencies to other lOLMs. Accordingly, no explicit constraint definition is required.

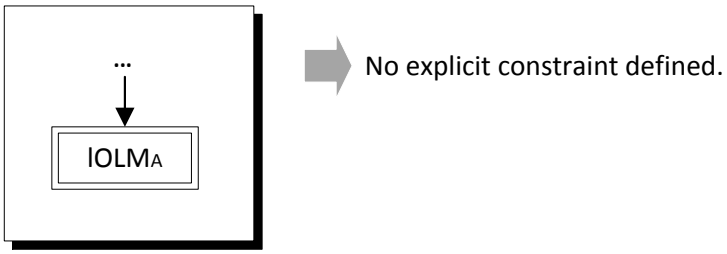


Figure 4.13: Scenario 4. Final State

The result is a set of formalized CTL constraints defined on lOLMs. To make them applicable to the business process derivation, they need to restrict the

composition of business process fragments. Each lOLM is related to a fragment goal, which is connected to a business process fragment by a traceability link as defined in Section 4.3.3. Following these links, a lOLM can be related to a business process fragment. To define suitable composition constraints, the lOLM is replaced in the CTL constraint by the corresponding business process fragment reference by using the introduced `Replace`($lOLM$, $f$) operation.

Using the defined constraints, the next section describes how they are used to compose business process fragments to a consistent business process model.

## 4.5 Composition of Business Process Model

In this section, we describe the composition of business process models based on the elicited business goal model and the inferred constraints. In contrast to existing approaches which focus on the composition of atomic services, we propose an approach that enables the composition of business process fragments.

### 4.5.1 Definition of Fragment Composition Problem

The calculation of a fragment composition that achieves the overall business goal and also fulfills the given constraints is the main challenge of the business process derivation. To scope this task we specify the underlying challenge of the composition. In line with the service composition problem described in [OAS+12], we define the composition problem as follows.

> We consider a business goal model $\mathcal{G}$ with top goal $g_{top}$, a set of identified business process fragments $F$ and a set of constraints $\Psi$. We need to identify a composition $C$ of fragments $\in F$ that achieves goal $g_{top}$ and satisfies all constraints in $\Psi$.

Following our approach to the goal-oriented modeling of business requirements, constraints are not defined on the level of atomic actions. Instead, we apply the concept of business process fragments in order to define con-

straints on a more abstract level in order to reduce the computational complexity. The derivation of a business process that is consistent with respect to the stated constraints requires the composition of the identified fragments fulfilling these constraints. Enabling the fragment-oriented composition, the next section introduces an approach based on model-checking techniques.

### 4.5.2 Business Process Fragment Composition

To solve the stated composition problem, we discuss two possible solution strategies. First, an assisted approach is described which comprises the manual composition of fragments and an automated validation of the defined constraints. This approach provides a high degree of modeling flexibility and can also be used to validate the consistency of an evolved business process model.

Second, we apply an existing approach that enables the automated calculation of valid compositions based on a set of behavioral constraints [OSB11b, ARWB11, OAS+12]. We apply this technique and describe how it can be adopted and integrated into our approach for the fragment-based composition. While the actual composition is done automatically, it is not possible to restrict the way, the fragments are composed. Further, this approach requires a manual translation of the business process fragment specification into formalized Kripke Structures (KS) [Sin09].

#### Assisted Composition

The assisted composition provides a high degree of flexibility for the business process engineer. First, the fragments are composed manually, by connecting the first and last actions of consecutive fragments. To facilitate the correct composition, fragments dealing with same object types can be composed according to the order in the defined object life cycle model.

In the next step, the composed business process model is validated against the defined constraints. To enable the automated validation of constraints, the business process model needs to be available in a formal representation with uniquely defined semantics. For that purpose, a labeled transition system (LTS) is generated that represents all potential execution paths of the composed business process model. For its generation, we rely on the Dynamic

Meta-Modeling (DMM) Framework developed at the University of Paderborn [EHHS00]. With DMM the semantics of modeling languages can be described visually and at the same time precisely in terms of graph transformations. Exemplary semantic specifications have been done successfully for a wide range of modeling languages, e.g. UML 2.0 Activity Diagrams [FESVDS07]. The advantage is that the semantics of a language, for example BPMN, need to be described only once and all models defined in the same language can be translated accordingly.

As depicted in Figure 4.14 the constraints are verified on the LTS of the business process. For this purpose, a model checker (e.g. NuSMV[1]) is used. If a business process model does not fulfill the stated constraints, the model checking results in a counter example [FESVDS07].

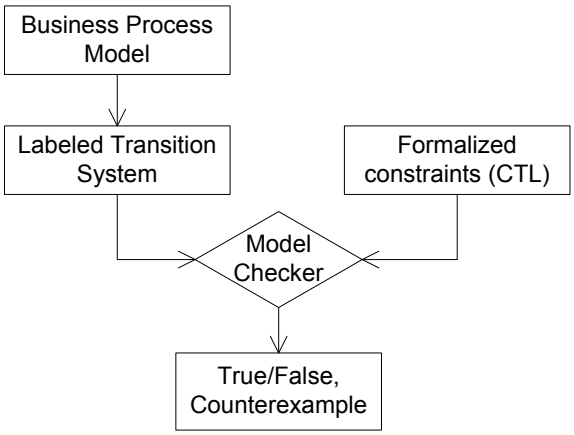


Figure 4.14: Model-checking of Business Process Model

The presented technique for the assisted composition can also be used to validate the consistency of an evolved business process model. An additional validation whether the composed business process sufficiently achieves the related business goals is described by the traceability analysis in Section 5.3.

The main disadvantage of this approach is the lack of verification whether a valid composition exists. Therefore, we present a second approach which enables the automated calculation of a business process model, including a

[1] http://nusmv.fbk.eu/

validation of whether it is possible to compose them in a consistent manner.

**Automated Calculation of Composition**

To solve the stated composition problem in an automated manner, we apply an existing approach for the calculation of valid service compositions based on a set of behavioral constraints [OSB11b, ARWB11, OAS+12]. As input, the algorithm uses a set of fragments $F$ which is given by the identified fragments. Behavioral constraints indicated by the temporal dependencies among goals and the dependencies derived from the object life cycle models are considered by a set of formalized CTL constraints $\Psi$.

The algorithm requires a formalized representation of fragments in terms of Kripke Structures (KS). These models are widely used to describe system behavior and to validate it by model-checking techniques. Based on [CGP99] a Kripke structure (KS) is defined as follows.

**Definition 8 (Kripke Structure (KS))** *Let $AP$ be a set of atomic propositions. A Kripke Structure $KS$ over $AP$ is a tuple $KS = (S, S_0, R, L)$ where*

- *$S$ is a finite set of states.*
- *$S_0 \subseteq S$ is a set of initial states.*
- *$R \subseteq S \times S$ is a transition relation that must be total, that is, for every state $s \in S$ there is a state $s' \in S$ such that $R(s, s')$.*
- *$L : S \rightarrow 2^{AP}$ is a function that labels each state with the set of atomic propositions true in that state.*

In our application scenario, the atomic propositions are given by the different states of a certain object type. If an object is in the specific state this atomic proposition is considered as *true*, otherwise as *false*. Accordingly, the different states $s \in S$ in the KS are labeled with an understandable description. The different states and transitions among them can be derived from the given business process fragment and the related object life cycle model. A detailed description for a systematic translation to KS is provided in [DNV90].

Figure 4.15 sketches an overview about our fragment composition approach.

To ensure that the resulting composition achieves the stated business goals and considers the logical operators in the decomposition hierarchy, the overall functional requirement $\theta$ is defined by the aggregated achievement state of the top goal.

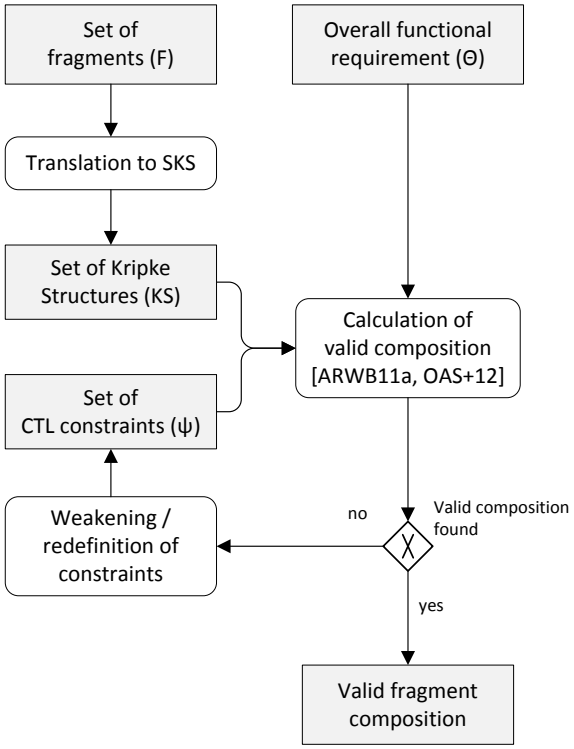


Figure 4.15: Composition Approach for Business Process Fragments

The approach applied to the automated calculation of a valid composition is based on a tableau-based algorithm [Cle90]. By combining the application of tableaus and model-checking techniques it is analyzed whether a composition $C$ exists that is valid with respect to the stated constraints $\Psi$ and fulfills the aggregated requirement defined in $\theta$. By investigating the complete state space a path representing the order in which the different states are achieved is calculated. If a path is found, the algorithm returns a valid composition.

Depending on the given constraints, it is possible that no valid composition can be generated. To resolve such contradictions and to enable a valid composition, the constraints need to be weakened. This can be done by deleting temporal dependencies or by the redefinition of inputs and outputs of actions in order to resolve context dependencies among the fragments. Based on this

redefinition, an updated set of constraints is generated and the composition algorithm is restarted.

In the applied algorithm, OR-decompositions are interpreted as a design-time decision. This means that an OR-decomposition between two goals $g_1$ and $g_2$ is fulfilled if the derived business process model considers only one of these goals. It is important to notice that the automated composition returns a valid composition but not necessarily an optimal or preferred one. The synchronization of actions or the consideration of all fragments in an OR-decomposition might be useful depending on the given business scenario. For these reasons, a manual adaption of the calculated composition might be necessary. The consistency of the changed business process model with respect to the given constraints can be validated by the approach described in assisted composition.

A detailed discussion of the complexity of the composition algorithm is given in [OAS+12]. We consider $n$ as the number of atomic functional requirements in the aggregated requirement $\theta$ and $k$ as the number of maximum elements that need to be composed. The complexity of a valid composition calculation is $O(k^n 2^n 2^{|\Psi|})$. Since our approach uses the identified fragments for the composition instead of atomic actions, the computational complexity can be reduced. Due to the exponential growth of the computational complexity, the reduction of the number of elements $k$ has a significant effect. In the worst case, the number of elements remains the same if each fragment contains exactly one action.

### 4.5.3 Enrichment with Context Information

To enrich the composed business process model with additional context data, we use the information provided by the action definitions and the object type model. Hereby, we aim for a better and deeper understanding of the involved actors and the handled artifacts. Further, these annotations are used for the traceability analysis in Section 5.3.

The object flow and the resulting states are represented in the business process model by the notation `objectType[objectState]` [KRG07]. In the first step, *PreCondition* and *PostCondition* of the actions are added in the business process model. The performed intra-fragment composition (cf. Section 4.3.2) ensures

that the pre- and postconditions of consecutive actions match within a fragment. Further, the composition algorithm described in the previous section excludes mismatches between pre- and postconditions between fragments. If such mismatches occur during a manual adaption of the business process model, the corrupted model needs to be corrected, e.g. by a re-composition of the actions, before the context enrichment step can be finalized.

Nonetheless, not every action has an explicit precondition that needs to be fulfilled. Figure 4.16 sketches an example. *Authorize credit card payment* and *Instruct bank transfer* are marked by the label "—" which indicates that these actions have no explicit preconditions.

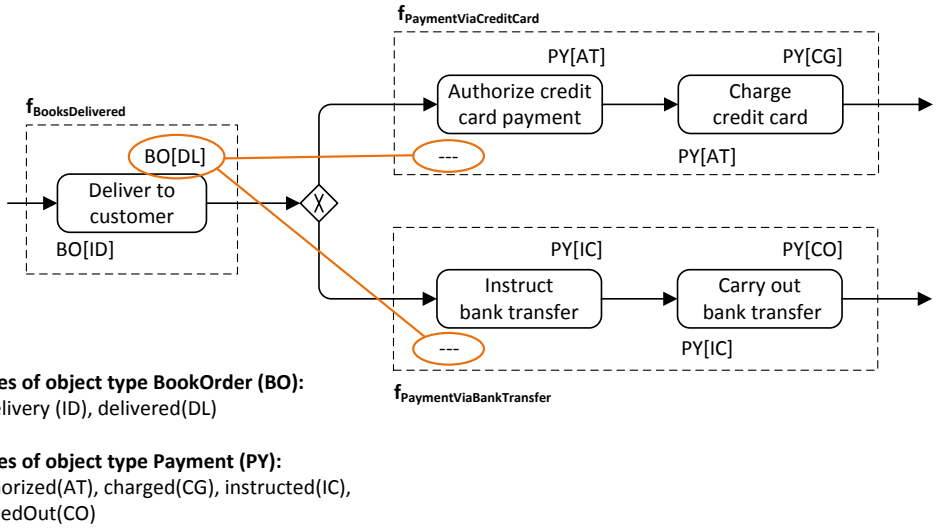


Figure 4.16: Business Process Model with Context Information

As depicted, the specification of the object flow might be incomplete, in the sense that an action has no explicit precondition, but the composition defines an object type that is processed as input. In order to consolidate the specification of the object flow, the incoming object type and state are adjusted according to the outgoing object flow of the preceding action. In the given example the incoming object for action *Authorize credit card payment* is set to the outgoing object of action *Deliver to customer*. The resulting specification is depicted in Figure 4.17.

The action specification does not only provide information about pre- and

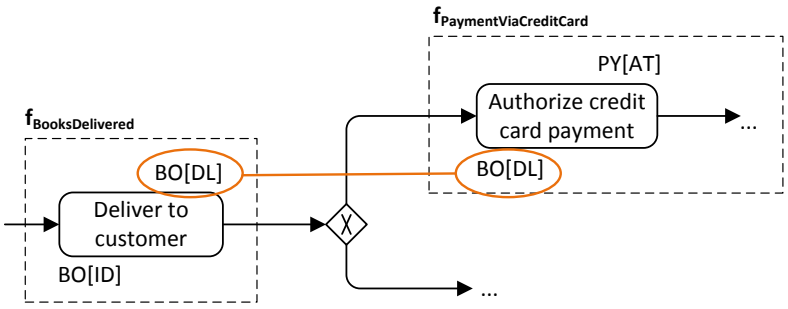


Figure 4.17: Updated Context Information in Business Process Model

postconditions, but also about actor roles which can be used to define pools within the business process model. Each action is assigned to exactly one actor who performs it. To consider relevant roles, pools are created and the actions are assigned to these pools. Figure 4.18 shows the specification of actors defined in the BPMN modeling language. The applied context information can also be mapped to elements of other modeling languages, e.g. UML activity diagrams.

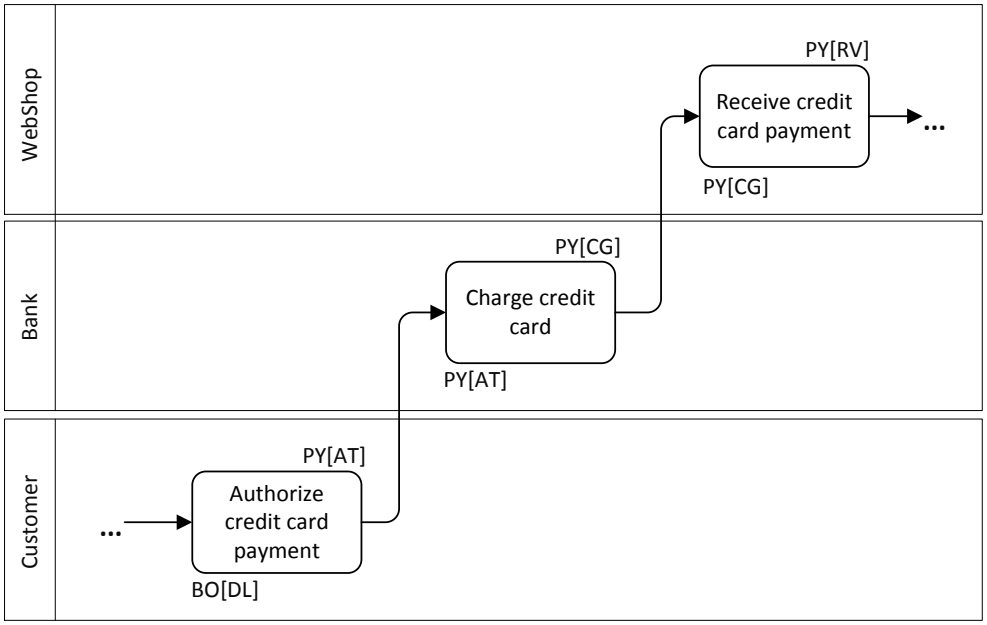


Figure 4.18: Business Process Model enriched with Actor Information

## 4.6 Summary and Discussion

In this chapter, we presented an approach for the systematic derivation of business process models from business goal specifications. We have established requirements for the goal-oriented business process derivation. Addressing these requirements, we introduced an approach to the traceable engineering of business process models that achieve the stated business goals and also consider constraints inferred from goal dependencies. Furthermore, the fulfillment of constraints with respect to valid states and transitions is ensured constructively.

We described the fragment-based calculation of coherent business processes that take into account all relevant constraints. In addition, we described how manual changes of these models can be validated by using model-checking techniques. Finally, we have shown how the composed business process is enriched with context information representing the object flow within the workflow.

In the next chapter, we describe a quality assurance approach to validate the consistency of the different models in the business goal specification and the related business process model.

# Chapter 5

# Quality Analysis and Assurance

Requirements models, like business goal models and business process models, specify strategic and operational aspects which have a high impact on further development phases. For example, the modeled business processes are used as a foundation for the composition of technical services in a service-oriented architecture (SOA) [WM06]. To enable the suitable consideration of the actual requirements, their precise and unambiguous specification is a crucial prerequisite for subsequent modeling and implementation steps. To improve the quality of these specifications, this chapter introduces our approach for quality analysis and assurance.

In Section 5.1, we discuss possible threats that impact the specification quality and infer requirements for a suitable approach. Based on these requirements we evaluate existing approaches and discuss the results. In Section 5.2, we introduce our approach to the consistency analysis of business goal models focusing on linguistic aspects of the specification. Addressing the consistency between business goal models and business process models, we present a traceability analysis approach in Section 5.3. Finally, we conclude this chapter in Section 5.4. This chapter is partially based on the earlier publication [PNEM14].

## 5.1 Requirements and Related Work

This section investigates the suitability of existing approaches and point outs their shortcomings by raising requirements for our quality analysis and assurance approach. For this purpose, we analyze possible threats for the validity and quality of a goal-oriented business process specification. We infer core challenges which are further refined to concrete requirements. Related work is evaluated against these requirements and their suitability is discussed.

### 5.1.1 Problem Analysis

The specification process for business goals involves stakeholders with different roles and backgrounds. Various levels of know-how and different perspectives lead to different understandings and viewpoints, resulting in inconsistent goal descriptions. Furthermore, the usage of natural language for goal specification can increase the likelihood of inconsistencies in terms of inaccuracies or ambiguities. Figure 5.1 sketches an exemplary overview about stakeholders who are involved in the specification process and the role of natural language. Since such specifications are required as a common communication base, inconsistencies in this specification can lead to misunderstandings or ambiguities which directly affect the stakeholders involved and their work related to these goals.

In order to ensure the consistency of business goal models specified in natural language, we have to consider two different aspects. First, business goal model elements are connected by hierarchical decomposition relations which represent different levels of abstraction. Consistency with respect to these relations implies that a set of goals is an appropriate decomposition of its super goal. Missing guidelines and the high degree of flexibility for the specification of goal models increase the threat of gaps in terms of inconsistent terminology between goals in decomposition relations. Second, the natural language itself has to be considered as it introduces inconsistencies, such as overloaded or homonymous concepts in goal descriptions. In consequence, ambiguous concepts arise and complicate the communication actions between business stakeholders and requirements engineers [Kam05].

Figure 5.2 depicts a slightly adapted excerpt from the *Fulfill book order* exam-

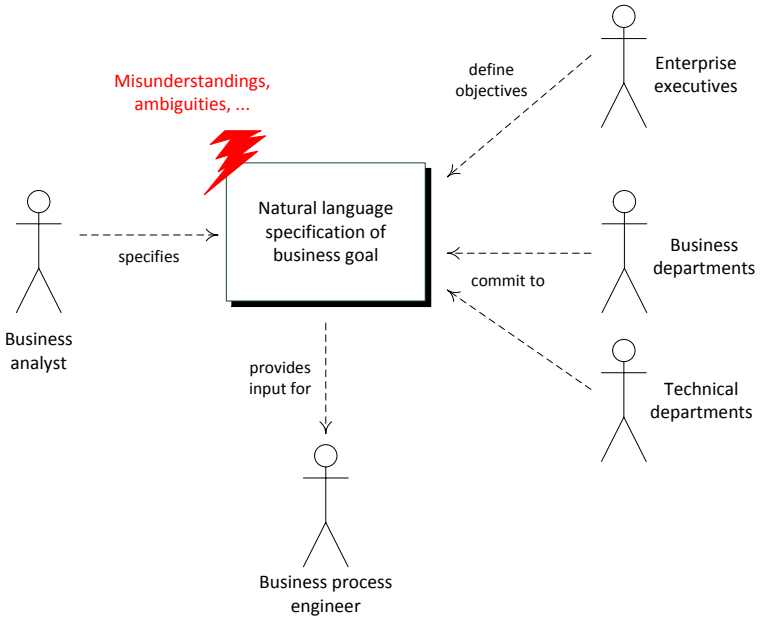


Figure 5.1: Problems Related to the Natural Language Specification of Business Goals

ple and illustrates possible inconsistencies which are grounded in the use of natural language.

The first inconsistency is shown in the goal *Quote given*. In this case, the word *quote* might relate to a sales quote or a cost estimation. As the meaning is not obvious from the given context it may lead to misunderstandings among different stakeholders. Hence, the word *quote* is a good example for a homonym that impacts the specification quality of the business goal model.

A second inconsistency can be observed with the business goals *Payment via credit card* and *Payment via bank transfer*. Both goals specify that a payment needs to be executed, but lack information about the object that has to be paid. This information, i.e. the book order, is only given implicitly by the context. A more explicit labeling naming the considered object would improve the understandability.

