Developing and Validating Customizable Process Models

by

Mohsen Asadi

M.Sc., Sharif University of Technology, 2009

B.Sc. (Hons.), Islamic Azad University of Mashhad, 2006

A Dissertation submitted in partial fulfillment
of the requirements for the degree of

Doctor of Philosophy

in the
School of Interactive Arts and Technology
Faculty of Communication, Art and Technology

SIMON FRASER UNIVERSITY
Fall 2014

APPROVAL

Name: Mohsen Asadi

Degree: Doctor of Philosophy

Title of Dissertation: Developing and Validating Customizable Process Models

Examining Committee:

Dr. Lyn Bartram
Associate Professor, SIAT
Chair

Dr. Marek Hatala, Senior Supervisor
Professor and Director, SIAT

Dr. Dragan Gašević, Co-Supervisor
Professor, CRC Semantic and Learning Technologies,
Computing and Information Systems,
Athabasca University

Dr. Uwe Gläesser, Supervisor
Professor and Associate Dean,
Computing Science, SFU

Dr. Halil Erhan, Internal Examiner
Associate Professor, SIAT

Dr. Krzysztof Czarnecki, External Examiner
Professor, Electrical and Computer Engineering
Waterloo University

Date Approved: September 2nd, 2014

Partial Copyright Licence

Abstract

A process model defines the activities of a business process and their attributes (e.g. cost and time). Process models typically are instantiated several times and every process instance may be executed differently. Hence, several variants of the same process model may coexist in organizations which urges the organizations to support flexible processes. Motivated by the need of flexible process models, the notion of customizable process models has been proposed which integrate variability in process models.The development and adaptation of customizable process models raise several challenges: 1) the need for taking into account several variability types; 2) integrating variability into process models impose additional modeling complexity; 3) deriving a process variant from a customizable process model requires the close consideration of a target application requirements; 4) ensuring compliance of a process variant with behavioral and configuration constraints formulated in a customizable process model.

This dissertation presents a feature oriented customization and validation framework for customizable process models. The customization component relies on software product lines and utilizes feature modeling techniques for modeling variability in customizable process models. Additionally, a pre-configuration process, a decision making technique called Stratified Analytical Hierarchy Process (S-AHP), and Artificial Intelligent planning techniques are provided to derive a process variant from a customizable process model based on the stakeholders requirements. The validation component identifies a set possible inconsistency patterns between requirements model (represented by goal model), variability model (represent by feature model), and customizable process models and employs Description Logic to detect the inconsistencies.

We evaluated the framework using a set of experiments and a comparative analysis. The results show the efficient running time of the proposed techniques and improvements over state of the art in customizable process models.

To my loving wife and my parents.

"Two there are who are never satisfied – the lover of the world and the lover of knowledge"

Rumi

Acknowledgments

First and foremost, I would like to express my sincere gratitude to my supervisory team at the Simon Fraser University for supporting me throughout my studies. I am grateful to my senior supervisor Prof. Marek Hatala for providing invaluable feedback in different stages of my research and for granting me the opportunity to obtain a Ph.D. I would like to thank my co-supervisor Prof. Dragan Gašević for many hours he has spent with me, exploring ideas, examining the details, challenging my thoughts, and improving my skills. His gentleness, earnestness, and positiveness have always impressed me, and will continue to inspire me in my future career. My supervisor, Prof. Uwe Glässer, my internal examiner, Prof. Halil Erhan, and my external examiner Prof. Krzysztof Czarnecki are thanked for accepting to review this work and for their useful and constructive comments and feedback.

My deepest and sincere thanks go the my best colleague, best friend, and my beloved wife, Samaneh, for her love, encouragement, trust, understanding, endurance and admirable support over these years, without which I would not have made it.