As a third inconsistency, we observe that the goal *Payment received* and the top goal *Fulfill book order* deal with different objects. While the latter deals with a payment that needs to be received, the top goal is concerned with the fulfillment of a book order. Possibly, using the label *Book order payment received*

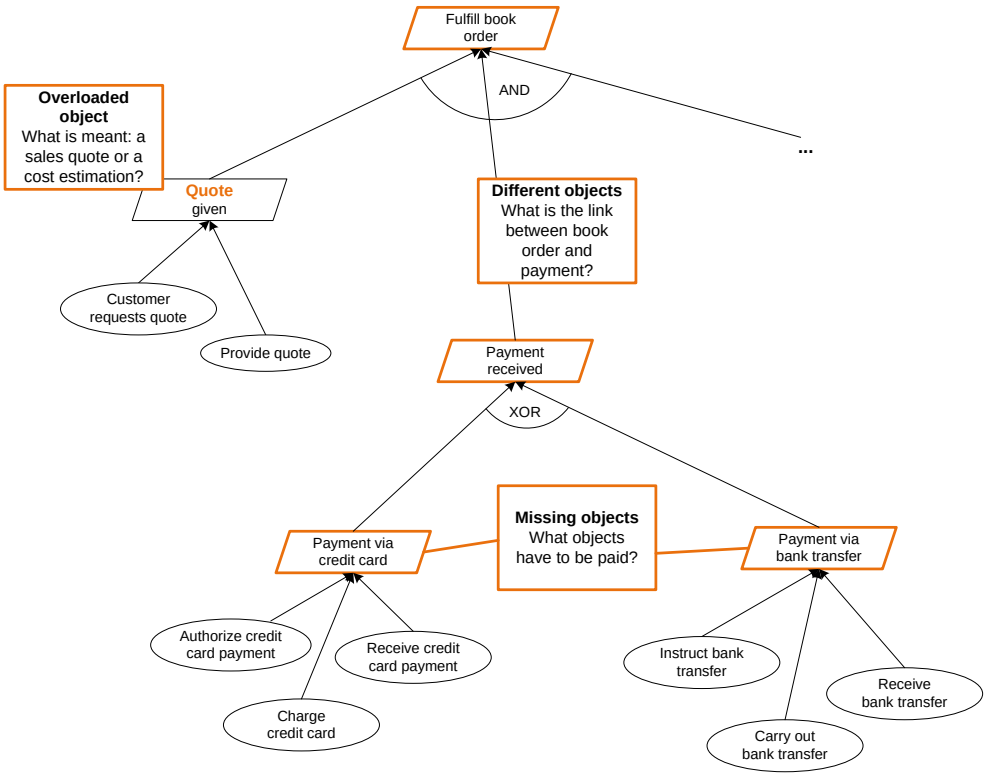


Figure 5.2: Exemplary Inconsistencies in a Natural Language Business Goal Specification

would be more concise.

The given examples of three inconsistencies caused by the imprecise usage of natural language illustrate possible negative impacts on the specification quality. While the consistency of formalized definitions has been addressed by the constructive approach in Chapter 3 (e.g. validation of action composability), the natural language specification needs to be handled in a separate manner by applying analysis and validation techniques. Therefore, we state the first challenge for quality analysis and assurance:

**Ensuring consistency of the natural language business goal model specification (C1)**

The approach for business goal modeling and business process model deriva-

tion introduced in the previous chapters provides an integrated method for the initial specification of these models. Addressing changing strategic objectives, customer requirements or market conditions, these models evolve over time. For example, new business goals can arise which need to be specified or business process models can evolve according to organizational changes in the enterprise.

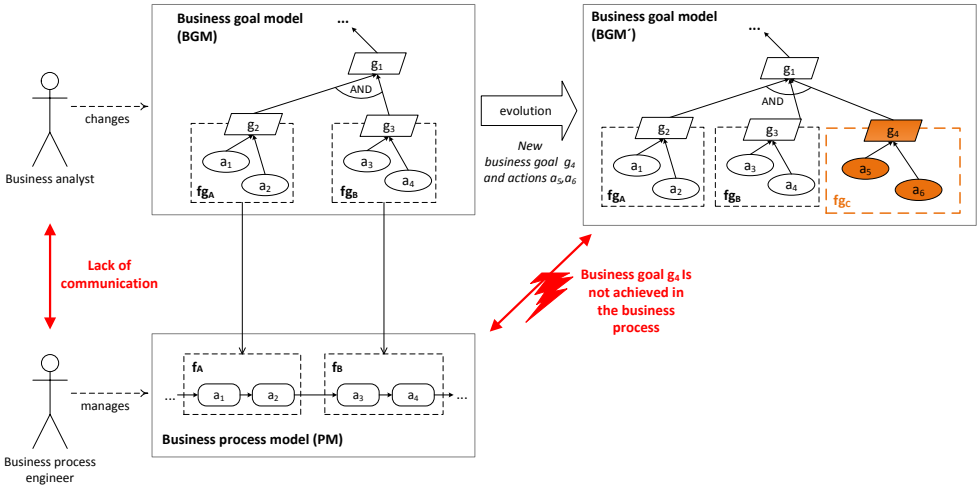


Figure 5.3: Exemplary Inconsistency caused by the Evolution of a Business Goal Model

Figure 5.3 depicts the example of an initial specification. Two business goals $g_2$ and $g_3$ are sufficiently achieved by the derived business process. We assume that a new business goal ($g_4$) is defined and operationalized by the actions $a_5$ and $a_6$. The required changes in the business goal model are made by a business analyst. The related business process model is managed by a business process engineer. If the communication among both actors involved does not work well, changes in the business goal model are not reflected in the business process model and vice versa. In this example, the lack of consideration leads to an inconsistency among both models, since goal $g_4$ is not achieved by the described business process. To overcome such inconsistencies caused by

model evolution, we state the second challenge:

**Ensuring consistency of evolving requirements models (C2)**

The problem analysis results in two challenges for quality analysis and assurance. In the next section, we derive concrete requirements from these challenges which need to be fulfilled by a suitable approach.

### 5.1.2 Requirements Definition

The challenges which have been analyzed and defined in the previous section are used to infer quality analysis and assurance requirements (QAR) which are used for the evaluation of existing approaches.

**Ensuring consistency of the natural language business goal model specification (C1)**

The specification of business goals in natural language provides a common foundation for the communication, discussion and justification of goals between the stakeholders involved. In order to ensure consistency of this specification, we infer the following requirements for our approach:

- **QAR-01** Applicability for KAOS goal models

  Several approaches to the linguistic analysis exist which can be applied to different kinds of models. The prerequisite for our scenario is the applicability for KAOS goal models. This means that the analysis considers basic KAOS concepts, like decomposition hierarchies.

- **QAR-02** Analysis of natural language specification

  We motivated the need for the validation of aspects in goal models which are described in natural language, since they provide an important foundation for the communication. Therefore, we state the ability to analyze the consistency of natural language specifications as a requirement.

- **QAR-03** Validation of consistency among goals.

  Based on the requirements QAR-01 and QAR-02, KAOS models and especially the natural language aspects need to be validated with respect to their consistency. Hence, the approach needs to be able to detect ambiguities of goals and to validate the consistent usage of logical operators in decompositions. For this purpose, the validation of consistency among goal descriptions needs to be supported.

**Ensuring consistency of evolving requirements models (C2)**

- **QAR-04** Completeness of goal achievement

  During the evolution of business goal models, new goals can arise which need to be achieved. In order to ensure their appropriate consideration in the related business process model, the completeness of the goal achievement needs to be validated by our approach.

- **QAR-05** Relevance of business processes

  Business process models evolve, e.g. by adding new actions that are performed in the actual business processes. Furthermore, business goals can become obsolete and be removed from the business goal models. As a consequence, actions in the business process model may become obsolete as well. In order to ensure cost- and time-efficient business processes it needs to be validated whether all actions are relevant. This means that they contribute to the achievement of a business goal.

- **QAR-06** Assisted correction of inconsistencies

  The evolution of models can cause inconsistencies which need to be corrected in order to ensure a consistent specification. Hence, our approach needs to support the assisted correction of detected inconsistencies.

In the following, we compare the inferred requirements against related work, describe their fulfillment and point out why these approaches are not suffi-

cient.

### 5.1.3 Evaluation of Existing Approaches

Addressing the requirements discussed in the previous section, we analyze related work. Existing approaches are discussed and weaknesses are pointed out.

Fuxman et al. [FPMT01] propose an approach for consistency analysis of i*/Tropos goal models. It provides an extended notation that is used to derive a set of formalized LTL constraints. Using these constraints, different consistency checks can be performed by applying model checking techniques. This approach focuses on verification properties instead of ensuring consistency of the natural language definitions in goal specifications.

An approach for conflict management in goal models is presented in [vLDL98, vLL00]. This work focuses on divergence detection and resolution as well as obstacle handling. These approaches are applicable to KAOS goal models and use formal goal specifications, i.e. state definitions, to validate their consistency.

Another approach to the validation of formal goal specifications is described in [BMMZ06, BGM09]. Hereby, consistency with respect to a set of given privacy and security constraints is validated. By applying a planning approach, it derives suitable design alternatives.

Traceability analysis between goal models defined in Tropos/i* and business process models is addressed in [KVB+06b, KGB06, GK07]. Traceability links are established between goals and business process elements in a manual manner. In this approach, actions in the business process models are annotated in order to illustrate their contribution to the achievement of goals. The results of the traceability analysis are categorized as either *normal*, *exceptional* or *unsatisfied*. A decision support giving assistance for the handling of unsatisfied goal is not provided.

*ProcessSEER* provides a concept and tool implementation for the semantic effect annotation of business process models [HGK09]. The explicit representation of action effects enables the analysts to identify changes in the business

| Approach | QAR-01 | QAR-02 | QAR-03 | QAR-04 | QAR-05 | QAR-06 |
|---|---|---|---|---|---|---|
| Fuxman et.al. [FPMT01] | o | - | - | - | - | - |
| van Lamsweerde et.al. [vLDL98, vLL00] | + | - | o | - | - | - |
| Bryl et.al. [BMMZ06, BGM09] | o | - | o | - | - | - |
| Kolliadis et.al. [KVB+06b, KGB06, GK07] | o | - | - | + | + | - |
| Process SEER [HGK09] | - | - | - | o | o | - |

Table 5.1: Evaluation Results for Quality Analysis and Assurance

process model and to assess their impact. These changes cannot be related to the achievement of a business goal. Further, the approach does not address the removal of inconsistencies caused by the changes.

The evaluation results are summarized in Table 5.1. As illustrated in the result table, none of the existing approaches sufficiently fulfills all stated requirements.

### 5.1.4 Discussion of Evaluation Results

The evaluation results depicted in Table 5.1 show that none of the existing approaches fulfills the stated requirements. Most approaches support goal models even though only one of them focuses on the applied KAOS notation (QAR-01). A main drawback of existing work consistency validation is the lack of consideration of natural language specifications (QAR-02). Existing approaches focus on the verification of constraints based on formal goal definitions. These approaches cannot be adopted for the validation of goal descriptions specified in natural language. The approaches analyzed only partially support the validation of consistency among different goals within the goal model (QAR-03).

Traceability analysis for business goal models and related business process

models is not sufficiently supported either. The evaluations of completeness (QAR-04) and relevance (QAR-05) are addressed well by [KVB+06b, KGB06, GK07]. The handling of inconsistencies detected in the traceability analysis is not supported by existing approaches.

## 5.2 Linguistic Consistency Analysis

In this section, we describe our approach for linguistic consistency analysis. Hereby, we focus on the natural language elements of the business goal specification and aim for the identification and removal of inconsistencies. Providing the foundation for our approach, Section 5.2.1 defines the preliminaries. In Sections 5.2.2 and 5.2.3, we introduce syntactic and semantic consistency checks for goal models.

### 5.2.1 Preliminaries

Following the definition given in Section 3.3.3, we can consider the basic structure of a business goal model as a set of goals $G$, a set of actions $A$, decomposition relations $D \subseteq (G \times G)$ and operationalization links $L \subseteq (G \times A)$.

Based on the work of Antón [Ant96], goals express a state that needs to be achieved. In the goal label, this is expressed by a verb that describes the desired state and the corresponding object which needs to be in the defined state.

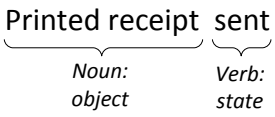


Figure 5.4: Components of a Goal Label

Figure 5.4 illustrates an example of a goal label structure. Considering the business goal *Printed receipt sent* from our running example, we can identify the verb *sent* in the past participle as well as the noun *printed receipt* which describes the object.

The structure of an action label can be decomposed in a similar manner. We

consider the action *Print receipt* as an example. The verb *print* describes the action that is performed and the noun *receipt* defines the object on which this action is executed.

Based on these insights, we consider a goal as being labeled with a verb $g^v$ that describes a distinct state and a noun $g^n$ that describes a distinct object. Accordingly, the label of an action contains a verb $a^v$ and a noun $a^n$. To extract these components, available techniques can be applied, such as [LSM12], in order to automatically annotate goals according to these components.

Since the techniques in this thesis focus on goal model decompositions and their components, we extend the definition of decompositions in business goal models. We define the following conventions to formulate the linguistic consistency conditions:

**Definition 9** *Given a goal model $\mathcal{G}$ and a goal $g' \in G$ that is decomposed into several goals or actions. Then, we define:*

- *The set of all subordinate goals of $g'$: $SG^{g'} = \{g \in G \mid (g', g) \in D\}$.*
- *The set of all subordinate actions of $g'$: $SO^{g'} = \{a \in A \mid (g', a) \in L\}$.*
- *The set of all elements that are part of a decomposition of $g'$: $DC^{g'} = SG^{g'} \bigcup SO^{g'}$.*

Applying the provided preliminaries and definitions, the following two sections define conditions for the linguistic consistency of goal models.

### 5.2.2 Syntactic Consistency Checking

In this section, we introduce conditions for the validation of goal models from the syntactical dimension of natural language. This comprises two validation checks. First, we check if a goal or action label contains all necessary components. Second, we validate whether a goal decomposition is consistent based on the object for which the goal has been decomposed.

### Component Consistency

As outlined by Antón [Ant96] and described in the preliminaries, goal and action labels contain a verb (state or action word) and a noun (object). Accordingly, we can check the decomposition of a goal with respect to the completeness of the components, i.e. whether all goals actually specify an object state and whether all actions define an object action.

The rationale behind this validation is that a goal which does not contain all components is underspecified and may cause confusion in the communication of the goals in the software development process. As an example, we consider the action *Deliver to courier*. Obviously, the action properly comprises an action word but misses the object that has to be delivered. To check goal models for this inconsistency, we define component consistency in a goal decomposition as follows:

**Definition 10 (Component Consistency)** *Let $DC^{g'}$ be a goal decomposition of a goal $g' \in G$. $DC^{g'}$ is consistent with respect to the components, iff*

- $\forall g \in DC^{g'}$ : $g^v$ *and* $g^n$ *exist*
- $\forall a \in DC^{g'}$ : $a^v$ *and* $a^n$ *exist*

### Component Stringency

The essential characteristic of goal-oriented requirements engineering is the step-wise decomposition of goals and their operationalization to concrete actions [vL03, AKRU09a, Ant96]. The decomposition process involves the creation of logical subgroups of goals until the goal achievement can be described on the level of atomic actions.

To ensure consistency among goals on the different levels of abstraction defined by the decomposition relations, it needs to be validated whether the subgoals are related to their super goal. Considering our example, the top goal *Fulfill book order* is decomposed to five subgoals. The decomposition comprises the subgoal *Payment received* which states that the payment needs to be received in order to achieve the top goal. A comparison of the components of both goal labels shows that they are dealing with different objects. The goal *Payment received* lacks information about the actual object to be paid.

This might lead to misunderstandings in the communication about this goal. Hence, we infer that the decomposition is not stringent with respect to the goal label components. To enable the validation of component stringency, we define the condition as follows:

**Definition 11 (Component Stringency)** *Let $DC^{g'}$ be a goal decomposition of a goal $g' \in G$. $DC^{g'}$ is stringent with respect to the components, iff*

- $\forall g \in DC^{g'} : g'^n = g^n$
- $\forall a \in DC^{g'} : a'^n = a^n$

### 5.2.3 Semantic Consistency Checking

The syntactic consistency validation focuses on structural aspects of goal labels and consistent object usage in goal decompositions. To enable the validation of the semantic dimension this section introduces conditions that address the unambiguousness of goals and the consistency with respect to logical operators in their decomposition relations.

#### Homonym and Synonym Consistency

Business goal models are specified by business analysts and serve as a communication foundation for several stakeholders. For this purpose, the unambiguity of the specified business goals has to be preserved. An unambiguity in the understanding often suffers from linguistic ambiguities such as homonymy (a word has more than one meaning) or synonymy (different words have the same meaning) [DP06].

To illustrate a homonym, we consider the top goal *Fulfill book order* from the running example. As discussed, the goal label can be separated into its components. The object that is addressed by this goal is the *book order*. This could be interpreted as an order of books (as intended) or as an order for reading books. Hence, homonyms can lead to misunderstandings and ambiguities in the goal specification. To enable the detection of homonyms we formalize the following consistency condition, based on Deissenboeck [DP06]:

**Definition 12 (Homonym Consistency)** *Let $Senses_Y$ denote a function that retrieves all word senses of a given word from a dictionary $Y$. Further, let $DC^{g'}$ be a goal decomposition of a goal $g' \in G$. $DC^{g'}$ is consistent with respect to homonym usage, iff*

- $\forall g \in DC^{g'} : (|Senses_Y(g^v)| = 1 \wedge |Senses_Y(g^n)| = 1)$.
- $\forall a \in DC^{g'} : (|Senses_Y(a^v)| = 1 \wedge |Senses_Y(a^n)| = 1)$.

As an example of a synonym, we consider the goal *Books ordered* and the action *Place order to supplier* in the running example. It is obvious that the verbs *to order* and *to place* both intend to lodge the book order. In order to provide a clear and precise understanding of the specified business goals and their intentions, synonyms should be avoided. To enable the validation of labels in business goal models, we define the following formalized consistency condition (cf. [DP06]):

**Definition 13 (Synonym Consistency)** *Let $Senses_Y$ denote a function that retrieves all word senses of a given word from a dictionary $Y$. Further, let $DC^{g'}$ be a goal decomposition of a goal $g' \in G$. $DC^{g'}$ is consistent with respect to synonym usage, iff*

- $\forall g_1, g_2 \in DC^{g'} :$
  $(Senses_Y(g_1^v) \cap Senses_Y(g_2^v) = \emptyset) \wedge (Senses_Y((g_1^n) \cap Senses_Y(g_2^n) = \emptyset))$
- $\forall a_1, a_2 \in DC^{g'} :$
  $(Senses_Y(a_1^v) \cap Senses_Y(a_2^v) = \emptyset) \wedge (Senses_Y((a_1^n) \cap Senses_Y(a_2^n) = \emptyset))$

**Decomposition-Logic Consistency**

Business goal models are specified in a hierarchical structure defined by decomposition relations. The relation between a super goal and its subgoals is defined by logical operators (AND, OR, XOR). In addition, we need to consider operationalization links between leaf goals and actions. The semantics of an operationalization is equal to the semantic of an AND-decomposition. To ensure consistent modeling, the business goal definition needs to be in line

with the specified logical operators. The consistency of formal goal achievement states between super goals and subgoals is ensured by the state aggregation presented in Section 3.4. However, the consistency of goal labels with respect to the logical operators needs to be ensured as well.

To validate the decomposition-logic consistency the semantic closeness is calculated based on the components of the goal/action label. Algorithms for the calculation of this semantic measure are provided by Lesk [Les86], Wu and Palmer [WP94], Resnik [Res95]. For our approach we leverage the Lin measure as it correlates best with human judgment and thus is also capable to quantify weak and moderate relations [Lin98]. Based on the measurement of the semantic closeness we formalize conditions for the evaluation of decomposition-logic consistency.

The AND-decomposition constitutes that all subgoals need to be achieved in order to fulfill the super goal. Thus, we do not expect parts of a greater whole to be strongly connected with each other rather than moderately related. For example, the actions *Authorize credit card payment* and *Charge card number* operationalize the goal *Payment via credit card*. Although the objects *credit card number* and *credit card payment* do not perfectly match, we require a moderate connection between these since they related to the object *credit card*.

To infer a consistency condition, we state that the goals in an AND-decomposition need to have a semantic closeness measure within a specified range of thresholds $\tau_{and}^{min}$ and $\tau_{and}^{max}$. Due to the identical semantic meaning of AND-decomposition and the operationalization links we consider both relations in the same validation check. Accordingly, we formalize the following AND-consistency condition:

**Definition 14 (AND-Consistency)** *Let $sim(e_1, e_2)$ be a function that calculated the closeness of two goal model elements $e_1, e_2$ with $e_1, e_2 \in G \cup A$, $\tau_{and}^{min}$ and $\tau_{and}^{max}$ be semantic closeness thresholds for an AND-decomposition. The value of $sim(e_1, e_2)$ ranges from 0 to 1, where a value closer to 0 indicates a low semantic closeness and a value closer to 1 indicates a high semantic closeness. Further, let $DC^{g'}$ be a goal decomposition of a goal $g'$. $DC^{g'}$ is consistent with respect to the AND-logic, iff*

$$\tau_{and}^{min} \leq \sum_{e_1, e_2 \in DC^{g'}} \frac{sim(e_1^v, e_2^v) + sim(e_1^n, e_2^n)}{|DC^{g'}|} \leq \tau_{and}^{max}$$

If the measure is below the threshold $\tau_{and}^{min}$ it can be interpreted in two different ways. First, the labeling of the goals may by be imprecise and needs to be reworked in order to close the semantic gap among them. Second, the decomposition may be defined by the wrong logical operator. In case of a very weak semantic closeness among the goals in the decomposition, an OR or XOR operator might be suitable.

The semantic consistency of goals within an OR/XOR decomposition needs to be validated as well. As an example, we consider the subgoals *Payment via credit card* and *Payment via bank transfer* of the XOR-decomposed super goal *Payment received*. Since the goal *Payment via credit card* is conceptually exclusive to the *Payment via bank transfer*, the semantic closeness of these goals is expected to be small.

OR-decompositions express alternatives for the achievement but do not explicitly exclude the achievement of multiple subgoals. Although the OR-decomposition is intended to express different alternatives, we expect a semantic distance for OR-decomposed goals as well. Since the usage and distinction of OR/XOR is not restricted by the modeling approach, we provide a common consistency condition for both types of decompositions.

We require for the goals in an OR-/XOR-decomposition that the semantic closeness is smaller than a given threshold $\tau_{or}$. The formalized condition for OR-/XOR-Consistency is defined as follows:

**Definition 15 (OR-/XOR-Consistency)** *Let $sim(g_1, g_2)$ be a function that calculated the closeness of two goals $g_1, g_2 \in G$ and $\tau_{or}$ be a semantic closeness threshold for an OR-/XOR-decomposition of goals. The value of $sim(g_1, g_2)$ ranges from 0 to 1, where a value closer to 0 indicates a low semantic closeness and a value closer to 1 indicates a high semantic closeness. Further, let $DC^{g'}$ be a goal decomposition of a goal $g'$. $DC^{g'}$ is consistent with respect to the OR/XOR-logic, iff*

$$\sum_{g_1,g_2 \in DC^{g'}} \frac{sim(g_1^v, g_2^v) + sim(g_1^n, g_2^n)}{|DC^{g'}|} \leq \tau_{or}$$

To summarize, the definitions presented here provide formalized consistency conditions which can be used to validate the semantic consistency of business goal models.

## 5.3 Traceability Analysis

In this section, we address the consistency between business goal models and business process models focusing on the aspect of traceability. For this purpose, we present an approach that enables the detection of inconsistencies and the assisted correction of them. Hereby, we assure the fulfillment of two quality criteria. First, we validate completeness in the sense that all defined business goals are achieved by the execution of the modeled business process. Second, to ensure the relevance of all actions in the business process model, we validate their contribution to the business goal achievement.

Section 5.3.1 describes a preprocessing step that accumulates all state changes in the business process model in order to enable their validation in an aggregated manner. The algorithms for the actual analysis are presented in Section 5.3.2. In Section 5.3.3, we discuss how the inconsistencies can be removed in an assisted manner.

### 5.3.1 Effect Accumulation

The first step is a preprocessing of the business process model prior to the actual analysis. Hereby, the effects caused by the executions of atomic actions are accumulated through the control flow of the business process. The accumulated effect of a business process model element is described by the state of a business object that is achieved if the business process is executed to the position of this element.

Accumulated effects are used in two different ways. First, the accumulation is used to validate whether an action within a business process fragment sufficiently contributes to the achievement of the related fragment goal. Second, the accumulated effect of the whole business process model is used to validate whether the overall business goal is achieved by the execution of the modeled business process.

#### Intermediate effect of an action

The accumulation of effects is based on the effects caused by atomic actions, termed intermediate effects. The intermediate effect of an action is defined

by the state changes of a business object during the execution of the action. In the effect accumulation, we consider the postcondition of an action as its immediate effect.

To illustrate the concept of effect accumulation we consider the example depicted in Figure 5.5. The actions in the business process model are annotated with the caused changes of object states, as described in Section 4.5.

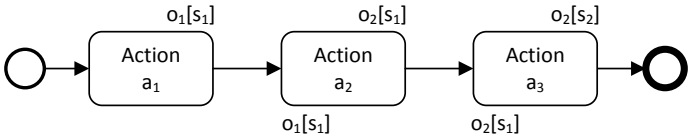


Figure 5.5: Annotated Business Process Model for Effect Accumulation

Accordingly, action $a_1$ has no precondition and changes the state of an object of type $o_1$ to $s_1$. This means that the immediate effect of this action is defined as $o_1[s_1]$. Action $a_2$ requires this state as precondition (which is fulfilled sufficiently) and changes the state of an object of type $o_2$ to $s_1$.

The immediate effects of these two actions handle different object types. This means that the effect caused by $a_1$ is not overwritten by the effect caused by $a_2$. Therefore, we consider these effects as independent from each other. The accumulated effect for $a_2$ is defined as $o_1[s_1] \wedge o_2[s_1]$.

Action $a_3$ sketches an example for a dependent effect. The state $o_2[s_1]$ defines the precondition and the immediate effect of $a_3$ is given by the postcondition $o_2[s_2]$. In order to calculate the accumulated effect of $a_3$, we need to consider the effects of the preceding actions given by the accumulated effect of action $a_2$: $o_1[s_1] \wedge o_2[s_1]$. Since the immediate effects of $a_2$ and $a_3$ change the state of objects from the same type, we consider these as dependent effects which need to be handled in the accumulation.

For a dependent effect, the preceding state of the object type is overwritten by the immediate effect of the current action. In this example, the accumulated effect of action $a_3$ is expressed as $o_1[s_1] \wedge o_2[s_2]$. Table 5.2 summarizes the immediate and accumulated effects in terms of state definition for the example.

The general guideline for the accumulation of effects for actions in a business process model is given Definition 17.

| **Process Element** | **Immediate effect** | **Accumulated effect** | **Consolidated acc. effect** |
|---|---|---|---|
| Start event | $\emptyset$ | $\emptyset$ | $\emptyset$ |
| Action $a_1$ | $o_1[s_1]$ | $o_1[s_1]$ | $o_1[s_1]$ |
| Action $a_2$ | $o_2[s_1]$ | $o_1[s_1] \wedge o_2[s_1]$ | $o_1[s_1] \wedge o_2[s_1]$ |
| Action $a_3$ | $o_2[s_2]$ | $o_1[s_1] \wedge o_2[s_1] \wedge o_2[s_2]$ | $o_1[s_1] \wedge o_2[s_2]$ |
| End event | $\emptyset$ | $o_1[s_1] \wedge o_2[s_1] \wedge o_2[s_2]$ | $o_1[s_1] \wedge o_2[s_2]$ |

Table 5.2: Distinction between Immediate and Accumulated Effects

**Definition 16 (Effect Accumulation)** *The accumulated effect at an action $v$ is its immediate effect $e_v$ plus all preceding effects $b_v^{pre}$: $ACC_v : e_v \wedge b_v^{pre}$*

**Definition 17 (Consolidated Effect Accumulation)** *The accumulated effect at an action $v$ is its immediate effect $e_v$ plus all independent preceding effects $b_v^{pre}$: $ACC_v : e_v \wedge b_v^{pre}$*

Effects are not only accumulated for actions but also for the gateways in a business process model. Fork gateways do not require an explicit effect accumulation. The accumulated effect at the fork gateway is identical to the accumulated effect of the preceding actions.

First, we consider the effect accumulation for joins of parallel gateways. The fork of a parallel gateway splits the control flow and activates all outgoing flows. This means that at the related join gateway the accumulated effects of all preceding nodes must hold. To enable the effect accumulation, we assume that no contradictions occur on the parallel paths. In this case, contradictions are conflicting actions in terms of dependent effects on the same object type. In order to ensure a conflict-free business process model, existing approaches like [GHSW09] can be applied in order to detect and remove contradicting effects before they are accumulated.

The accumulation of effects at parallel joins can be formalized as follows.

**Definition 18 (Parallel-Join Effect Accumulation)** *The accumulated effect at a parallel join $j$ with n incoming flows is $ACC_j : ACC_1 \wedge ACC_2 \ldots \wedge ACC_n$, where $ACC_i$ is the accumulated effect of the i-th preceding flow node.*

The second gateway is an inclusive-OR gateway. At the inclusive fork, one or more of the possible paths can be executed. Therefore, the accumulated effect at the inclusive join needs to consider that each possible path might have been executed. Accordingly, the effect accumulation for an inclusive join is defined as described in Definition 19.

**Definition 19 (Inclusive-Join Effect Accumulation)** *The accumulated effect at an inclusive join j with n incoming flows is $ACC_j : ACC_1 \vee ACC_2 \ldots \vee ACC_n$, where $ACC_i$ is the accumulated effect of the i-th preceding flow node.*

The exclusive gateway is more restricted than the inclusive gateway. An exclusive fork defines possible paths, but only one of them is executed. Accumulated effects for exclusive joins define that the accumulated effects of exactly one path are valid at the join gateway. Hence, the exclusive join effect accumulation is formalized in Definition 20.

**Definition 20 (Exclusive-Join Effect Accumulation)** *The accumulated effect at an exclusive join j with n incoming flows is: $ACC_j : ACC_1 \oplus ACC_2 \ldots \oplus ACC_n$, where $ACC_i$ is the accumulated effect of the i-th preceding flow node.*

Applying the definitions presented, we iterate through the business process model and calculate the accumulated effects for actions and gateway joins iteratively. In the following section, we describe the application of accumulated effects for the validation of the business goal achievement of the related business process.

### 5.3.2 Business Goal Traceability Analysis

Based on the definitions provided, we analyze the traceability between a business goal model and a business process model. The analysis can be performed for a business process fragment or for the complete business process model. In the following, we describe the analysis of fragments, but the validation of

the whole business process model is comparable, as it would consider the whole business process as a fragment and the top business goal as the goal that needs to be achieved.

The analysis aims to answer two questions in order to validate traceability. First, it is analyzed whether the fragment goal is *always* achieved by executing the related business process fragment. This means that the goal is achieved on all possible execution paths in the business process fragment. Second, in order to validate the relevance of the business process elements, it is analyzed whether there are any additional effects caused by the execution of the business process fragment. Hereby, it is assured that no side-effects occur that could have a negative impact. Such a negative impact could be that actions in a business process are not contributing to a business goal and therefore consume resources, e.g. time or money.

By following the approach described in Algorithm 4, we validate the traceability between a business process fragment $f$ and the related fragment goal $fg$. As input for the analysis the algorithm requires a list of immediate effects $IE_f$, the accumulated effect $ACC_f$ and the consolidated accumulated effect $CAC_f$. These information are gathered by the effect accumulation described in Section 5.3.1.

The list of immediate effects provides an overview about all achieved states within the business process fragment. First, we check whether the achievement state of the fragment goal is contained in the list of immediate effects to validate whether the fragment goal is achieved on any path. If this check fails, we can abort the analysis and return the status `ERROR`.

To illustrate the validation, we consider a fragment goal with the achievement state $\Diamond\ o_1.s_2$. This means, objects of type $o_1$ need to be in state $s_2$ at some point in time. To validate that this goal is achieved we analyze whether the effect $o_1[s_2]$ occurs.

To illustrate the possible scenarios, Figure 5.6 sketches alternative business process models. These models represent possible evolutions of a business process fragment that is related to the fragment goal. Since action $a_2$ has the immediate effect $o_1[s_2]$ and is part of each fragment, all three examples would pass the first check.

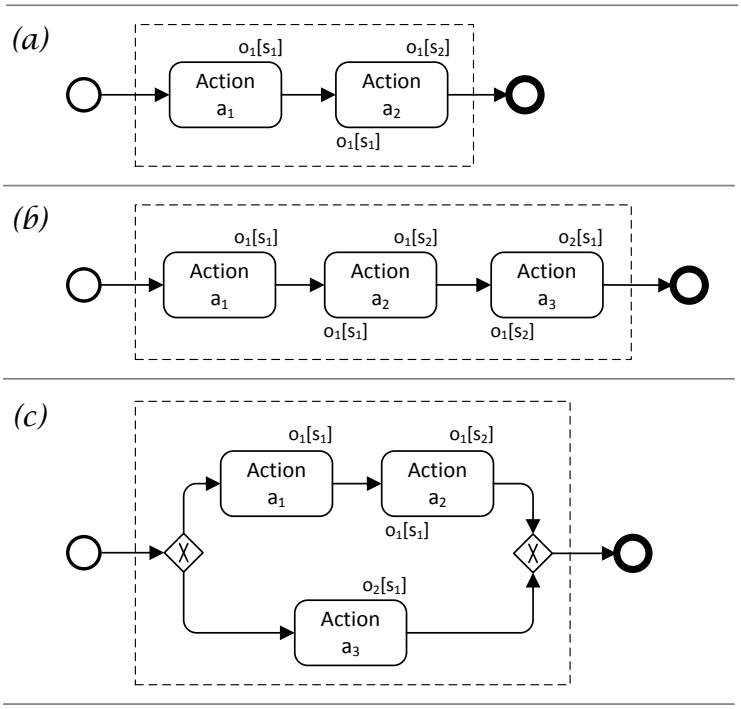


Figure 5.6: Scenarios of Evolved Business Process Models

In the second step, we validate whether the required state is achieved on each possible execution path and if additional effects occur. To illustrate, we consider scenario a) from Figure 5.6. In this simple example, the fragment contains a sequence of two actions with the immediate effects $o_1[s_1]$ and $o_1[s_2]$. This means that in each (exactly one in this example) possible combination, the achievement state $o_1[s_2]$ is reached.

Scenario b) describes a different business process fragment, which has been extended by an additional action $a_3$. Action $a_3$ has an immediate effect on object $o_2$ defined by state $s_1$. There is only one valid path that includes execution of all actions. On the one hand, the result shows that the fragment goal is achieved. On the other hand, a new effect $o_2[s_1]$ occurs that does not contribute to the related goal and might be unnecessary.

In this case, an additional check is performed against the consolidated accumulated effect which is $o_1[s_2] \wedge o_2[s_1]$ for this example. It reveals that the effect $o_1[s_1]$ is not an additional effect since it is overwritten by $o_1[s_2]$ and therefore contributes to the achievement of the desired state. In contrast, the effect $o_2[s_1]$

is also identified in the consolidated accumulated effect and is marked as an additional effect. In order to trigger the manual decision whether action $a_3$ is required, the algorithm returns a warning.

The accumulated effect is also used to validate whether the achievement state is overwritten, i.e. it is not the last state of this object within the fragment. For this purpose, we check if the desired state is part of the consolidated accumulated effect. In case the effect cannot be identified it is inferred that it has been overwritten and a warning is returned.

Scenario c) shows the third example, with two exclusive alternative paths defined by the actions $a_1$, $a_2$ on the one path and $a_3$ on the other path. Accordingly, two possible paths can be executed, i.e. that either the states $o_1[s_1]$ and $o_1[s_2]$ are reached or state $o_2[s_1]$ is reached. In the second case, we identified an execution path on which the fragment goal is not achieved. Hence, the algorithm returns an error message.

**Algorithm 4** Validation of Traceability

**Precondition:** Business process fragment $f$, related fragment goal $fg$, list of immediate effects $IE_f$, the accumulated effect $ACC_f$ and the consolidated accumulated effect $CAC_f$.
**Postcondition:** Traceability status $t$

```
t ← "Ok"
s ← fg.getAchievementState()
if s ∈ IE_f then
    for all ExecutionPath e in f do
        if ¬ e.achieves(fg) then
            t ← "Error: Fragment goal is not achieved on all paths."
            break;
    if CAC_f.remove(s) ≠ ∅ then
        t ← "Warning: Additional effects detected."
    else if CAC_f.remove(s) = CAC_f then
        t ← "Warning: Achievement state overwritten."
else
    t ← "Error: Fragment goal is not achieved."
return t
```

A precise description of the traceability analysis is given in Algorithm 4. Possible results of the algorithm are summarized in Table 5.3. Status *OK* indicates that no changes are required. *Warnings* describe that the fragment goal

is achieved but side effects might trigger the need for a redefinition of the process model or the specific fragment. An error marked in *red* explicitly requires a correction since the fragment goal is not achieved sufficiently. In the next section, we describe a set of strategies to support the correction in an assisted manner.

| Result | Description |
|---|---|
| OK | Fragment goal achieved. |
| WARNING | Additional effects detected. |
| WARNING | Achievement state overwritten. |
| ERROR | Fragment goal is not achieved on all execution paths. |
| ERROR | Fragment goal is not achieved on any execution path. |

Table 5.3: Results of Traceability Analysis

### 5.3.3 Assisted Traceability Correction

The traceability analysis can detect several inconsistencies as sketched in Table 5.3. To remove such inconsistencies, the models need to be adapted to restore a consistent specification. This can be done by either changing the business goal model, the business process model or the fragment definition.

Usually, it cannot be decided in an automated manner which changes are required. For example, if a fragment goal is not achieved the goal might be obsolete or the business process model could be incomplete. As a basic guideline, the requirements defined in goal models are considered as more stable than business process models [AMP94, vL01]. More unstable information means that the corresponding model is more likely to evolve. However, it cannot be defined in a clear-cut way whether the business goal model or the business process model needs to be changed. To support the correction of the models, we propose a set of guidelines which are applied in a manual manner.

**Additional effects detected.** This status is given as a warning. Since all business goals are achieved, it might not be mandatory to adapt the models. Nonetheless, we discuss strategies to solve this warning. First, it needs to be decided whether the detected additional effect is desired or not.

If the additional effect is desired, it means that the action or the set of actions that cause this effect are required in the business process model. Either the action can be related to an existing business goal or a new business goal is created which is achieved by the corresponding action. The creation of a new business goal requires an update of the whole specification, including definition of actions, generation of object life cycle models and an update of the business object type model. Referring to the new business goal, a fragment is created encapsulating the action or set of actions causing the additional effect. Finally, the assignment to the current fragment is deleted.

In the second scenario, the additional effect is not desired. In this case, the action is deleted from the business process model. If the action is the only one in the branch, this is deleted as well.

**Achievement state overwritten.** This status is returned as a warning. It means that the fragment goal is achieved but the state is overwritten by the immediate effect of a subsequent action. If this effect is undesired, the warning can be resolved in a similar way to the warning *Additional effects detected*. One way to handle this scenario is the deletion of the action or set of actions that overwrite the achievement state. In contrast to an additional effect which is defined by a state of a different object type, the overwriting expresses a new state of the same object type. This can be an indication that it is not a desired effect but the further processing of the object. Therefore, it might be more expedient to validate whether the activities should be re-assigned to a different fragment or if the achievement state of the fragment goal needs to be updated to this state.

**Fragment goal is not achieved on all execution paths.** This status describes that the fragment goal is achieved on at least one execution path but not on all possible paths within the business process fragment. Figure 5.7 sketches possible resolutions for this scenario. We consider an exclusive-OR gateway and also assume that the achievement state $o_1[s_2]$ needs to be achieved. As illustrated, the execution of the second path including action $a_3$ does not fulfill the stated goal.

The first resolution strategy assumes that $a_3$ is required to describe the actual business processes appropriately in the model. In order to restore consistency,

the fragment definition is adapted and a new fragment is created. In this example, we create fragment $f_B$ and assign action $a_3$ to it. In addition, the new fragment needs to be assigned to an existing business goal or a new goal needs to be created that is assigned as a fragment goal.

In the second scenario, we assume that $a_3$ is an obsolete action which is not required in the business process model. In this case, the whole path is deleted (including the gateway) and the resulting fragment ensures that the fragment goal is achieved on each execution path.

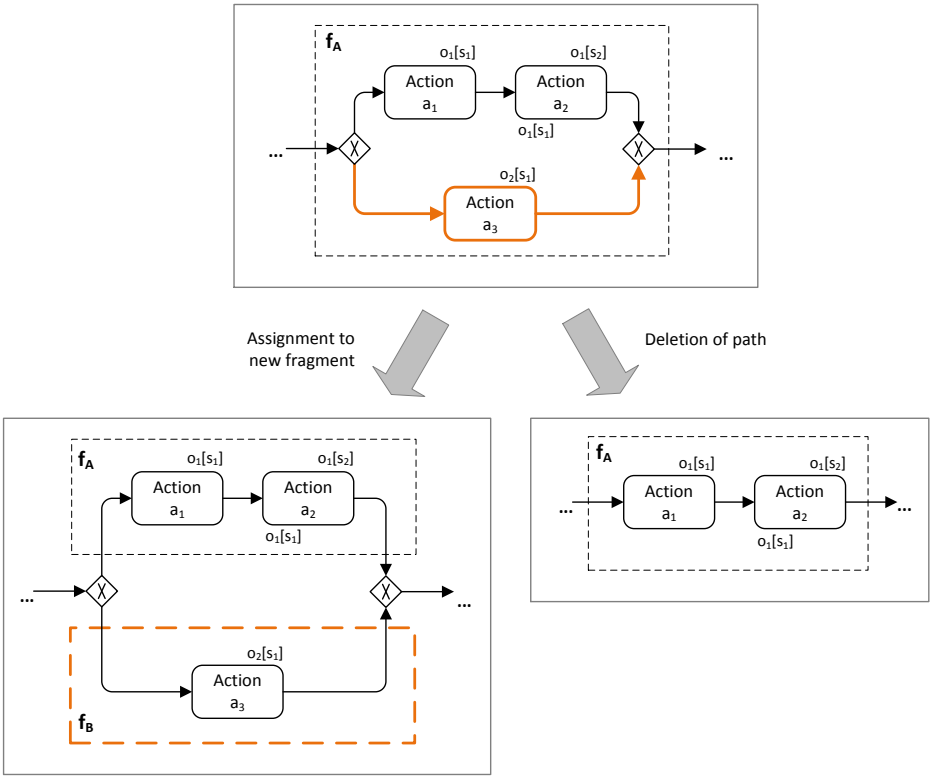


Figure 5.7: Resolution of Paths which do not achieve the Fragment Goal

A further solution is the redefinition of the XOR gateway as an AND gateway. This ensures that the desired state is achieved but would cause an additional effect warning, since action $a_3$ is executed as well.

**Fragment goal is not achieved on any execution path.** In the last scenario, the fragment is not achieved on any execution path in the business process fragment. This error needs an intensive analysis by business analysts and

business process engineers since it shows a complete loss of traceability for the business process fragment. It might be an indication that the business goal has become obsolete and the fragment actually contributes to a different goal or a goal that has not been defined yet. Otherwise, the fact that the fragment is not related to a business goal can also indicate that the fragment has become obsolete and should be deleted.

## 5.4 Summary and Discussion

In this chapter, we presented a quality analysis and assurance approach for business goal and business process models. Our solution addresses all requirements related to the core challenges: consistency of the natural language business goal model specification (C1) and consistency of evolving requirements models (C2).

First, we introduced a linguistic consistency analysis comprising syntactic and semantic consistency conditions. Consolidating the natural language specifications of goal models, our approach improves the overall quality of the requirement specification and reduces the risk of misunderstandings or ambiguities.

Second, we presented a traceability analysis approach for preserving consistency among evolving business goal and business process models. In addition to algorithms for an automated analysis, we contribute a set of pragmatic guidelines to remove the identified inconsistencies. Hereby, we address the fact of constantly evolving requirements resulting in changing business goals or adapted business processes.

To improve the applicability and to enable the evaluation of our approach, the prototypical tool implementation is presented in the next chapter.

# Chapter 6

# Tool Support

The concepts developed in this thesis have been prototypically implemented in the GooPE workbench. GooPE stands for **Go**al-**o**riented **P**rocess **E**ngineering. Figure 6.1 depicts an overview about the GooPE architecture.

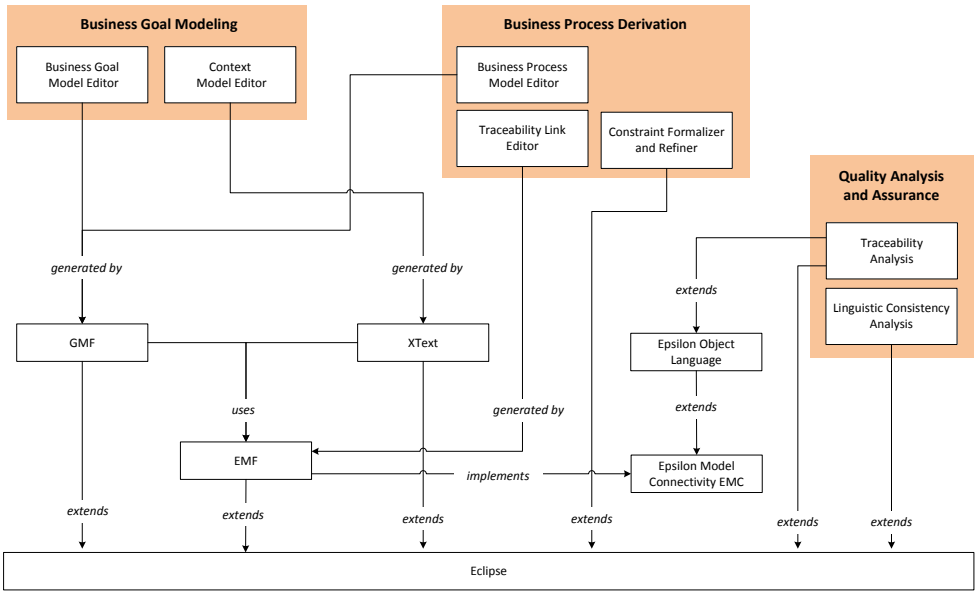


Figure 6.1: Architectural Overview about the Implemented Tool Support

The implementation comprises tool support for the three main building blocks of our approach to business goal modeling, business process derivation and

quality analysis and assurance. The Eclipse framework serves as a platform for the implementation that is described in more detail in the following sections. The individual components for business goal modeling are described in Section 6.1. In Section 6.2, the tooling for business process derivation is presented. Finally, we discuss the automated quality analysis and assurance functionalities in Section 6.3.

## 6.1 Business Goal Modeling with GooPE

### Business Goal Model Editor

To support the creation and management of business goal models, we implemented a graphical editor as depicted in Figure 6.2. Technically, the editor is based on EMF and GMF. These frameworks are used to define the language definition through an abstract syntax (EMF) and the graphical representation through a concrete syntax (GMF). Based on these definitions, a graphical editor can be generated.

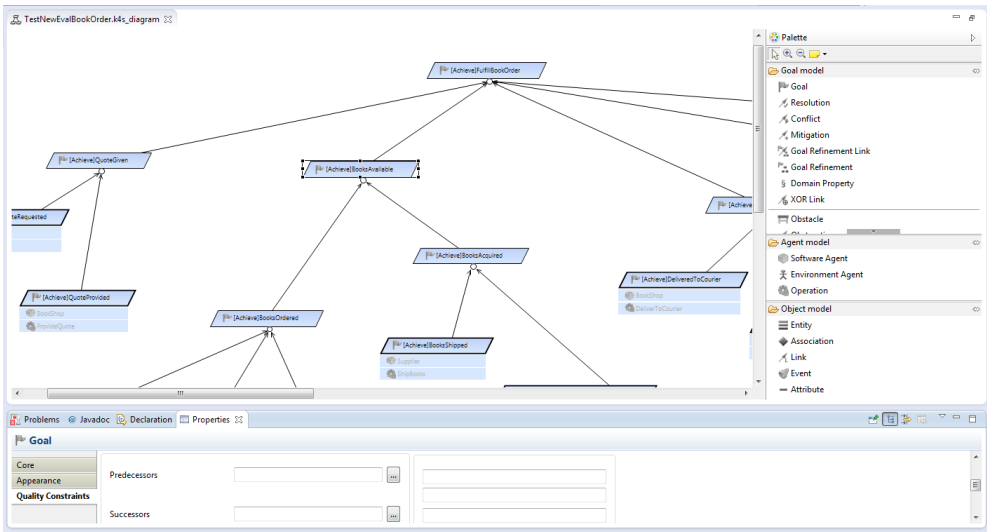


Figure 6.2: Graphical Goal Modeling Editor

The central area contains the actual goal editor that supports the work with the business goal model. Valid modeling elements, as defined by the language definition, are listed in the right area and can be added to the goal model by

drag & drop. The definition of temporal dependencies is fully integrated into the modeling process and can be performed in the bottom area. Hereby, the dependent goals can be selected from a list. This selection is also supported by validity checks to avoid the definition of inconsistent dependencies. For this purpose, we implemented the checks described in Section 3.3.4 to make sure that only valid goals are selectable.

#### Context Model Editor

To create and maintain the business object type models, we implemented a visual editor that supports the user. A screenshot of the editor is depicted in Figure 6.3. A set of notation elements is provided which can be added to the model by using drag & drop.

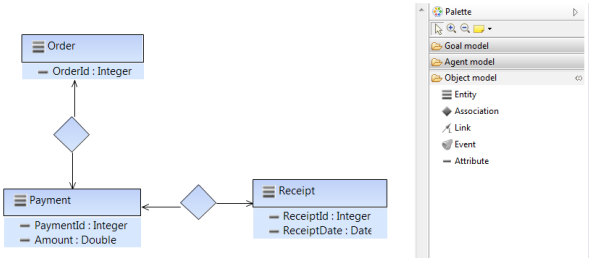


Figure 6.3: Visual Business Object Type Editor

The object life cycle models can be created, edited and viewed in a textual editor, implemented based on XText. Figure 6.4 shows an exemplary definition of valid states and state transitions for the object types *Money Order* and *Receipt*.

## 6.2 Business Process Derivation with GooPE

#### Business Process Modeler

In addition to goal models, our approach is also strongly related to business process models. For this purpose, we provide an editor for business process models that is based on the existing BPMN2 Eclipse Modeler [1]. A screenshot of the editor is depicted in Figure 6.5. The applied notation is based on the BPMN 2.0 specification [OMG11a] and supports the definition of business

[1] http://www.eclipse.org/bpmn2-modeler/

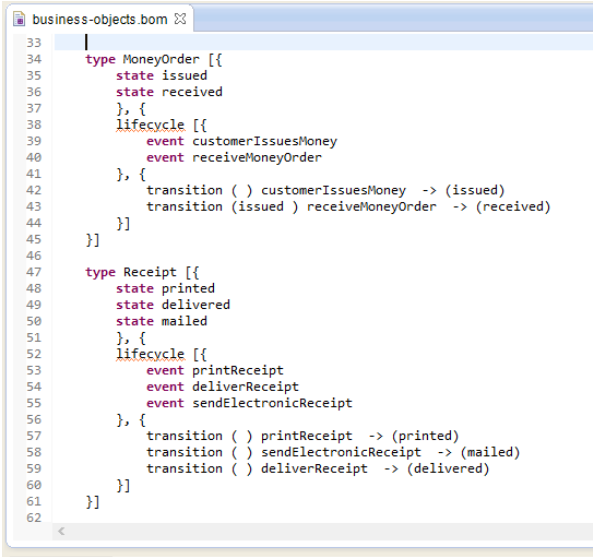


Figure 6.4: Textual Object Life Cycle Model Editor

process models.

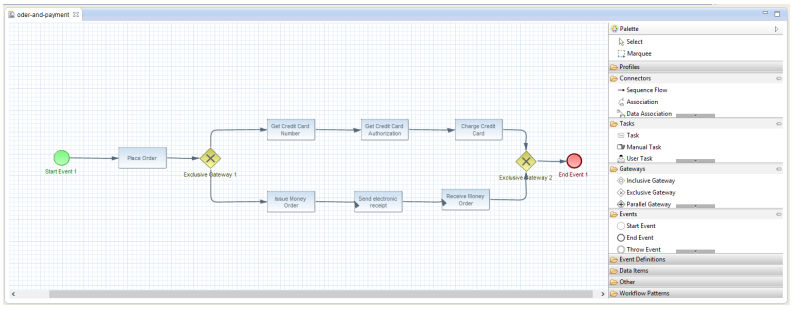


Figure 6.5: Business Process Modeling Editor

### Traceability Link Editor

Traceability links between business goals and business process elements can be managed in a separate editor. This tooling can be used to work with existing links or to create new links manually. For example, it might be useful to define that a more abstract fragment is related to a higher level business goal. The screenshot in Figure 6.6 shows the support for the manual definition of traceability links between a business goal *Receive Payment* and the related business process fragment.

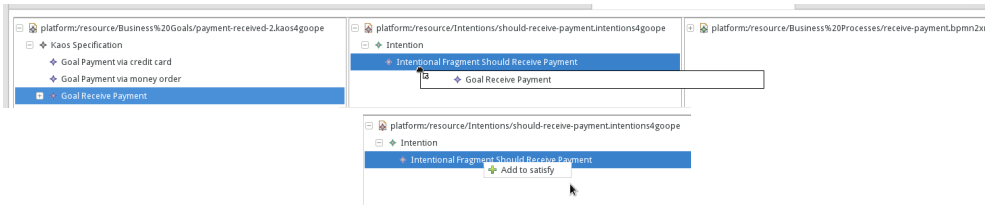


Figure 6.6: Editor for Traceability Links

### Constraint Formalizer and Refiner

The approach for the translation of goal dependencies into formalized CTL constraints has been implemented as well. This component formalizes the defined dependencies and refines the resulting constraints to the level of fragment goals. Based on a concrete business goal model, the formalization and refinement of defined temporal dependencies can be triggered. To facilitate the handling and the manual changes of the constraints, GooPE provides a textual CTL editor. A screenshot is depicted in Figure 6.7. The editor is implemented by using the XText plugin. Hereby, a formal CTL grammar (cf. Appendix A) is defined in the XText syntax and the editor is generated in an automated manner.

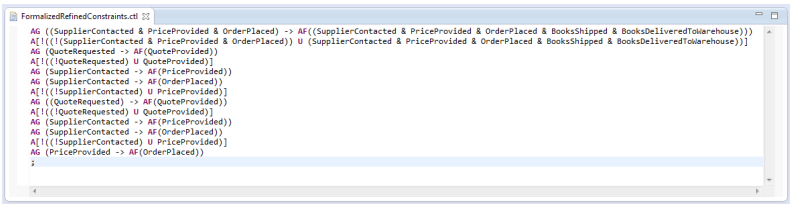


Figure 6.7: CTL Constraint Editor

The application of this editor has two advantages for the user. First, the provided capabilities like syntax highlighting improve the readability and maintainability of the CTL constraints. Second, the defined constraints are checked automatically against the defined language grammar. This means that their consistency with the CTL language is validated to improve the correctness and overall specification quality.

## 6.3 Quality Analysis and Assurance with GooPE

The quality analysis and assurance implemented here comprises components for the linguistic consistency analysis and traceability analysis. In the following, both components are described in detail.

#### Linguistic Consistency Analysis

The proposed linguistic consistency checks have been implemented in cooperation with the HU Berlin[2] and WU Wien[3]. The foundation for our tooling was an existing application which has been developed to validate the linguistic consistency of different business process models within a repository. The results of this work have been published in [PLM13].

We adopted the existing work for the application of linguistic consistency within goal models. We implemented a transformation functionality that enables the translation of a given goal model into a processable input format that captures all the information from the graphical representation. In order to analyze different word senses for the semantic consistency conditions, our implementation relies on the lexical database WordNet version 3.0 [MF98, Mil95].

#### Traceability Analysis

The implemented traceability analysis is based on the languages and tools developed in the epsilon project [KPP06, Kol07, KPP08, KRGDP14], which enable the definition and validation of constraints. Moreover, the epsilon object language is used to implement the repair strategies. In our implementation, we leverage the general purpose Epsilon Object Language (EOL) and the Epsilon Validation Language (EVL) as a task specific language.

EVL is a superset of EOL which supports the application of all manipulation operations provided by the epsilon object language. A further advantage of EVL, e.g. in contrast to OCL, is the ability to differentiate between constraints and critiques. A constraint is a critical error that needs to be fixed while a

[2]https://www.hu-berlin.de/
[3]http://www.wu.ac.at/

critique describes a problem that might not be critical if it is not fixed.

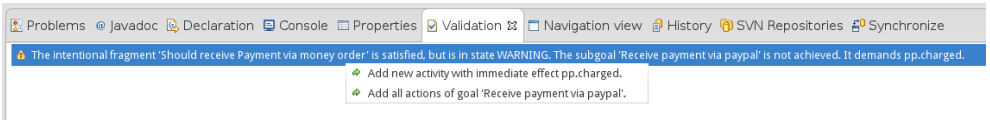


Figure 6.8: Results and Proposed Repair Strategy for Traceability Analysis

The application of EVL and EOL and the tooling provided for the Eclipse environment enables the integration of a quick-fix decision support. Figure 6.8 shows the message that is provided to the user and the proposed repair strategy to fix the identified error. As input, the analysis relies on the defined traceability links between fragment goals and business process fragments.

## 6.4 Conclusion

In this chapter, we gave an overview about the tool support developed to implement the concepts presented in this thesis. Although the current implementation is in a prototypical stage it provides a suitable foundation for the evaluation that is performed and described in the following chapter.

# Chapter 7

# Evaluation

In the previous chapters, we presented our approach comprising a modeling language for business goal models, a method for business process derivation and quality analysis and assurance techniques. In this chapter, we evaluate our approach with respect to applicability, expressiveness and integrateability with other modeling techniques.

In Section 7.1, we describe our experiences with the applicability and integrateability of our approach during an industrial cooperation project. In Section 7.2, we discuss a case study addressing the applicability and expressiveness of the introduced business goal modeling language. A second case study is presented in Section 7.3 that comprises three different goal models. We apply the linguistic consistency validation to these models and demonstrate the contribution of our quality analysis approach. The experimental application of the traceability analysis is described in Section 7.4. Finally, threats to validity are discussed in Section 7.5 and Section 7.6 concludes this chapter.

## 7.1 Project Report: Applicability & Integrateability

The first evaluation is discussed as a report of the research project named *Entwicklung eines Werkzeugs zur automatisierten Überführung von Anwendungssoftware-Komponenten in eine Cloud-Computing-Umgebung* (AACC). The AACC project is funded by the German *Bundesministerium für Wirtschaft und Technologie*

as part of the funding program *ZIM - Zentrales Innovationsprogramm Mittelstand*. The project is a cooperation between the s-lab - Software Quality Lab of the University of Paderborn and the S&N AG. Details about the project experiences and the lessons learned can be found in our publication [NSB+15].

### 7.1.1 Evaluation Setup

#### Evaluation Context

This project deals with the model-based migration of components to a cloud-enabled Software-as-a-Service (SaaS). The main objective of this project is the development of a method for cloud migration. This comprises the definition of requirements and their refinement to executable transformation rules. Considering the context of this thesis, the report focuses on the early phases of the requirements engineering performed during the project.

Following the model-driven approach, we aim for the engineering of requirements and their refinement to formalized, executable model transformation rules. Figure 7.1 shows an overview about the requirements specification in a goal-oriented manner. First, the migration goals are elicited and operationalized to migration use cases (MUCs). These MUCs provide an abstracted view on a migration step. In the second step, the MUCs are formalized to transformation rules. In the following, we discuss the application of our goal modeling approach and describe our experiences during this project.

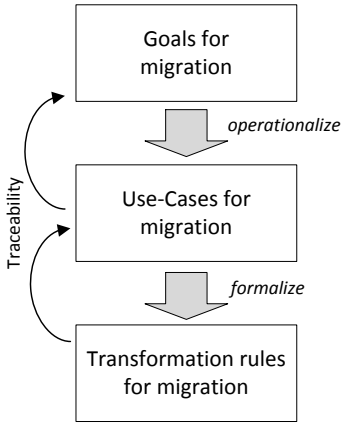


Figure 7.1: Goal Modeling in the AACC Project

**Evaluation Questions**

In this project report, we evaluate our goal modeling approach with respect to the following questions:

**EQ1** Can the goals be elicited by existing techniques?

**EQ2** Are the modeling capabilities applicable to define requirements and their dependencies in early project phases?

**EQ3** Can the presented approach be integrated with other modeling techniques?

### 7.1.2 Evaluation Execution and Results

In the AACC project, we had two main sources of requirements. First, requirements arise from the business case addressing small to medium-sized insurance companies with a mobile-enabled application implemented as a cloud service. Furthermore, the migration to a cloud-service as well as the deployment on a specific platform lead to requirements which need to be considered. To elicit these requirements in a goal-oriented manner, we applied different techniques addressing the different stakeholder groups.

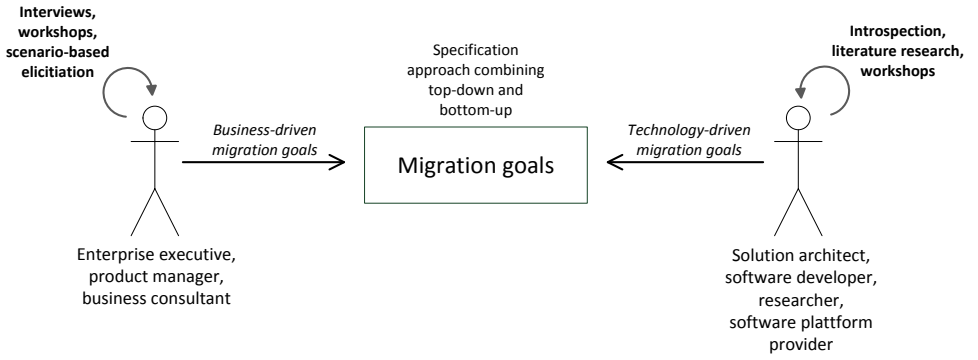


Figure 7.2: Elicitation of Migration Goals

Figure 7.2 sketches an overview of the elicitation process. The business-driven migration goals have been elicited from stakeholders in three different roles: enterprise executive, product manager and business consultant. To elicit the

goals from this stakeholder group, we held interviews and workshops in an initial phase combined with brain storming techniques. The application of scenarios proved very useful to illustrate relevant processes which need to be considered. Further, the application of scenarios facilitates the evaluation of completeness in an early stage. From this information, we gathered the business-driven migration goals which were stated in the interviews and are fulfilled by the scenarios identified.

The elicitation of technology-driven migration goals was performed in a bottom-up manner. Technology-driven goals address platform-independent and plat- form-specific requirements which need to be considered in the migration. Platform-independent requirements were identified and specified by the solution architect and software developer using their technical backgrounds. The platform-specific requirements have been elicited from the software platform provider and by studying the available documentation. From the identified requirements, we elicited technology-driven goals in a bottom-up manner. This allowed us to provide an abstracted view on technical requirements which makes them understandable for stakeholders with a business background as well.

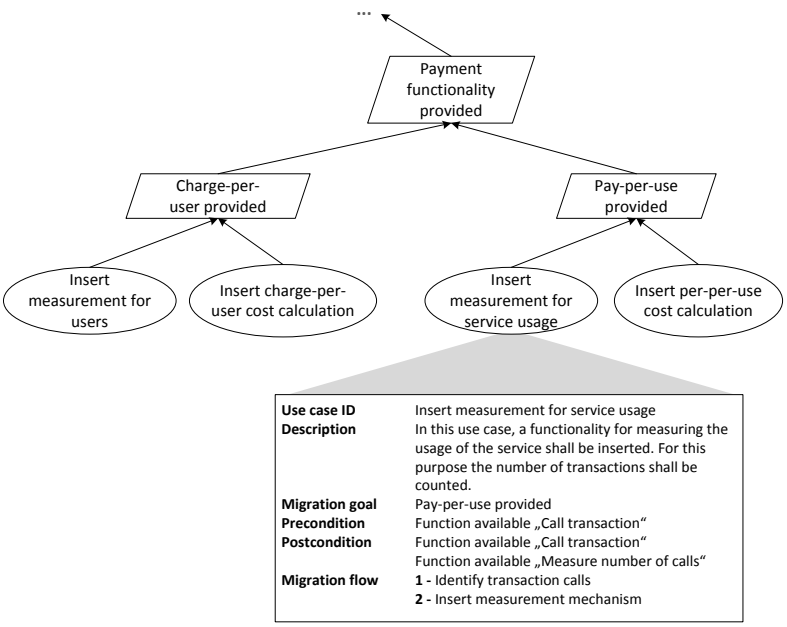


Figure 7.3: Example for the Integration of Goals and Use Cases

Following the iterative decomposition, we refined the elicited goals. Due to the context and setting of the project, an integration with use cases was preferred by the industrial partner. For this purpose, we integrated use cases in our modeling approach by using them to operationalize the leaf goals. An example of the refinement and operationalization is depicted in Figure 7.3. The information used for the specification of use cases is similar to definition of actions as discussed in Section 3.3.3. Hereby, they can be integrated in an easy and understandable way.

To express a temporal order of migration goals, we used the modeling capabilities introduced in Section 3.3.4. In the first step, we specified these relationships between goals. By using this information, we created a project plan for the realization of the migration use cases by applying existing project management techniques.

### 7.1.3 Evaluation Discussion

With respect to evaluation question EQ1, the experiences gathered during the AACC project have shown that our goal modeling approach is well-suited to the applied elicitation techniques. A main insight of this project was the combination of top-down and bottom-up approaches, since it was necessary to consider business-driven as well as technology-driven goals appropriately. In general, the application of existing elicitation techniques, like scenarios, interviews, workshops, introspection etc. worked well with our approach.

Addressing evaluation question EQ2, the experiences from the AACC project have shown that our modeling capabilities were applicable. The explicit definition of temporal dependencies among goals helped to bring the migration goals and the related migration use cases into the desired order. Therefore it was easy to derive a project plan for the execution of the migration.

As discussed, we combined the goal-oriented approach with the application of use cases. The integration could have been performed with minor changes to our approach. Accordingly, evaluation question EQ3 can also be answered positively with respect to the application and integration of use cases.

The application in the AACC project and the evaluation in terms of a project report has some limitations. In the AACC project, six stakeholder were in-

volved in the elicitation and specification of requirements. We recognize that the involvement of six stakeholders is a low number compared to large-scale development projects. Nonetheless, this evaluation aims to validate the compatibility of our approach with existing elicitation and specification techniques. For this purpose, the general applicability of these concepts can be assessed as suitable.

## 7.2 Case Study: Applicability and Expressiveness

### 7.2.1 Evaluation Setup

#### Evaluation Context

The second evaluation aims to investigate the applicability and expressiveness of temporal dependencies in a case study. Unfortunately, only a few case studies are available that define suitable goal models. Usually, the exemplary models comprise about 10-15 goals. For our evaluation, we picked the *Safe Elevator System* case study provided in [Res07]. To the best of our knowledge, this example provides the most complex and largest (w.r.t to the number of goals) model that is available. It comprises a hierarchy of 41 goals and has a maximal decomposition depth of seven levels.

The formalization and refinement of constraints is executed automatically by the implemented tool support. For the execution of these calculations the following computing environment was used:

- Intel Core2 Duo P8600 2.40GHz with 4 GB RAM
- Windows 7 Professional (32 BIT) SP1

#### Evaluation Questions

By performing this case study, we aim to evaluate our approach with respect to the following questions:

**EQ1** Is it suitable to express temporal dependencies in goal models?

**EQ2** Are the provided modeling capabilities applicable to the specification

of temporal dependencies?

**EQ3** Can the temporal dependencies be formalized and refined to verifiable constraints?

### 7.2.2 Evaluation Execution and Results

During this case study we performed three steps: (1) Definition of temporal dependencies, (2) Formalization to verifiable constraints and (3) Refinement of constraints. These steps are described in the following.

**Definition of temporal dependencies.** First, we analyze whether there are temporal dependencies among the defined goals that need to be expressed. By using the modeling capabilities introduced in Section 3.3.4, we identified 13 temporal dependencies in the goal model. An overview about the defined temporal dependencies can be found in Table B.1 in Appendix B.1.

**Formalization to verifiable constraints.** In the next step, the defined temporal dependencies are formalized by the implemented tool support (cf. Chapter 6), according to the translation rules described in Section 4.4.2.

**Refinement of constraints.** Finally, the formalized constraints are refined in an automated manner. For this, we apply the algorithm described in Section 4.4.3 and refine the constraints to the level of fragment goals. An overview about the CTL constraints generated is given in Section B.2.

By using the implemented tool support, steps 2 and 3 have been executed automatically. The calculation of the formal constraints and their refinement through the decomposition hierarchy to the level of fragment goals took 2066 ms.

### 7.2.3 Evaluation Discussion

The performed case study brought several interesting insights for the applicability of the introduced temporal dependency concept. With respect to EQ1, we can state that we identified several temporal dependencies in the given

goal model. Their explicit specification facilitates the consideration of these dependencies in subsequent design and development steps.

An interesting observation was the level of abstraction on which the dependencies have been identified. Most of them have been defined on more concrete goals instead of high-level goals. Hence, we can infer that the level of abstraction might have an impact on the applicability of dependency specifications.

With respect to EQ2, we can summarize that the provided modeling capabilities perfectly match the required level of expressiveness. A more precise definition of order constraints would restrict the flexibility of goal modeling.

By applying the implemented tool support, the temporal dependencies which were identified and modeled have been formalized and refined in an automated manner (EQ3). In the performed study we could not identify any problems regarding performance or quality of the resulting constraints specification.

## 7.3 Case Study: Linguistic Consistency Analysis

### 7.3.1 Evaluation Setup

#### Evaluation Context

In order to demonstrate the capabilities of the linguistic consistency conditions presented above, we test them against three goal models from literature varying in size and domain. Thus, we aim to learn whether our proposed consistency conditions are capable of finding linguistic inconsistencies in these models. Additional information about this evaluation can be found in our publication [PNEM14].

| **Characteristic** | **BOF** [LMSM10] | **CR** [NOH11] | **SES** [Res07] |
|---|---|---|---|
| No. of Goals | 12 | 16 | 21 |
| No. of Actions | 18 | 13 | 0 |
| No. of Decompositions | 12 | 13 | 6 |
| No. of Unique States | 18 | 14 | 5 |
| No. of Unique Objects | 14 | 17 | 18 |

Table 7.1: Details of the Evaluated Goal Models

#### Evaluation Question

In this evaluation, we address the following question:

**EQ1** Are the linguistic consistency analysis techniques suitable for identifying inconsistencies in goal models?

#### Applied Models from Literature

For our evaluation scenario, we reviewed existing literature for available goal models and selected three goal models with varying size, domain and expected quality of goal labeling. Table 7.1 summarizes the main characteristics of these goal models. Our data set includes:

- *Book order fulfillment* (BOF): The goal model presented by Liaskos et al. [LMSM10] addresses a goal model that is similar to the running example of this thesis. The goal model encompasses 12 goals that are decomposed into subgoals and operationalized to actions. In total, the goal model includes 18 actions. By applying the described decomposition of the labels, we can distinguish 18 unique state labels and 14 unique objects which indicates a certain consistency and standardization. The complete goal model is illustrated in Figure D.1 in Appendix D.

- *Cleaning Robot* (CR): The second scenario is taken from [NOH11] and specifies goals that a cleaning robot system has to satisfy when cleaning a room. It comprises 16 goals and 13 actions and is similar to the size and complexity of the BOF case study. The goal labels in this model include 14 unique states and 17 unique objects. Figure D.2 in Appendix D sketches this scenario.

- *Safe Elevator System* (SES): The safe elevator system scenario [Res07] has a slightly different structure to the BOF and CR goal models. In contrast to the other scenarios, it specifies 21 goals and no actions. This might indicate an early stage of the specification or that no operationalization is required in the given application scenario. Moreover, there are only six goal decompositions. The goal model itself contains 18 distinct objects, but only five distinct states. In Appendix D the complete model is shown in Figure D.3.

### 7.3.2 Evaluation Execution and Results

The quantitative results of the three case studies are summarized in Table 7.2 and selected qualitative consistency results are shown in the Tables 7.3 and 7.4. From the results of the *component consistency* conditions, we can infer that all three goal models have such inconsistencies. According to this criterion, the BOF case study performs best, since 6 out of 12 goal decompositions are specified correctly according to their components. As an example, we consider the decomposition of goal *Books ordered* and its decomposed actions *Contact supplier*, *Supplier provides quote*, and *Place order to supplier*. We can observe that the goal and the decomposed actions are all specified by an object and a state or an object and the actual action.

In contrast, the SES model has the worst performance with only 6 consistent goal decompositions out of 21. On closer inspection, we observe many goals are defined by an object without the desired state. An example of such an inconsistency is the goal *No casualties* and its subgoals *Safe entrance and exit* and *Stay safe inside the cage* as depicted in Table 7.4. In this case, the super goal and the first subgoal only specify the object, but lack a state description. Accordingly, we consider this goal decomposition as inconsistent with respect to component consistency.

| | Characteristic | BOF | CR | SES |
|---|---|---|---|---|
| **Component Consistency** | No. of checked goals | 12 | 16 | 21 |
| | No. of consistent goals decompositions | 6 | 6 | 6 |
| | No. of inconsistent goals decompositions | 6 | 10 | 15 |
| **Component Stringency** | No. of checked goals | 12 | 16 | 21 |
| | No. of consistent goals decompositions | 4 | 5 | 16 |
| | No. of inconsistent goals decompositions | 8 | 11 | 5 |
| **Homonym Consistency** | No. of checked goals | 12 | 16 | 21 |
| | No. of homonym consistent goals decompositions | 0 | 1 | 11 |
| | No. of homonym inconsistent goals decompositions | 12 | 15 | 10 |
| **Synonym Consistency** | No. of checked goals | 12 | 16 | 21 |
| | No. of synonym consistent goals decompositions | 11 | 14 | 21 |
| | No. of synonym inconsistent goals decompositions | 1 | 2 | 0 |
| **Goal-Logic Consistency** | No. of decompositions | 12 | 13 | 6 |
| | No. of AND-decompositions | 11 | 11 | 6 |
| | No. of AND-consistent decompositions | 4 | 4 | 0 |
| | No. of AND-inconsistent decompositions | 7 | 7 | 6 |
| | No. of OR-Decompositions | 1 | 2 | 0 |
| | No. of OR-consistent decompositions | 0 | 0 | 0 |
| | No. of OR-inconsistent decompositions | 1 | 2 | 0 |

Table 7.2: Results of Linguistic Consistency Checks

| Consistency Criterion | Super Goal | Sub Goals | Consistent (y/n) | Explanation |
|---|---|---|---|---|
| **Component Consistency** | No casualties | Safe entrance and exit<br>Stay safe inside the cage | n | Action is missing in super goal and first sub goal |
| | Books ordered | Contact supplier<br>Supplier provides price<br>Place order to supplier | y | All goals contain an action and an object |
| **Component Stringency** | Books delivered | Deliver to courier<br>Courier delivers books to customer | n | Object *book* missing in both sub goals |
| | Station reachable | Find station<br>Get close to station | y | Object *station* present in all goals |
| **Homonym Consistency** | Dust reachable | Find dust<br>Get close to dust | n | Homonyms found: find, dust |
| | Main crawler available | – | y | No Homonyms found |

Table 7.3: Qualitative Results of Consistency Checks (Part I)

The second consistency check addresses the criterion *component stringency*. The results of this check identify the SES goal model as the most consistent as it consistently narrows down the objects in the goal decomposition. For this example, the analysis returns that 16 of 21 goal decompositions are consistent. As an example for a consistent decomposition, we choose the goal *Station reachable* from the CR model. This goal is decomposed into the actions *Find station* and *Get close to station*. Since the goal and its actions deal with the same object, the decomposition is assessed as consistent.

In the BOF goal model, our analysis detected the highest number of component stringency inconsistencies. Only 4 out of 12 decompositions were recognized as stringent. This result may indicate that an intermediate decom-

| Consistency Criterion | Super Goal | Sub Goals | Consistent (y/n) | Explanation |
|---|---|---|---|---|
| **Synonym Consistency** | Battery maintained | Observe battery level<br>Station reachable<br>Charge battery | n | Synonyms found: maintain, observe |
| | Books acquired | Supplier ships books<br>Books arrive at warehouse | y | No Synonyms found |
| **AND-Consistency** | No casualties | Safe entrance and exit<br>Safe stay inside the cage | n | Closeness: 0.064 |
| | Payment via money order | Customer issues money order<br>Customer sends money order<br>Receive money order | y | Closeness: 0.444 |
| **OR-Consistency** | Payment received | Payment via money order<br>Payment via credit card | n | Closeness: 0.5 |

Table 7.4: Qualitative Results of Consistency Checks (Part II)

position layer is actually missing for a stringent decomposition. For example, the decomposition of the goal *Books delivered* fails to specify the object *book* for both sub goals. This means that it is unclear what object shall be delivered to the courier or to the customer. The example illustrates that a lack of stringency negatively effects the understandability.

The check for *homonym consistency* detects several homonym inconsistencies in all three goal models, i.e. each goal decomposition contains at least one homonym. However, we have to qualify this observation as nearly every action or object has multiple word senses according to lexical databases. In the evaluation, the SES model scores best with respect to this criterion. The low

number of homonyms is related to the discussed lack of state descriptions in the goal label. As a result, states which could be ambiguous are missing. An example for identified homonyms can be found in the CR model, by the goal *Dust reachable* and the related actions *Find dust* and *Get close to dust*. In this case, the action term *to find* and the object *dust* are recognized as homonyms.

In addition to *homonym consistency*, we also evaluated the *synonym consistency* for each goal decomposition. In contrast to the homonym case, we could identify only a small number of inconsistencies with respect to synonymy. The BOF and CR goal models contain synonym violations which are caused by synonym actions. We consider goal *Battery maintained* and the action *Observe battery level* from the CR model as an example. This decomposition is considered as inconsistent with respect to synonyms, since the verbs *to maintain* and *to observe* were recognized as synonyms.

Finally, we evaluated the goal-logic consistency by analyzing the *AND-consistency* and *OR-consistency* of all three goal models. For this purpose, we excluded those decompositions which specify none or only one subgoal or action as this criterion requires a pair-wise comparison of the subgoals and actions in the goal decomposition. It became apparent that all three goal models contain several inconsistencies.

First, we consider the consistency of AND-decompositions. The BOF model appears to be the most consistent as 4 of 11 decompositions satisfy this criterion. For illustration purposes, we consider the goals *No casualties* of the SES model and *Payment via money order* of the BOF model. For the first example, the goal *No casualties* is decomposed into the subgoals *Safe entrance and exit* and *Safe stay inside the cage*. For this decomposition, the Lin measure calculates a closeness of 0.064 which reflects a very loose relation and might indicate an inconsistency with respect to the AND-decomposition. In the second example, the Lin measure evaluates to 0.444 which indicates a moderate relation. Accordingly, we consider this decomposition as consistent with respect to the AND-decomposition.

Second, we also analyzed the consistency of OR-decompositions in the goal models. For the OR-decomposition, the results are even more striking. The approach detects inconsistencies for all OR-decompositions in the CR and the BOF model which indicates that the subgoals are extremely close to each other although the opposite might be intended.

### 7.3.3 Evaluation Discussion

The performed case study evaluation has shown that the analysis techniques which were applied detect inconsistencies in each model. Further, the different types of inconsistencies have been detected successfully in these models. We can conclude that the linguistic consistency analysis developed and applied here is suitable for identifying inconsistencies and therefore answer the evaluation question EQ1 positively.

In addition to the concrete evaluation question the evaluation brought implications for the application of our consistency analysis approach in practice and research. An important implication for practice is that the proposed techniques can be integrated into goal modeling approaches. Inconsistent goals or goal decompositions can be identified and the stakeholders are pointed to these inconsistencies to resolve them directly. Hereby, the integration of the analysis approach improves the specification quality during the development process of goal models. For existing goal models, the linguistic consistency checks can help stakeholders to effectively validate and consolidate created goal models.

Our approach to business goal modeling is based on the well-known KAOS approach. Accordingly, we performed the evaluation on KAOS goal models as well. As discussed in Section 2.2.3, goal models can be defined in different notations, such as i* or Tropos. Analogously, goal models using any of these notations have to be consistent in order to avoid risks for the software development project and to improve the chances for a successful software deployment [ABD+04]. The consistency criteria are conceptualized as notation-independent and are thus also applicable to i*, Tropos and other goal modeling notations since they show similar linguistic characteristics to the KAOS goal models that were subject to this evaluation. Addressing the applicability to different notations, we can state that the suitability of the analysis capabilities (EQ1) is not restricted to a single goal modeling notation.

## 7.4 Experiment: Traceability Analysis

### 7.4.1 Evaluation Setup

#### Evaluation Context

The experiment for traceability analysis has been performed in a master thesis, described in [Hah14]. In this experiment, the traceability analysis approach is applied to our running example. For this, we rely on the business goal model depicted in Figure 3.12 and a derived business process model shown in Appendix C.

To consider different possible evolutions, we discuss the following scenarios in the experiment:

- Scenario 1 - Definition of a new goal
- Scenario 2 - Deletion of an existing goal
- Scenario 3 - Definition of a new action in the business process model
- Scenario 4 - Deletion of an existing action in the business process model
- Scenario 5 - Reordering of actions in the business process model

#### Evaluation Questions

In particular the case study seeks to answer the following two questions:

**EQ1** Is the annotation algorithm suitable for the purpose of a traceability analysis?

**EQ2** Can inconsistencies be detected and repaired in a sufficient manner?

### 7.4.2 Evaluation Execution and Results

#### Preprocessing

Before we start with the execution of the different evolution scenarios, preprocessing is performed to provide the foundation for the subsequent traceability analysis. Hereby, the implemented tool support calculates the immediate and accumulated effects for the considered business process.

#### Scenario 1: Definition of a new goal

In this scenario, we consider that a new business goal is defined. We assume that the existing payment options (money order, credit card) are not sufficient to stay competitive anymore. Hence, we define a new business goal *Payment via paypal* that is operationalized by three actions *Get paypal account information*, *Forward to paypal account* and *Initiate payment*. The goal model is adapted accordingly, as sketched in Figure 7.4.

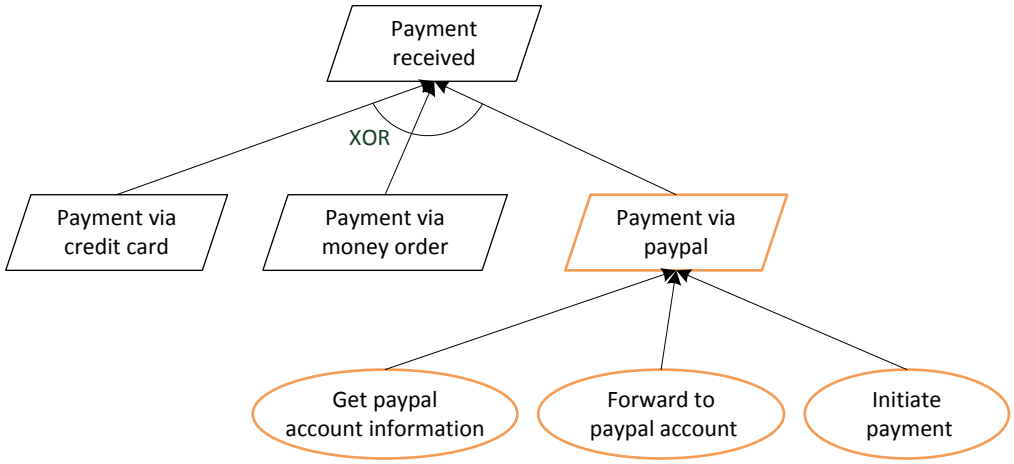


Figure 7.4: Excerpt from the Updated Goal Model extended by the new Goal

We consider that the business process model is not updated accordingly and the goal is not considered appropriately. To analyze these models, we use the aforementioned approach to traceability analysis to detect inconsistencies among them. The analysis algorithm detects that the new goal is not achieved by the business process and proposes to add all actions that operationalize the

new goal. To enable this solution, the tool requires a manual definition of a new fragment to determine the position for the insertion of the actions.

**Scenario 2: Deletion of an existing goal**

In the second scenario, we consider that an existing goal has become obsolete and is removed from the model accordingly. If the business process model is not adapted, it contains actions which are not related to a current business goal. In our example, we consider that no printed receipts shall be sent to the customer anymore. Therefore, the obsolete goal *Printed receipt sent* and all connected actions are removed from the goal model. The traceability analysis results in a warning. It highlights the additional effects caused by the actions that are still executed in the modeled business process. To resolve this problem, the tool proposes to delete these actions from the business process model.

**Scenario 3: Definition of a new action in the business process model**

The definition of a new action in the business process model and the assignment to an existing fragment can result in two different warnings depending on the position and the effect caused by the action. Either the action causes an additional effect or it overwrites the desired achievement state of the related fragment goal. The traceability analysis returns the corresponding warning and the requirements engineer can decide whether the action shall be deleted, assigned to another or new fragment or whether the goal achievement state definition needs to be adapted in order to match the new state.

**Scenario 4: Deletion of an action in the business process model**

In the fourth scenario, we consider that an action is deleted in the business process model and the goal model is not updated accordingly. We consider two deletions addressing different positions of actions.

An action within a sequence of actions (not the last one) is deleted, the resulting gap leads to an error with respect to the valid order of actions (mismatching of pre- and postconditions). If the deletion of the action is desired, the pre- and postconditions of the previous and following actions need to be

adapted.

A deletion of the last action in an action sequence changes the final state that is achieved in the corresponding fragment. Accordingly, an error is returned since the fragment goal is not achieved on all execution paths. One way to fix this problem is to redefine the achievement state of the related business goal.

**Scenario 5: Reordering of actions in the business process model**

In the last scenario, we consider the reordering of actions within a business process model. Hereby, the position of the action is changed and it is assigned to another business process fragment. This means that the reordering affects two different fragments: the fragment to which the actions are added and the fragment the actions are deleted from.

The analysis of the first fragment results in a warning and proposes the resolution of this effect by the removal of the action or the assignment to a new fragment (which needs to be created manually). In the second fragment, an error is returned since the related fragment goal is no longer achieved. This can be resolved by a redefinition of the fragment goal or corresponding changes of the business process model.

### 7.4.3 Evaluation Discussion

The experiment has shown that the conceptual approach and the implemented tool support for traceability analysis worked well for the addressed example. Hence, we can positively answer the evaluation questions EQ1 and EQ2. The preprocessing approach is suitable for annotating an existing business process model. Furthermore, the analysis capabilities identified the errors caused by the evolutionary changes constructed in the different scenarios.

Another observation from the experiment is the need for manual input by the requirements engineer. This input is required to decide which model (goal model or business process model) represents the desired state. Based on this decision suitable repair strategies can be selected.

## 7.5 Threats to Validity

So far, we have evaluated our approach through a project report, two case studies and an experiment. In this section, we discuss the validity of the results. In [CCD79] a categorization for threats to the validity of an evaluation is given which distinguished between (1) conclusion validity, (2) internal validity, (3) construct validity and (4) external validity. We apply this classification scheme and discuss the validity of our evaluation with respect to these dimensions. References for concrete threats for each dimension are given in [WRH+12]. In the following we discuss our evaluation approach with respect to relevant threats.

### 7.5.1 Conclusion Validity

The conclusion validity is concerned with the relationship between the treatment and the related outcome. It addresses threats which impact the ability to draw the correct conclusions between the treatment of an evaluation and the resulting outcome.

A common threat to the conclusion validity is that the researcher performing the evaluation influences the results according to the expected results. This threat is termed *fishing* since a specific outcome is searched for which impacts the objectivity of the analysis [WRH+12].

To handle this threat, we involved third parties in the execution and analysis (e.g. master students performing the case studies as part of their theses). The case study about linguistic consistency analysis has been published and supervised by external reviewers. Furthermore, both case studies are based on existing examples from other researchers. Hereby, we can reduce the risk that *fishing* impacts the setting of the goal models. Nonetheless, a negative impact of the fishing threat cannot be excluded with respect to the selection of the models from literature and the defined settings of the evaluation.

The reliability of the applied measures is another threat to the conclusion validity. The main idea is that two measurements of the same phenomenon should result in the same outcome. A main prerequisite is the usage of objective measures which are more reliable than the application of subjective

measures.

The automated analysis capabilities, like the linguistic consistency analysis, result in concrete results in terms of objective measures. Furthermore, these case studies and experiments can be repeated and result in the same outcome after every execution. Evaluating the applicability of software engineering methods with objective measures is very difficult. The evaluation in the project report relies on subjective metrics and may affect the conclusion validity.

### 7.5.2 Internal Validity

While the conclusion validity addresses the existence of a relationship between treatment and outcome, this validity is concerned with the validation of whether it is a causal relation. Through this, it shall be ensured that the outcome is actually caused by the treatment.

The effect caused by the selection of artifacts which are used for the evaluation is termed *instrumentation*. A bad selection and design of the experiment affects the overall validity negatively. To minimize this threat, we applied the evaluation on goal models taken from literature. The selection of the models can still affect the results, but the atomic objects in these models (goals and actions) have a certain degree of maturity and are consistent to each other in the corresponding models. Hereby, we can reduce the risks related to *instrumentation*.

### 7.5.3 Construct Validity

Construct validity is concerned with the relation between the evaluation setting and object or construct which is under study. A common design threat for the construct validity is the *mono-operation bias*. This threat considers that if an experiment is performed with a single independent variable, e.g. a document, the evaluation may be under represented.

To address this threat, we applied several goal models in the evaluation of the linguistic consistency analysis. For the evaluation of applicability and expressiveness we picked a comparably large and complex goal model, which

comprises several temporal dependencies of both types to consider various refinement paths.

Due to the underlying objectives of the evaluation, the experimenters' bias can affect the results (*experimenter expectancies*). During the project which is described in the project report, different stakeholders were involved who did not have specific expectations for the evaluation. Hereby, we can reduce this threat in this evaluation. The other evaluations are performed in an automated manner. The related risks are similar to the descriptions of the threat *fishing*.

### 7.5.4 External Validity

The external validity addresses the generalization of the relationship between treatment and outcome. This means that it needs to be checked whether the results can be generalized from the limited scope of the evaluation.

One limitation for the generalization of the results is the selection of the wrong persons within the evaluation. If the evaluation is performed based on persons, person groups or roles which are not representative for the persons representing the desired generalization, e.g. a specific group of stakeholders. In the project report, we summarize the experiences from stakeholders in different roles and with different backgrounds. Each of them represents a target group that is relevant for the generalization of the applicability of our approach. The case studies and the experiment have been executed by a software engineer with a strong background in requirements engineering. He also represents the target group for the application of the analysis capabilities.

A second relevant threat is the *interaction of setting and treatment*. This is concerned with using a non-representative set of materials for the evaluation. We address this threat by selecting the largest and most complex goal models that are available. Nonetheless, the work in [VL04] discusses that industrial case studies can include up to several hundred goals. Since those models describe critical strategic requirements of enterprises, such models are not available in the public domain.

## 7.6 Summary and Discussion

In this chapter, we have presented the evaluation of our approach through a project report, two case studies and an experiment. In the case studies, we applied existing goal models taken from literature and performed the different steps by using the implemented tool support. In order to assess the validity of our evaluation, we discussed conclusion validity, internal validity, construct validity and external validity.

# Chapter 8

# Conclusion

In this final chapter, we conclude this thesis. In Section 8.1 we summarize and discuss the contributions of our approach. An overview about future work in the field of goal-oriented business process engineering is given in Section 8.2.

## 8.1 Summary of Contributions

To achieve the objectives of this thesis, we presented a solution that comprises three parts. Figure 8.1 sketches an overview about the different parts and the corresponding scientific contributions. In the following, we discuss these contributions in detail.

#### Business Goal Modeling

The first part of our approach deals with business goal modeling. We developed and introduced the concept of temporal dependencies among goals. Hereby, we enable the consideration of a temporal dimension in goal models.

To provide the foundation for a composition of valid business processes, we introduced an approach to ensure the composability of actions during the business goal specification. By applying object life cycle models, the composability of actions is checked and possible mismatches can be identified and resolved in a systematic manner. In addition, the constructed object life cy-

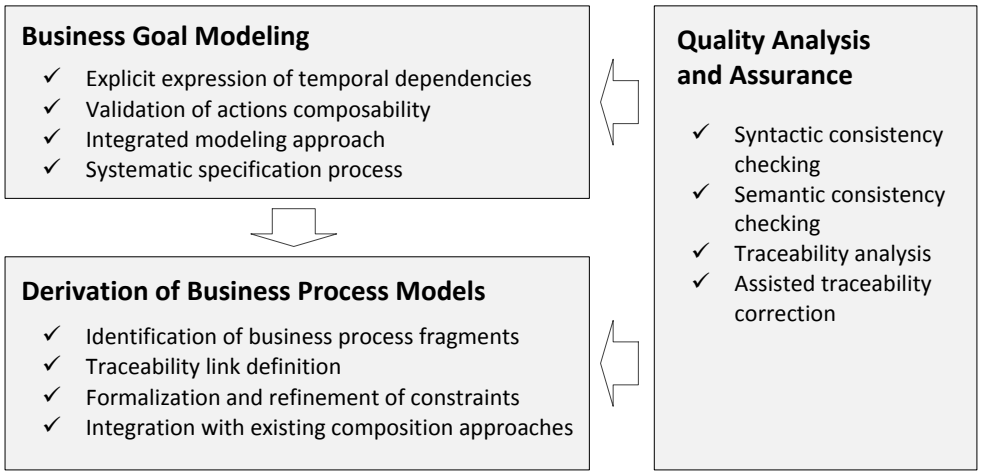


Figure 8.1: Summarized Contributions of our Approach

cle models visualize valid states and state transitions leading to an improved understandability.

We proposed an integrated specification approach that considers business goals as well as the relevant context in terms of business objects and object life cycle models. To provide a formal language specification comprising these elements, we presented a meta model that considers the interrelations among the different modeling elements.

To facilitate the application of the business goal modeling, we described a systematic specification process that ensures a consistent specification of goal achievement states through the goal model hierarchy.

**Derivation of Business Process Models**

For the composition of business processes, we leverage the concept of business process fragments. We described the identification of business process fragments from a given business goal model. Further, we presented a meta model for the definition of explicit traceability links between related business goals and business process fragments.

To enable the composition of the identified fragments according to the speci-

fied dependencies, we proposed an algorithm for the systematic formalization and refinement. The inferred constraints are verifiable and can be applied for the calculation of a valid fragment composition.

Bringing together the traceable business process fragments and the formalized constraints, we described how existing composition approaches can be leveraged to derive a business process model which is consistent to the related business goal model. By using the context information provided, we enriched the resulting business processes with information about the processed business objects and the state changes caused by the processing.

**Quality Analysis and Assurance**

Our quality analysis and assurance approach comprises the linguistic consistency analysis and the traceability analysis.

In the linguistic consistency analysis, we developed syntactic and semantic consistency checks that enable the validation of natural language goal specifications. Although we rely on existing analysis algorithms, to the best of our knowledge no existing approach supports their application to goal models. As demonstrated in the case study (cf. Section 7.3) the approach is suitable to identify linguistic inconsistencies in goal models.

The traceability analysis supports the automated validation of completeness and relevance of a business goal model and a derived business process model. Goals which are not achieved sufficiently (completeness) as well as business process actions that do not contribute to a business goal (relevance) are identified in an automated manner. According to the identified scenario, our approach proposes repair strategies to resolve the identified inconsistencies.

## 8.2 Future Work

Although the presented approach for goal-oriented business process engineering provides a complete piece of research, we still see several directions for future research.

**Integration with Adaptive and On-the-Fly Business Process Composition Approaches.**

Nowadays, business processes are not only considered as a model that describes a certain perspective but as a runtime artifact. Process-aware information systems (PAIS) or on-the-fly process composition address the adaptation or re-composition of business processes at runtime in an automated or autonomous manner. Our work, especially the definition and analysis of traceability, seems to indicate a promising research direction in the preservation of consistency between adapted business process models and business goals. The validation of consistency in a highly dynamic environment raises new challenges, e.g. with respect to scalability and the level of automation of the quality analysis and assurance.

**Extended Linguistic Consistency Validation Capabilities**

The checks that have been introduced for syntactic and semantic consistency provide valuable results. Nonetheless, the current approach lacks the consideration of the context. This is particularly relevant for improving the preciseness of the homonym and synonym detection. Consideration of the context raises new research challenges on the appropriate modeling as well as the integration into the existing validity conditions. Another interesting research challenge is the improvement of the decomposition-logic consistency validation by the application of ontology-oriented technologies.

We also presented an approach for the linguistic consistency of goal models. Existing work in [PLM13] deals with the analysis of business process models in repositories. Both approaches can be combined in order to evaluate whether a business goal model and a derived business process model are linguistically consistent with each other. For this purpose, new consistency conditions need to be defined. Such an analysis could also provide potential for

the consideration of semantics in the traceability analysis. Hereby, it could be possible to check if actions in a business process are semantically close to the goal they are contributing to.

#### Multi-Process Derivation

Our approach to the goal-oriented business process engineering is suitable to derive exactly one business process model from a certain business goal model. If we consider very large or complex business goal models which address divergent concerns, it might be useful to derive more than one business process model, as discussed in [SW07]. This means that a goal model needs to be divided into submodels which are used to derive different business process models. In order to support this partition into submodels, it needs to be decided which goals are considered in the same business process. For this purpose, it would be an interesting research challenge to develop metrics that predict the quality of resulting business processes, e.g. based on business process quality metrics [VCM+07, HMR09] or existing work in the field of component identification [Szy02]. To enable the derivation of multiple business process models from our approach, the defined constraints also need to be resolved or represented as inter-process dependencies expressing the temporal order of process executions.

#### Consideration of Non-Functional Goals

The composition of business processes is based on functional goals and their operationalization to actions. While these goals are suitable to derive required functionalities, non-functional requirements are not considered. Such aspects can also be defined on the level of goals as described in [Gli07, CPL09]. In the context of business process modeling, non-functional requirements can be considered in terms of *Key Performance Indicator* (KPI) or *Service Level Agreement* (SLA) definitions. For this purpose, new refinement strategies need to be developed that enable the specification of precise and measurable KPIs and SLAs based on non-functional goals. Furthermore, explicit relations between functional and non-functional goals need to considered and represented in the derived business process models.

**Evaluation in a Large-Scale Industrial Case Study**

A main drawback of our evaluation is the lack of availability of a real-world example in terms of a large-scale industrial case study. Hence, the generalization of our approach with respect to applicability and scalability is limited to the size and complexity of the available models. The execution of a case study based on a large-scale real-world example would help to improve the evaluation of our approach.

# List of Figures

# List of Tables

# Bibliography


[AB01] Wil M. P. van der Aalst and Twan Basten. Identifying Commonalities and Differences in Object Life Cycles Using Behavioral Inheritance. In *Applications and Theory of Petri Nets 2001*, number 2075 in Lecture Notes in Computer Science, pages 32–52. Springer Berlin Heidelberg, 2001.

[ABD+04] Alain Abran, Pierre Bourque, Robert Dupuis, James W. Moore, and Leonard L. Tripp. *Guide to the Software Engineering Body of Knowledge - SWEBOK*. IEEE Press, 2004.

[ADG09] Raian Ali, Fabiano Dalpiaz, and Paolo Giorgini. A Goal Modeling Framework for Self-contextualizable Software. In *Enterprise, Business-Process and Information Systems Modeling*, volume 29 of *Lecture Notes in Business Information Processing*, pages 326–338. Springer Berlin Heidelberg, 2009.

[AKRU09a] Dalal Alrajeh, Jeff Kramer, Alessandra Russo, and Sebastin Uchitel. Learning operational requirements from goal models. In *Proceedings of the 31st International Conference on Software Engineering*, pages 265–275. IEEE Computer Society, 2009.

[AKRU09b] Dalal Alrajeh, Jeff Kramer, Alessandra Russo, and Sebastin Uchitel. Learning operational requirements from goal models. In *Proceedings of the 31st International Conference on Software Engineering*, ICSE '09, pages 265–275. IEEE Computer Society, 2009.

[AMP94] AnnieI Antón, W.Michael McCracken, and Colin Potts. Goal decomposition and scenario analysis in business process reengineering. In Gerard Wijers, Sjaak Brinkkemper, and Tony Wasserman,

editors, *Advanced Information Systems Engineering*, volume 811 of *Lecture Notes in Computer Science*, pages 94–104. Springer Berlin Heidelberg, 1994.

[Ant96] Annie I Anton. Goal-based requirements analysis. In *Proc. of the Second Int. Conf. on Requirements Engineering*, pages 136–144, 1996.

[ARWB11] Syed Adeel Ali, Partha Roop, Ian Warren, and Zeeshan Ejaz Bhatti. Unified Management of Control Flow and Data Mismatches in Web Service Composition. In *Proceeding of the IEEE 6th International Symposium on Service Oriented System Engineering*, pages 93–101. IEEE, 2011.

[AT90] R. Agarwal and M. R. Tanniru. Knowledge Acquisition Using Structured Interviewing: An Empirical Investigation. *J. Manage. Inf. Syst.*, 7(1):123–140, 1990.

[BBH+96] Cornelia Boldyreff, EL Burd, RM Hather, Malcolm Munro, and EJ Younger. Greater understanding through maintainer driven traceability. In *Fourth Workshop on Program Comprehension*, pages 100–106. IEEE, 1996.

[BC03] Marco Benedetti and Alessandro Cimatti. Bounded model checking for past LTL. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 18–33. Springer, 2003.

[BGH+07] Kamal Bhattacharya, Cagdas Gerede, Richard Hull, Rong Liu, and Jianwen Su. Towards Formal Analysis of Artifact-Centric Business Process Models. In *Business Process Management*, number 4714 in Lecture Notes in Computer Science, pages 288–304. Springer Berlin Heidelberg, 2007.

[BGK+07] Manfred Broy, Eva Geisberger, Jürgen Kazmeier, Arnold Rudorfer, and Klaus Beetz. Ein Requirements-Engineering-Referenzmodell. *Informatik-Spektrum*, 30(3):127–142, 2007.

[BGM09] Volha Bryl, Paolo Giorgini, and John Mylopoulos. Supporting Requirements Analysis in Tropos: A Planning-Based Approach. In *Agent Computing and Multi-Agent Systems*, pages 243–254. Springer, 2009.

[BHL+07] Victor Basili, Jens Heidrich, Mikael Lindvall, Jürgen Münch, Myrna Regardie, Dieter Rombach, Carolyn Seaman, and Adam Trendowicz. Bridging the gap between business strategy and software development. In *ICIS*, page 25, 2007.

[BMMZ06] Volha Bryl, Fabio Massacci, John Mylopoulos, and Nicola Zannone. Designing Security Requirements Models Through Planning. In *CAiSE'06*, pages 33–47. Springer, 2006.

[BO00] Linden J. Ball and Thomas C. Ormerod. Putting ethnography to work: the case for a cognitive ethnography of design. *International Journal of Human-Computer Studies*, 53(1):147–168, 2000.

[Boe84] Barry W. Boehm. Verifying and validating software requirements and design specifications. *Software, IEEE*, 1(1):75–88, 1984.

[Boe88] Barry W Boehm. Understanding and controlling software costs. *Journal of Parametrics*, 8(1):32–68, 1988.

[Bon13] Bonitasoft. Bonita Studio, 2013. http://de.bonitasoft.com/produkte/bonita-open-solution-open-source-bpm.

[BPG+04] Paolo Bresciani, Anna Perini, Paolo Giorgini, Fausto Giunchiglia, and John Mylopoulos. Tropos: An Agent-Oriented Software Development Methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3):203–236, 2004.

[BRF14] P. Bourque and eds. R.E. Fairley. Guide to the software engineering body of knowledge, version 3.0. IEEE Computer Society, 2014. www.swebok.org.

[Bro87] Bradley J Brown. Assurance of software quality. Technical Report CMU/SEI-CM-7-1-1, CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST, 1987.

[BRU00] Jörg Becker, Michael Rosemann, and Christoph von Uthmann. Guidelines of business process modeling. In *Business Process Management*, number 1806 in Lecture Notes in Computer Science, pages 30–49. Springer Berlin Heidelberg, 2000.

[Bur95] Carol Burt. OMG BOMSIG Survey with published definition of a business object. Technical Report OMG document 95-02-04, OMG, 1995. www.omg.org.

[Car05] Edward R Carroll. Estimating software based on use case points. In *Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 257–265. ACM, 2005.

[Cas97] Cory Casanave. Business-object architectures and standards. In *Business Object Design and Implementation*, pages 7–28. Springer, 1997.

[CAS03] Gilberto A.A. Cysneiros, Filho Andrea, and Zisman G Spanoudakis. Traceability Approach for i* and UML Models. 2003.

[CCD79] Thomas D. Cook, Donald Thomas Campbell, and Arles Day. *Quasi-experimentation: Design & Analysis Issues for Field Settings*. Houghton Mifflin Boston, 1979.

[CES86] Edmund M. Clarke, E. Allen Emerson, and A. Prasad Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 8(2):244–263, 1986.

[CGP99] Edmund M. Clarke, Orna Grumberg, and Doron Peled. *Model Checking*. MIT press, 1999.

[Cle90] Rance Cleaveland. Tableau-Based Model Checking in the Propositional Mu-Calculus. *Acta Informatica*, 27(8):725–747, 1990.

[CPL09] Lawrence Chung and Julio Cesar Sampaio Prado Leite. On Non-Functional Requirements in Software Engineering. In *Conceptual Modeling: Foundations and Applications*, volume 5600, pages 363–379. Springer Berlin Heidelberg, 2009.

[CST05] Artur Caetano, António Rito Silva, and José Tribolet. Using Roles and Business Objects to Model and Understand Business Processes. In *Proceedings of the 2005 ACM Symposium on Applied Com-*

*puting*, pages 1308–1313. ACM, 2005.

[DDH+06] A. Davis, O. Dieste, A. Hickey, N. Juristo, and A.M. Moreno. Effectiveness of Requirements Elicitation Techniques: Empirical Results Derived from a Systematic Review. In *Proceeding of the 14th IEEE International Conference Requirements Engineering*, pages 179–188, 2006.

[DeM79] Tom DeMarco. *Structured analysis and system specification*. Yourdon Press, 1979.

[DFvL91] Anne Dardenne, Stephen Fickas, and Axel van Lamsweerde. Goal-directed concept acquisition in requirements elicitation. In *Proceedings of the 6th International Workshop on Software Specification and Design*, pages 14–21. IEEE Computer Society Press, 1991.

[DNV90] Rocco De Nicola and Frits Vaandrager. Action versus state based logics for transition systems. In *Semantics of Systems of Concurrent Processes*, pages 407–419. Springer, 1990.

[DOJ+93] Alan Davis, Scott Overmyer, Kathleen Jordan, Joseph Caruso, Fatma Dandashi, Anhtuan Dinh, Gary Kincaid, Glen Ledeboer, Patricia Reynolds, Pradip Sitaram, et al. Identifying and Measuring Quality in a Software Requirements Specification. In *First International Software Metrics Symposium*, pages 141–152. IEEE, 1993.

[DP05] Åsa G. Dahlstedt and Anne Persson. Requirements interdependencies: state of the art and future challenges. In *Engineering and managing software requirements*, pages 95–116. Springer, 2005.

[DP06] Florian Deissenboeck and Markus Pizka. Concise and consistent naming. *Software Quality Journal*, 14(3):261–282, 2006.

[DvLF93] Anne Dardenne, Axel van Lamsweerde, and Stephen Fickas. Goal-directed requirements acquisition. *Science of Computer Programming*, 20(1-2):3–50, 1993.

[EAG06] Angelina Espinoza, Pedro P. Alarcon, and Juan Garbajosa. Analyzing and systematizing current traceability schemas. In *30th Annual IEEE/NASA Software Engineering Workshop*, pages 21–32.

IEEE, 2006.

[EE97] Jürgen Ebert and Gregor Engels. Specialization of Object Life Cycle Definitions. Fachberichte informatik 19/95, University of Koblenz-Landau, 1997.

[EH91] Michael Edwards and Steven L. Howell. A methodology for systems requirements specification and traceability for large real time complex systems. Technical Report NAVSWC-TR-91-584, NAVAL SURFACE WARFARE CENTER SILVER SPRING MD, 1991.

[EHHS00] Gregor Engels, Jan Hendrik Hausmann, Reiko Heckel, and Stefan Sauer. Dynamic Meta-Modeling: A Graphical Approach to the Operational Semantics of Behavioral Diagrams in UML. In *Proc. of the 3rd Int. Conf. on the Unified Modeling Language (UML 2000)*, volume 1939, pages 323–337. Springer, 2000.

[ELGS05] Angelina Espinoza Limón and Juan Garbajosa Sopeña. The need for a unifying traceability scheme. 2005.

[Eme90] E. Allen Emerson. Temporal and modal logic. *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*, 995:1072, 1990.

[EQJvS11] Wilco Engelsman, Dick Quartel, Henk Jonkers, and Marten van Sinderen. Extending enterprise architecture modelling with business goals and requirements. *Enterprise Information Systems*, 5(1):9–36, 2011.

[Erl05] Thomas Erl. *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall PTR, 2005.

[Fer11] Daniel Méndez Fernández. *Requirements Engineering: Artefact-Based Customisation*. PhD thesis, 2011.

[FESVDS07] A. Förster, G. Engels, T. Schattkowsky, and R. Van Der Straeten. Verification of Business Process Quality Constraints Based on Visual Process Patterns. In *First Joint IEEE/IFIP Symposium on Theoretical Aspects of Software Engineering, 2007. TASE '07*, pages 197–208, 2007.

[FF94] William Foddy and William H. Foddy. *Constructing Questions for Interviews and Questionnaires: Theory and Practice in Social Research*. Cambridge University Press, 1994.

[FPKB10] Daniel Méndez Fernández, Birgit Penzenstadler, Marco Kuhrmann, and Manfred Broy. A Meta Model for Artefact-Orientation: Fundamentals and Lessons Learned in Requirements Engineering. In *Model Driven Engineering Languages and Systems*, pages 183–197. Springer, 2010.

[FPMT01] A. Fuxman, M. Pistore, J. Mylopoulos, and P. Traverso. Model checking early requirements specifications in tropos. In *Proceeding of the Fifth IEEE Int. Symposium on Requirements Engineering*, pages 174–181, 2001.

[GBB+06] Eva Geisberger, Manfred Broy, Brian Berenbach, Juergen Kazmeier, Daniel Paulish, and Arnold Rudorfer. Requirements engineering reference model (REM). *Technische Universität München*, 2006.

[Ger13] Christian Gerth. *Business Process Models. Change Management*. Number 7849 in Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013.

[GF94] O. C. Z. Gotel and A. C. W. Finkelstein. An Analysis of the Requirements Traceability Problem. In *Proceedings of the First International Conference on Requirements Engineering*, pages 94–101, 1994.

[GFL+13] Gonzalo Génova, José M. Fuentes, Juan Llorens, Omar Hurtado, and Valentín Moreno. A framework to measure and improve the quality of textual requirements. *Requirements Engineering*, 18(1):25–41, 2013.

[GHSW09] Guido Governatori, Jörg Hoffmann, Shazia Sadiq, and Ingo Weber. Detecting regulatory compliance for business process models through semantic annotations. In *Business Process Management Workshops*, pages 5–17. Springer, 2009.

[Gil00] David Gillibrand. Technical Opinion: Essential Business Object Design. *Communications of the ACM*, 43(2):117–119, 2000.

[GK07] A.K. Ghose and G. Koliadis. Verifying Semantic Business Process Models. *Faculty of Informatics-Papers*, 2007.

[GKMP04] Paolo Giorgini, Manuel Kolp, John Mylopoulos, and Marco Pistore. The Tropos Methodology. In *Methodologies and Software Engineering for Agent Systems*, volume 11 of *Multiagent Systems, Artificial Societies, and Simulated Organizations*, pages 89–106. Springer US, 2004.

[GL93] Joseph A. Goguen and C. Linde. Techniques for requirements elicitation. In *Proceedings of IEEE International Symposium on Requirements Engineering*, pages 152–164, 1993.

[Gli07] Martin Glinz. On Non-functional Requirements. In *15th IEEE International Requirements Engineering Conference*, pages 21–26. IEEE, 2007.

[GMS05] Paolo Giorgini, John Mylopoulos, and Roberto Sebastiani. Goal-oriented requirements analysis and reasoning in the Tropos methodology. *Engineering Applications of Artificial Intelligence*, 18(2):159–171, 2005.

[GPW06] J. Gordijn, M. Petit, and R. Wieringa. Understanding Business Strategies of Networked Value Constellations Using Goal- and Value Modeling. In *Proceedings of the 14th IEEE International Conference Requirements Engineering*, pages 129–138, 2006.

[Hah14] Frederik Hahne. A Quality Analysis Framework for Goal-oriented Requirements Specifications. Master's thesis, University of Paderborn, 2014.

[HB91] V.L. Hamilton and M.L. Beeby. Issues of Traceability in Integrating Tools. In *Tools and Techniques for Maintaining Traceability During Design, IEE Colloquium on*, pages 4–1. IET, 1991.

[HD03] Ann M. Hickey and Alan M. Davis. Elicitation technique selection: How do experts do it? In *Proceedings of the 11th IEEE International Requirements Engineering Conference*, pages 169–178. IEEE, 2003.

[HGK09] K. Hinge, A. Ghose, and G. Koliadis. Process seer: A tool for se-

mantic effect annotation of business process models. In *Enterprise Distributed Object Computing Conference, 2009. EDOC '09. IEEE International*, pages 54–63, Sept 2009.

[HIR02] Karen Henricksen, Jadwiga Indulska, and Andry Rakotonirainy. Modeling context information in pervasive computing systems. In Friedemann Mattern and Mahmoud Naghshineh, editors, *Pervasive Computing*, number 2414 in Lecture Notes in Computer Science, pages 167–180. Springer Berlin Heidelberg, January 2002.

[HMR09] Mitra Heravizadeh, Jan Mendling, and Michael Rosemann. Dimensions of Business Processes Quality (QoBP). In *Business Process Management Workshops*, number 17 in Lecture Notes in Business Information Processing, pages 80–91. Springer Berlin Heidelberg, 2009.

[HRH01] Jonathan Hammond, Rosamund Rawlings, and Anthony Hall. Will it work? In *Fifth IEEE International Symposium on Requirements Engineering*, pages 102–109. IEEE, 2001.

[Hul08] Richard Hull. Artifact-Centric Business Process Models: Brief Survey of Research Results and Challenges. In *On the Move to Meaningful Internet Systems: OTM 2008*, number 5332 in Lecture Notes in Computer Science, pages 1152–1163. Springer Berlin Heidelberg, 2008.

[IBM13] IBM. IBM Business Process Manager, 2013. http://www-03.ibm.com/software/products/de/de/business-process-manager-family.

[IEE90] IEEE. Ieee standard glossary of software engineering terminology, 1990.

[IEE98] IEEE. IEEE Recommended Practice for Software Requirements Specifications. *IEEE Std 830-1998*, pages 1–40, 1998.

[JPP93] Richard C. Johnson, David Pearson, and Keshav Pingali. Finding Regions Fast: Single Entry Single Exit and Control Regions in Linear Time. Technical report, Cornell University, 1993.

[JPP94] Richard Johnson, David Pearson, and Keshav Pingali. The Program Structure Tree: Computing Control Regions in Linear Time. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '94, pages 171–185. ACM, 1994.

[Kam05] Erik Kamsties. Understanding ambiguity in requirements engineering. In *Engineering and Managing Software Requirements*, pages 245–266. Springer, 2005.

[KBS05] Dirk Krafzig, Karl Banke, and Dirk Slama. *Enterprise SOA: Service-Oriented Architecture Best Practices*. Prentice Hall Professional, 2005.

[KEB08] Eric Knauss and Christian El Boustani. Assessing the Quality of Software Requirements Specifications. In *16th IEEE International Requirements Engineering Conference*, pages 341–342. IEEE, 2008.

[KG06] George Koliadis and Aditya Ghose. Relating Business Process Models to Goal-Oriented Requirements Models in KAOS. In *Advances in Knowledge Acquisition and Management*, volume 4303, pages 25–39. Springer Berlin Heidelberg, 2006.

[KGB06] George Koliadis, Aditya Ghose, and Moshiur Bhuiyan. Correlating Business Process and Organizational Models to Manage Change. *ACIS 2006 Proceedings*, page 28, 2006.

[KNR06] Marco Kuhrmann, Dirk Niebuhr, and Andreas Rausch. Application of the V-modell - Report from a pilot project . In *Unifying the Software Process Spectrum*, pages 463–473. Springer, 2006.

[Kol07] Dimitris Kolovos. Epsilon weblog, a brief history of epsilon, November 2007.

[KPP06] Dimitrios S. Kolovos, Richard F. Paige, and Fiona A. C. Polack. Merging Models with the Epsilon Merging Language (EML). In *Model Driven Engineering Languages and Systems*, number 4199 in Lecture Notes in Computer Science, pages 215–229. Springer Berlin Heidelberg, 2006.

[KPP08] Dimitrios S. Kolovos, Richard F. Paige, and Fiona A. C. Polack. A framework for composing modular and interoperable model management tasks. In *In Model-Driven Tool and Process Integration Workshop*, pages 79–90, 2008.

[KPR04a] R. Kazhamiakin, M. Pistore, and M. Roveri. A Framework for Integrating Business Processes and Business Requirements. In *Proceedings of the Eighth IEEE International Enterprise Distributed Object Computing Conference*, pages 9– 20. IEEE, 2004.

[KPR04b] R. Kazhamiakin, M. Pistore, and M. Roveri. A framework for integrating business processes and business requirements. In *Proceedings of the Eighth IEEE International Enterprise Distributed Object Computing Conference, 2004*, pages 9– 20, 2004.

[KRG07] Jochen M. Küster, Ksenia Ryndina, and Harald Gall. Generation of Business Process Models for Object Life Cycle Compliance. In *Business Process Management*, number 4714 in Lecture Notes in Computer Science, pages 165–181. Springer Berlin Heidelberg, 2007.

[KRGDP14] Dimitris Kolovos, Louis Rose, Antonio Garcia-Dominguez, and Richard Paige. *The epsilon Book*. eclipse.org, feb 2014.

[Kru04] Philippe Kruchten. *The rational unified process: an introduction*. Addison-Wesley Professional, 2004.

[KVB+06a] George Koliadis, Aleksandar Vranesevic, Moshiur Bhuiyan, Aneesh Krishna, and Aditya Ghose. Combining i* and BPMN for business process model lifecycle management. In *Business Process Management Workshops*, pages 416–427. Springer, 2006.

[KVB+06b] George Koliadis, Aleksander Vranesevic, Moshiur Bhuiyan, Aneesh Krishna, and Aditya K. Ghose. A combined approach for supporting the business process model lifecycle. In *Proceeding of the Tenth Pacific Asia Conference on Information Systems*, pages 1305–1319, 2006.

[KW09] Hermann Kaindl and Patrick Wagner. A Unification of the Essence of Goal-oriented Requirements Engineering. In *Fourth Interna-*

*tional Conference on Software Engineering Advances*, pages 45–50. IEEE, 2009.

[Law97] Brian Lawrence. Requirements Happens. *American Programmer*, 10:3–9, 1997.

[Les86] Michael Lesk. Automatic sense disambiguation using machine readable dictionaries: how to tell a pine cone from an ice cream cone. In *Conference on Systems documentation (1986)*, pages 24–26. ACM, 1986.

[Let02] Patricio Letelier. A Framework for Requirements Traceability in UML-based Projects. In *Proceedings of the 1st International Workshop on Traceability in Emerging Forms of Software Engineering*, pages 30–41, 2002.

[Lin98] Dekang Lin. An information-theoretic definition of similarity. In *MLDM (1998)*, ICML '98, pages 296–304. Morgan Kaufmann Publishers, 1998.

[LLJM11] Sotirios Liaskos, Marin Litoiu, Marina Daoud Jungblut, and John Mylopoulos. Goal-Based Behavioral Customization of Information Systems. In *Advanced Information Systems Engineering*, volume 6741, pages 77–92. Springer Berlin Heidelberg, 2011.

[LMSM10] S. Liaskos, S.A. McIlraith, S. Sohrabi, and J. Mylopoulos. Integrating Preferences into Goal Models for Requirements Engineering. In *RE'10*, pages 135 –144, 2010.

[LS95] François Laroussinie and Ph. Schnoebelen. A hierarchy of temporal logics with past. *Theoretical Computer Science*, 148(2):303–324, 1995.

[LS96] Mikael Lindvall and Kristian Sandahl. Practical Implications of Traceability. *Software Practice and Experience*, 26(10):1161–1180, 1996.

[LSM12] Henrik Leopold, Sergey Smirnov, and Jan Mendling. On the refactoring of activity labels in business process models. *Information Systems*, 37(5):443–459, 2012.

[LW00] Dean Leffingwell and Don Widrig. *Managing Software Requirements: A Unified Approach*. Addison-Wesley Professional, 2000.

[LWE01] Brian Lawrence, Karl Wiegers, and Christof Ebert. The top risk of requirements engineering. *Software, IEEE*, 18(6):62–63, 2001.

[LY04] Lin Liu and Eric Yu. Designing information systems in social context: a goal and scenario modelling approach. *Information Systems*, 29(2):187–203, 2004.

[LYM07] Alexei Lapouchnian, Yijun Yu, and John Mylopoulos. Requirements-driven Design and Configuration Management of Business Processes. In *Business Process Management*, pages 246–261. Springer, 2007.

[Mai13] Neil Maiden. Requirements engineering as information search and idea discovery. In *Proceeding of the 21st IEEE International Requirements Engineering Conference (RE)*, pages 1–1, 2013.

[ME98] George M. Marakas and Joyce J. Elam. Semantic Structuring in Analyst Acquisition and Representation of Facts in Requirements Analysis. *Information Systems Research*, 9(1):37–63, 1998.

[MF98] G. Miller and Ch. Fellbaum. *WordNet: An Electronic Lexical Database*. MIT Press, Cambridge, MA, 1998.

[Mil95] G. A. Miller. WordNet: a Lexical Database for English. *Communications of the ACM*, 38(11):39–41, 1995.

[MKK05] Subhas Misra, Vinod Kumar, and Uma Kumar. Goal-oriented or scenario-based requirements engineering technique - What should a practitioner select? In *Canadian Conference on Electrical and Computer Engineering*, pages 2288–2292. IEEE, 2005.

[MM+03] Joaquin Miller, Jishnu Mukerji, et al. MDA Guide Version 1.0.1. *Object Management Group*, 234:51, 2003.

[MNP05] Oded Maler, Dejan Nickovic, and Amir Pnueli. Real time temporal logic: Past, present, future. In *Formal Modeling and Analysis of Timed Systems*, pages 2–16. Springer, 2005.

[MP84] Stephen M McMenamin and John F Palmer. *Essential Systems Analysis*. Yourdon Press, 1984.

[MPMG08] Alicia Martínez, Oscar Pastor, John Mylopoulos, and Paolo Giorgini. From Early to Late Requirements: A Goal-Based Approach. In *Agent-Oriented Information Systems IV*, pages 123–142. 2008.

[MR96] N. A. M. Maiden and G. Rugg. ACRE: selecting methods for requirements acquisition. *Software Engineering Journal*, 11(3):183–192, 1996.

[NGE14] Benjamin Nagel, Christian Gerth, and Gregor Engels. Goal-Driven Composition of Business Process Models. In *International Conference on Service-Oriented Computing ICSOC - Workshops*, number 8377 in Lecture Notes in Computer Science, pages 16–27. Springer, 2014.

[NGPE13a] Benjamin Nagel, Christian Gerth, Jennifer Post, and Gregor Engels. Ensuring Consistency among Business Goals and Business Process Models. In *Proceedings of 16th IEEE International Enterprise Distributed Object Computing Conference (EDOC)*, pages 17–26, 2013.

[NGPE13b] Benjamin Nagel, Christian Gerth, Jennifer Post, and Gregor Engels. Kaos4SOA - Extending KAOS Models with Temporal and Logical Dependencies. In *Proceedings of the CAiSE'13 Forum at the 25th International Conference on Advanced Information Systems Engineering (CAiSE)*, pages 9–16, 2013.

[NOH11] Hiroyuki Nakagawa, Akihiko Ohsuga, and Shinichi Honiden. gocc: a configuration compiler for self-adaptive systems using goal-oriented requirements description. In *SEAMS'11*, pages 40–49. ACM, 2011.

[NR02] Markus Nüttgens and Frank J. Rump. Syntax und Semantik Ereignisgesteuerter Prozessketten (EPK). In *Promise*, volume 2, pages 64–77, 2002.

[NSB+15] Benjamin Nagel, Klaus Schröder, Steffen Becker, Stefan Sauer, and

Gregor Engels. Kooperative Methoden- und Werkzeugentwicklung zur Cloudmigration von proprietären Anwendungskomponenten. In *Software Engineering 2015, Fachtagung des GI-Fachbereichs Softwaretechnik*, 2015. (accepted for publication).

[Nus01] Bashar Nuseibeh. Weaving together requirements and architectures. *Computer*, 34(3):115–119, 2001.

[OAS11] OASIS. Web Services Business Process Execution Language Version 2.0, 2011.

[OAS+12] Zachary J. Oster, Syed Adeel Ali, Ganesh Ram Santhanam, Samik Basu, and Partha S. Roop. A Service Composition Framework Based on Goal-Oriented Requirements Engineering, Model Checking, and Qualitative Preference Analysis. In *Service-Oriented Computing*, number 7636 in Lecture Notes in Computer Science, pages 283–297. Springer Berlin Heidelberg, 2012.

[OMG09] OMG. Model Driven Architecture (MDA), 2009.

[OMG11a] OMG. Business Process Model and Notation (BPMN), 2011.

[OMG11b] OMG. OMG Unified Modeling Language (OMG UML), Superstructure, Version 2.4.1, 2011.

[OP10] Alexander Osterwalder and Yves Pigneur. *Business Model Generation: A Handbook For Visionaries, Game Changers, And Challengers*. Wiley, 2010.

[OSB11a] Zachary J. Oster, Ganesh Ram Santhanam, and Samik Basu. Automating Analysis of Qualitative Preferences in Goal-Oriented Requirements Engineering. In *Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering*, pages 448–451, 2011.

[OSB11b] Z.J. Oster, G.R. Santhanam, and S. Basu. Identifying Optimal Composite Services by Decomposing the Service Composition Problem. In *Proceedings of the 9th International Conference on Web Services*, pages 267 –274, 2011.

[Pau93] Mark Paulk. *Capability maturity model for software*. Wiley Online Library, 1993.

[PG96] F.A.C. Pinheiro and Joseph A. Goguen. An Object-oriented Tool for Tracing Requirements. *IEEE Software*, 13(2):52–64, 1996.

[PLM13] Fabian Pittke, Henrik Leopold, and Jan Mendling. Spotting Terminology Deficiencies in Process Model Repositories. In *Enterprise, Business-Process and Information Systems Modeling*, pages 292–307. Springer, 2013.

[PNEM14] Fabian Pittke, Benjamin Nagel, Gregor Engels, and Jan Mendling. Linguistic Consistency of Goal Models. In *Enterprise, Business-Process and Information Systems Modeling*, pages 393–407. Springer, 2014.

[Poh96] Klaus Pohl. *Process-Centered Requirements Engineering*. John Wiley & Sons, Inc., 1996.

[Poh10] Klaus Pohl. *Requirements Engineering: Fundamentals, Principles, and Techniques*. Springer, 2010.

[Pos12] Jennifer Post. Goal-driven Refinement of Quality Constraints for Adaptive Business Processes. Master's thesis, University of Paderborn, 2012.

[PTDL03] Michael P. Papazoglou, Paolo Traverso, Schahram Dustdar, and Frank Leymann. Service-oriented computing. *Communications of the ACM*, 46:25–28, 2003.

[Ram98] Balasubramaniam Ramesh. Factors Influencing Requirements Traceability Practice. *Communications of the ACM*, 41(12):37–44, 1998.

[Res95] Philip Resnik. Using information content to evaluate semantic similarity in a taxonomy. In *IJCAI'95*, pages 448–453, 1995.

[Res07] Respect IT. A KAOS Tutorial, Objectiver, 2007.

[RJ01] B. Ramesh and M. Jarke. Toward reference models for requirements traceability. *IEEE Transactions on Software Engineering*,

27(1):58–93, 2001.

[RR06] S. Robertson and J.C. Robertson. *Mastering the Requirements Process*. Pearson Education, 2006.

[RSA98] C. Rolland, C. Souveyet, and C.B. Achour. Guiding Goal Modeling Using Scenarios. *IEEE Transactions on Software Engineering*, 24(12):1055–1071, 1998.

[SAP13] SAP. SAP NetWeaver Business Process Management, 2013.

[Sch97] Ken Schwaber. Scrum development process. In *Business Object Design and Implementation*, pages 117–134. Springer, 1997.

[Sch01] Bruno Schienmann. *Kontinuierliches Anforderungsmanagement: Prozesse-Techniken-Werkzeuge*. Pearson Deutschland GmbH, 2001.

[SGP10] Pnina Soffer, Johny Ghattas, and Mor Peleg. A Goal-Based Approach for Learning in Business Processes. In *Intentional Perspectives on Information Systems Engineering*, pages 239–256. Springer Berlin Heidelberg, 2010.

[Sin09] Roopak Sinha. *Automated techniques for formal verification of SoCs*. PhD thesis, ResearchSpace@ Auckland, 2009.

[SK98] Ian Sommerville and Gerald Kotonya. *Requirements Engineering: Processes and Techniques*. John Wiley & Sons, Inc., 1998.

[SO01] Norbert Schwarz and Daphna Oyserman. Asking questions about behavior: Cognition, communication, and questionnaire construction. *American Journal of Evaluation*, 22(2):127–160, 2001.

[Som07] I. Sommerville. *Software Engineering*. International computer science series. Addison-Wesley, 2007.

[Spe12] The Workflow Management Coalition Specification. Process Definition Interface – XML Process Definition Language 2.2. *Document Number WFMC-TC-1025*, 1, 2012.

[SRS+93] I. Sommerville, T. Rodden, P. Sawyer, R. Bentley, and M. Twidale. Integrating ethnography into the requirements engineering pro-

cess. In *Proceedings of IEEE International Symposium on Requirements Engineering, 1993*, pages 165–173, 1993.

[SS97] Ian Sommerville and Pete Sawyer. *Requirements engineering: a good practice guide*. John Wiley & Sons, Inc., 1997.

[SS02] Michael Schrefl and Markus Stumptner. Behavior-Consistent Specialization of Object Life Cycles. *ACM Transactions on Software Engineering Methodology*, 11(1):92–148, 2002.

[Sut98] Alistair Sutcliffe. Scenario-Based Requirements Analysis. *Requirements engineering*, 3(1):48–65, 1998.

[SW04] Pnina Soffer and Yair Wand. Goal-driven analysis of process model validity. In *Advanced Information Systems Engineering*, pages 521–535. Springer, 2004.

[SW07] Pnina Soffer and Yair Wand. Goal-Driven Multi-Process Analysis. *Journal of the Association for Information Systems*, 8(3), 2007.

[SZ05] George Spanoudakis and Andrea Zisman. Software traceability: a roadmap. *Handbook of Software Engineering and Knowledge Engineering*, 3:395–428, 2005.

[Szy02] Clemens Szyperski. *Component software: beyond object-oriented programming*. Pearson Education, 2002.

[TBBG07] Maurice Ter Beek, Antonio Bucchiarone, and Stefania Gnesi. Web Service Composition Approaches: From Industrial Standards to Formal Methods. In *Proceedings of the Second International Conference on Internet and Web Applications and Services*, page 15. IEEE, 2007.

[The06] The Standish Group. The standish group report chaos. Technical report, 2006.

[Tjo08] Sri Fatimah Tjong. *Avoiding ambiguity in requirements specifications*. PhD thesis, University of Waterloo, 2008.

[TP04] Paolo Traverso and Marco Pistore. Automated Composition of Semantic Web Services into Executable Processes. In *The Semantic*

*Web - ISWC 2004*, pages 380–394. Springer, 2004.

[VCM+07] Irene Vanderfeesten, Jorge Cardoso, Jan Mendling, Hajo A. Reijers, and Wil van der Aalst. Quality Metrics for Business Process Models. *BPM and Workflow handbook*, pages 179–190, 2007.

[VdATH05] Wil M.P. Van der Aalst and Arthur H.M. Ter Hofstede. YAWL: Yet Another Workflow Language. *Information systems*, 30(4):245–275, 2005.

[VKP02] Antje Von Knethen and Barbara Paech. A survey on tracing approaches in practice and research. *Frauenhofer Institut Experimentelles Software Engineering, IESE-Report No*, 95, 2002.

[vKPKH02] Antje von Knethen, Barbara Paech, Friedemann Kiedaisch, and Frank Houdek. Systematic requirements recycling through abstraction and traceability. In *IEEE Joint International Conference on Requirements Engineering*, pages 273–281. IEEE, 2002.

[vL01] A. van Lamsweerde. Goal-Oriented Requirements Engineering: A Guided Tour. In *Proceedings of the Fifth IEEE International Symposium on Requirements Engineering*, pages 249 –262, 2001.

[vL03] Axel van Lamsweerde. From system goals to software architecture. In *Third Int. School on Formal Methods for the Design of Computer, Communication and Software Systems: Software Architectures*, pages 25–43. Springer Berlin Heidelberg, 2003.

[VL04] A. Van Lamsweerde. Goal-oriented Requirements Engineering: A Roundtrip from Research to Practice. In *Proceedings of the 12th IEEE International Requirements Engineering Conference*, pages 4–7, 2004.

[VL09] Axel Van Lamsweerde. *Requirements Engineering: From System Goals to UML Models to Software Specifications*. John Wiley, 2009.

[vLDL98] A. van Lamsweerde, R. Darimont, and E. Letier. Managing Conflicts in Goal-Driven Requirements Engineering. *IEEE Transactions on Software Engineering*, 24(11):908–926, 1998.

[vLL00] A. van Lamsweerde and E. Letier. Handling Obstacles in Goal-Oriented Requirements Engineering. *IEEE Transactions on Software Engineering*, 26(10):978–1005, 2000.

[VVL07] Jussi Vanhatalo, Hagen Völzer, and Frank Leymann. Faster and More Focused Control-Flow Analysis for Business Process Models Through SESE Decomposition. In *Proceedings of the Fifth International Conference on Service-Oriented Computing*, pages 43–55. Springer Berlin Heidelberg, 2007.

[Wes99] Mathias Weske. Business-Objekte: Konzepte, Architekturen, Standards. *Wirtschaftsinformatik*, 41(1):4–11, 1999.

[Wes12] Mathias Weske. *Business Process Management: Concepts, Languages, Architectures*. Springer, 2012.

[Wie03] Karl Eugene Wiegers. *Software Requirements*. Microsoft press, 2003.

[Wil97] William Wilson. Writing Effective Requirements Specifications. In *Software Technology Conference*, 1997.

[WM06] Dan Woods and Thomas Mattern. *Enterprise SOA: Designing IT for Business Innovation*. O'Reilly Media, Inc., 2006.

[WP94] Zhibiao Wu and Martha Palmer. Verbs semantics and lexical selection. In *ACL'94*, pages 133–138. Association for Computational Linguistics, 1994.

[WP10] Stefan Winkler and Jens von Pilgrim. A survey of traceability in requirements engineering and model-driven development. *Software & Systems Modeling*, 9(4):529–565, 2010.

[WRH+12] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation in Software Engineering*. Springer, 2012.

[YLL+08] Yijun Yu, Alexei Lapouchnian, Sotirios Liaskos, John Mylopoulos, and Julio Leite. From Goals to High-Variability Software Design. In *Foundations of Intelligent Systems*, pages 1–16. 2008.

[Yu93] E.S.K. Yu. Modeling organizations for information systems requirements engineering. In *Proceedings of the IEEE International Symposium on Requirements Engineering*, pages 34–41, 1993.

[Yu96] Eric Siu-Kwong Yu. *Modelling strategic relationships for process reengineering*. PhD thesis, University of Toronto, 1996.

[Yu97] E.S.K. Yu. Towards modelling and reasoning support for early-phase requirements engineering. In *Proceedings of the Third IEEE International Symposium on Requirements Engineering*, pages 226–235, 1997.

[Yue87] Kaizhi Yue. What does it mean to say that a specification is complete? In *Fourth International Workshop on Software Specification and Design*, 1987.

[ZBP+12] Farhana H. Zulkernine, Piergiorgio Bertoli, Marco Pistore, Andreas Friesen, Jens Lemcke, Bernhard Thimmel, and Otfried Von Geisau. A constraint-driven business object model for service-based business processes. In *Third International Conference on Information Technology: New Generations*, pages 182–188. IEEE Computer Society, 2012.

[ZC05] Didar Zowghi and Chad Coulin. Requirements Elicitation: A Survey of Techniques, Approaches, and Tools. In *Engineering and Managing Software Requirements*, pages 19–46. Springer Berlin Heidelberg, 2005.

[ZCPT05] Marielba Zacarias, Artur Caetano, H. Sofia Pinto, and Jose Tribolet. Modeling Contexts for Business Process Oriented Knowledge Support. In *Professional Knowledge Management*, number 3782 in Lecture Notes in Computer Science, pages 431–442. Springer Berlin Heidelberg, 2005.

[Zmu80] Robert W. Zmud. Management of large software development efforts. *MIS Quarterly*, pages 45–55, 1980.

# Appendix A

# Formal XText Grammar for CTL Editor

In this appendix, we provide additional information about the grammar that we used to generate the CTL editor. The grammar is defined in the XText syntax [1] and has been used to generate the corresponding textual editor in an automated manner.

```
grammar org.xtext.CTL with org.eclipse.xtext.common.Terminals

generate cTL "http://www.xtext.org/CTL"

CTLFormulae:
  formulae+=CTLFormula*;

CTLFormula:
  ID
  | '(' CTLFormula
  | '!' CTLFormula
  | ')' CTLFormula
  | BinLogOperator CTLFormula
  | TemporalOperator CTLFormula
  | 'E' CTLFormula '[' CTLFormula
  | UntilLogOperator CTLFormula
  | ']' CTLFormula
  | '[' CTLFormula
  | 'A' CTLFormula
```

[1] https://eclipse.org/Xtext/

```
    | UNCLOSED_STRING
;

TemporalOperator:
    'AX' | 'EX' | 'AF' | 'EF' | 'AG' | 'EG'
;

UntilLogOperator:
    'U'
;

BinLogOperator:
    AND|OR|IMPL|BICOND
;

AND:
    '&'
;

OR:
    '|'
;

IMPL:
    '->'
;

BICOND:
    '<->'
;

terminal UNCLOSED_STRING : ';' (!';')* EOF;
```

# Appendix B

# Dependencies and Constraints of Case Study

In this appendix, we give an overview about further details about the results for the case study discussed in Section 7.2. We describe the defined temporal dependencies between business goals in Section B.1 and the formalized and refined CTL constraints in Section B.2.

## B.1 Temporal Dependencies between Goals

The case study in Section 7.2 deals with the analyis of the applicability and expressability of our approach for the definition of temporal dependencies among goals and their automated formalization and refinement. For this purpose, we defined a set of dependencies that need to be considered in the business process composition. An overview about the specified dependencies is provided in Table B.1.

| **Depender** | **Temporal dependency** | **Dependee** |
|---|---|---|
| ControlRoomLocked | hasPredecessor | ControlRoomLockable |
| ElevatorEquippedWithFloorDoors | hasSuccessor | DoorClosedWhenCageNotStopped OnAFloorLevel |
| ElevatorEquippedWithFloorDoors | hasSuccessor | NoDoorOpeningWhileMoving |
| LightsAlwaysOn | hasPredecessor | LightsInsideCage |
| ElevatorCageHasADoor | hasSuccessor | CageDoorClosedWhileMoving |
| ElevatorStoppedUponPowerFailure | hasPredecessor | EmergencyStopAvailable |
| StopButtonUsed | hasPredecessor | CageButtonPanelIncludesAStopButton |
| StopButtonUsed | hasSuccessor | ElevatorStopped |
| ElevatorStoppedUponPowerFailure | hasPredecessor | EmergencyStopAvailable |
| ElevatorStoppedUponPowerFailure | hasSuccessor | EmergencyPowerAvailable |
| EmergencyLightsOnWhenNeeded | hasPredecessor | EmergencyPowerAvailable |
| EmergencyLightsOnWhenNeeded | hasPredecessor | CageEquippedWithEmergencyLights |
| OverweightConditionsReported ToThePassenger | hasPredecessor | WeightConditionsChecked BeforeNextMove |
| ElevatorKeptOnCurrentFloorDoors OpenUntilOverweightConditionsDisappear | hasPredecessor | WeightConditionsChecked BeforeNextMove |

Table B.1: Specified Temporal Dependencies in Goal Model

## B.2 Formalized and Refined CTL Constraints

As an input for our constraint generation approach we used the temporal dependencies described in Section B.1. The result is a set of refined and formalized CTL constraints that can be verified in an automated manner. Therefore, they can be considered in the composition of a business process model.

```
AG (ElevatorEquippedWithFloorDoors -> AF(
    DoorClosedWhenCageNotStoppedOnAFloorLevel))
AG (ElevatorEquippedWithFloorDoors -> AF(NoDoorOpeningWhileMoving
    ))
A[!((!CageButtonPanelIncludesAStopButton) U StopButtonUsed)]
AG (StopButtonUsed -> AF(ElevatorStopped))
AG (ElevatorCageHasADoor -> AF(CageDoorClosedWhileMoving))
A[!((!LightsInsideCage) U LightsAlwaysOn)]
A[!((!(MovingElevatorStoppedNextFloorInCaseOfFireSignal & (
    ElevatorStopped & StopButtonUsed &
    CageButtonPanelIncludesAStopButton) &
    EmergencyStopAvailableInControlRoom)) U
    ElevatorStoppedUponPowerFailure)]
AG (ElevatorStoppedUponPowerFailure -> AF(EmergencyPowerAvailable
    ))
A[!((!EmergencyPowerAvailable) U EmergencyLightsOnWhenNeeded)]
A[!((!CageEquippedWithEmergencyLights) U
    EmergencyLightsOnWhenNeeded)]
A[!((!WeightConditionsCheckedBeforeNextMove) U
    OverweightConditionsReportedToThePassengers)]
A[!((!WeightConditionsCheckedBeforeNextMove) U
    ElevatorKeptOnCurrentFloorDoorsOpenUntilOverweight
    ConditionsDisappear)]
A[!((!ControlRoomLockable) U ControlRoomLocked)]
```

# Appendix C

# Business Process Model for Traceability Analysis Experiment

In Section 7.4, we described a traceability analysis experiment. Hereby, completeness and relevance between a business goal model and a related business process model have been validated. The applied business process model is depicted in Figure C.1. The relations between the fragments and the related business goals are indicated by the names of the fragments.

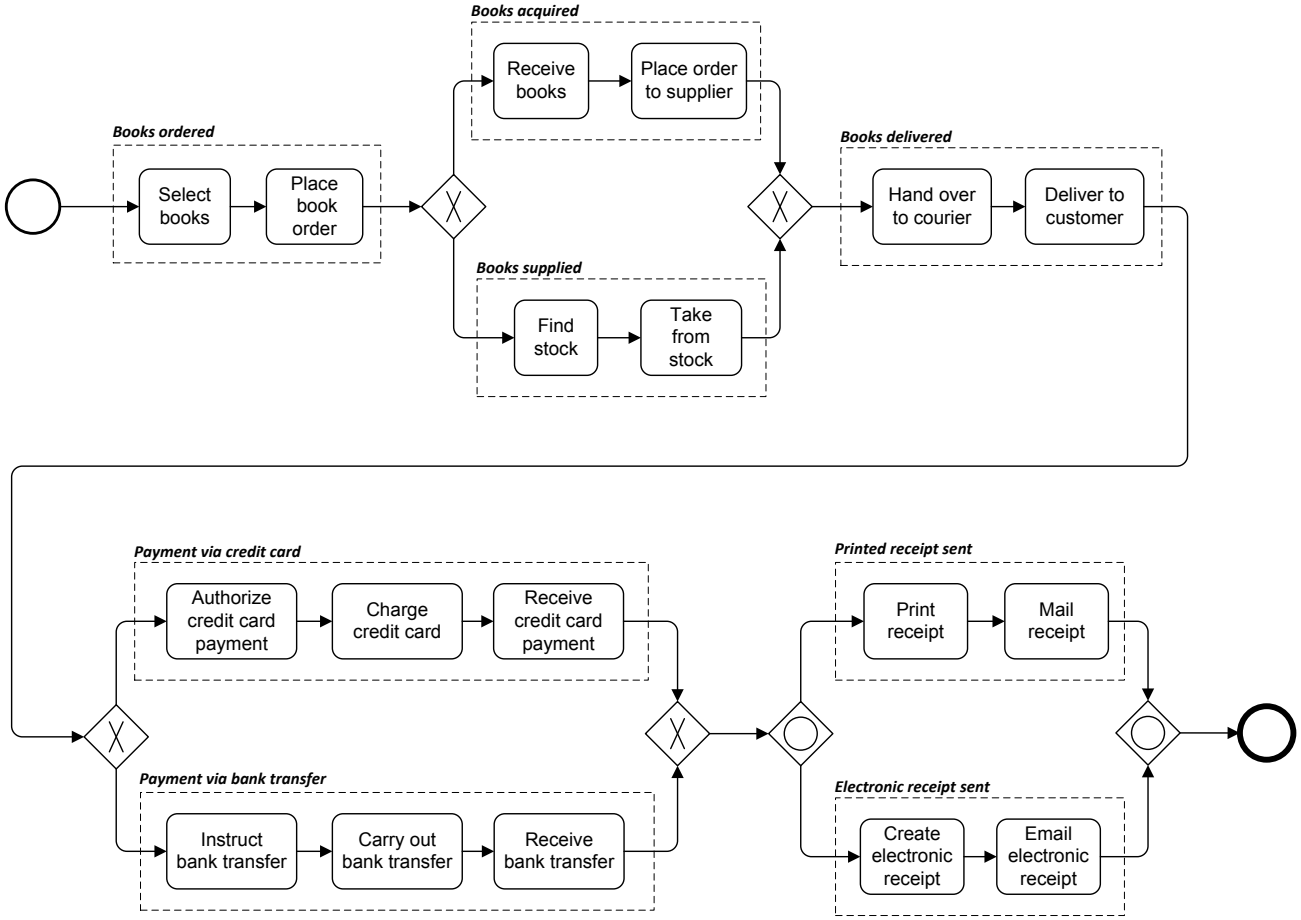


Figure C.1: Business Process Model for *Fulfill Book Order* Case Study

# Appendix D

# Goal Models for Linguistic Consistency Analysis

The linguistic consistency analysis discussed in Section 7.3 is based on three different goal models taken from literature. The applied goal models are depicted in Figures D.1 - D.3.

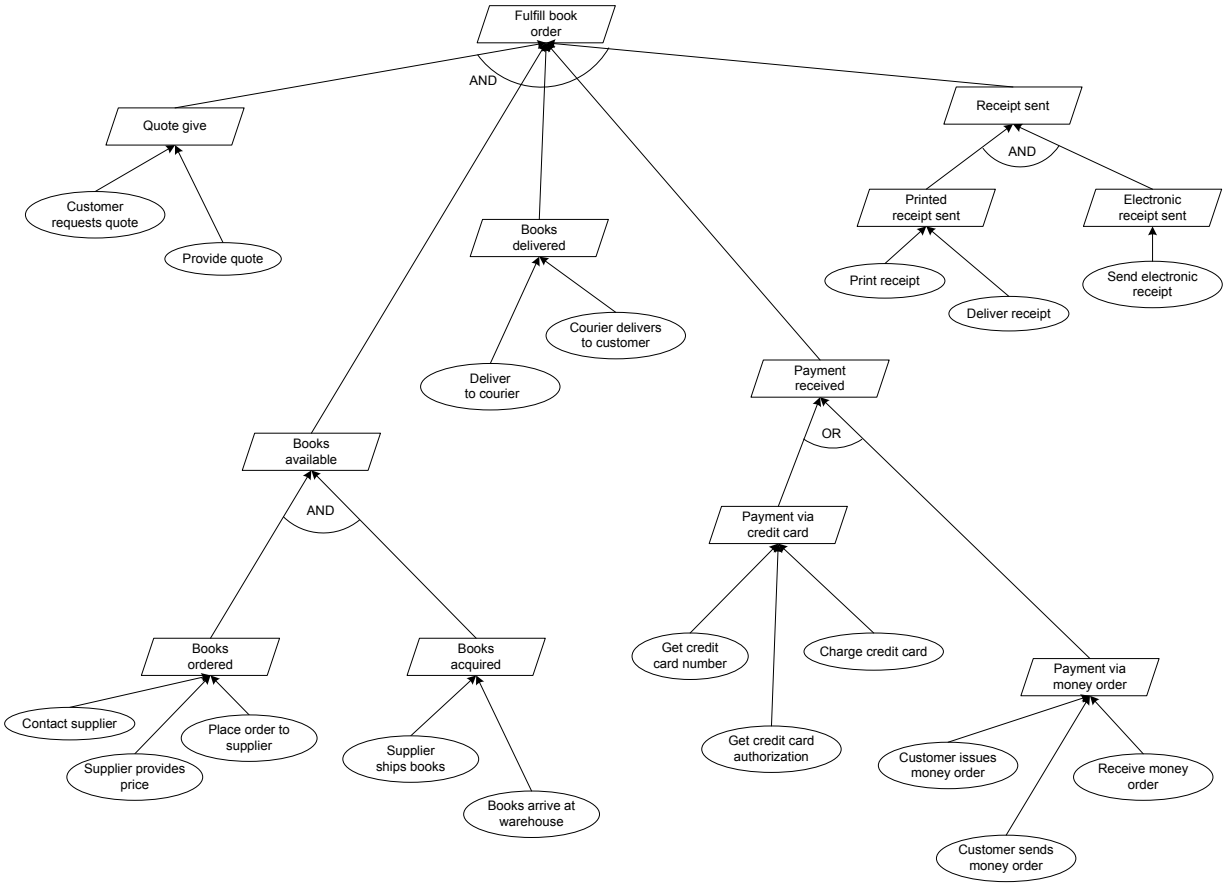


Figure D.1: Goal Model for *Fulfill Book Order* Case Study [LMSM10]

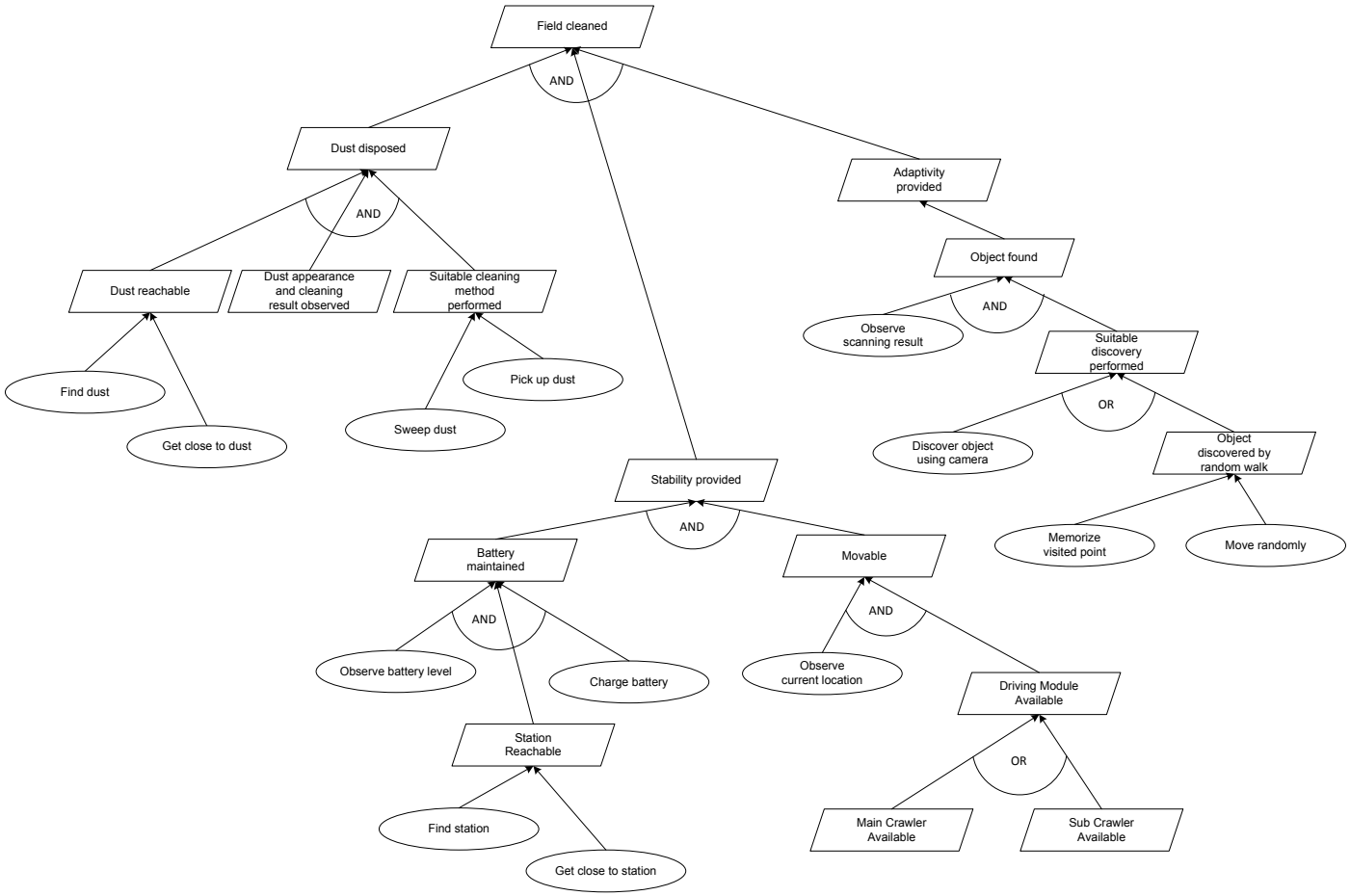


Figure D.2: Goal Model for *Cleaning Robot* Case Study [NOH11]

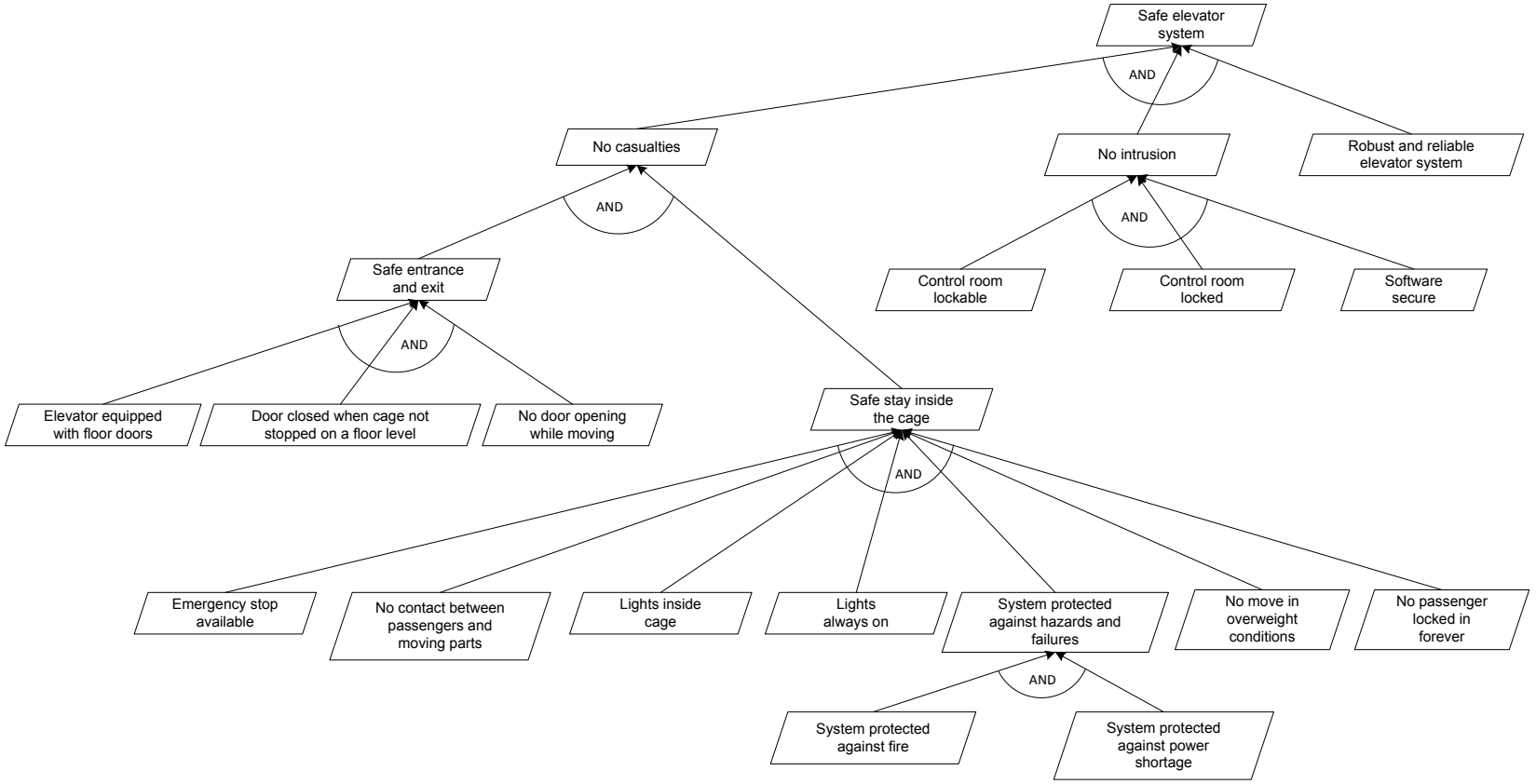


Figure D.3: Goal Model for *Safe Elevator System* Case Study [Res07]