Thank you to my colleagues in the Ontological Research Laboratory and my co-authors. This work was shaped through many discussions and collaborations with my colleagues and co-authors. Especially I would like to thank the core group of people, Dr. Ebrahim Bagheri, Dr. Bardia Mohabbati, Dr. Marko Bošković, Dr. Gerd Gröner, and Prof. Yair Wand. In particular, I would like to thank Dr. Bardia Mohabbati and Dr. Ebrahim Bagheri for their friendship and support.

I also want to thank my friends, most notably, Dr. Melody Siadaty, Liaqat Ali, Vitomir Kovanovic, Srecko Joksimovic, Sanam Shirazi, Dr. Mohsen Jamali, Dr. Samaneh Abbasi Moghaddam, Dr. Amin Milani Fard, and Dr. Hoda Akbari.

I also express my heartfelt gratitude to my parents and my sisters and brothers who constantly support and encourage me in pursuing my goals.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

A *business process* is defined as a set of activities which are logically and temporally ordered to realize a particular *business goal* [130, 44]. Business process models[1] are commonly used in all business domains to describe what companies do to deliver a service or a product to customers. Process models enable a process-centered alignment of organizations' information systems and IT infrastructure which improves their agility and performance and decreases their time to market in delivering services or products. The growing importance of managing processes leads to emergence of a new generation of information systems called *process aware information systems (PAISs)* [130, 45]. In this chapter, we review the notion of process-aware information systems, their development methodology, efforts in boosting flexibility in PAISs, and issue related to PAIS. Next, we explain the research challenges, method, and contributions.

1.1 Research Context

According to [45] a Process Aware Information System(PAIS) is *"a software system that manages and executes operational processes involving people, application, and/or information sources on the basis of process models."* The core notion in PAIS is the process model which describes the activities of a business process and their attributes (e.g. cost and time) as well as control and data flow between them. The process models are usually represented using a visual language such as Busniness Process Management Notation (BPMN) and Event Process Chain (EPC). Employing process models provides

[1] In this thesis we refer to them as Process Models

several advantages: The communication between business managers and process analyst who are concerned with the structure of process model, and software developers who design and implement processes is facilitated; and changes in the process levels (e.g. adding new activities) are abstracted from application services implementing processes and vice verse [130].

Process models typically are instantiated several times and every process instance may differently be executed based on the context and the requirements of target stakeholders [45]. Hence, several variants of a process model may coexist in an organization which urges organizations to support flexible processes to cope with process variabilities [136]. Motivated by the need of flexible process models, a number of approaches have been proposed for the development and customization of *reference process models* [136, 65, 83]. In this section, we briefly describe a lifecycle for developing process aware information systems and reference process models.

1.1.1 PAIS Development Methodology

Several efforts have been done in business process management domain to define a lifecycle for developing process-aware information systems [44, 45, 130]. A common development lifecycle encompasses the following phases:

- **Requirements and process analysis -** This phase aims to capture the requirements and business objectives of organizations. The requirements are explored and refined to precisely describe the problem. Also, the existing processes that are relevant to the problem are identified, their scope are delimited, and their issues are determined. At the end of this phase, a requirements specification and a collection of current processes related to the requirements are documented.

- **Process (re)design -** This phase aims to design and re-desing the processes and represent them using a process modeling language such as BPMN or EPC. Hence, the required changes to improve the existing processes are identified and new processes are built. The process models which are developed in this phase usually represent a rather high-level description of a process.

- **Process implementation -** This phase aims at refining the process models into operational models supported by software systems. To this end, technical (i.e. software and hardware) and organizational information about the environment are

taken into account. Afterward, the process models are tested and deployed into the target environment.

- **Process monitoring and improvement -** In this phase the new processes are executed. The executions of processes are monitored and relevant data is collected and analyzed for further improvements.

1.1.2 Reference and Customizable Process Models

Reference process models capture proven practices and recurrent business operations in a specific domain [166, 132, 131]. Reference process models are designed in a generic way and intended to be configured and customized to meet requirements of individual stakeholders. However, reference process models often lack an explicit representation of configuration options and alternatives [166, 132]. The explicit representation of the variability is one of the key aspects of variability and configuration management. Research agrees that the variability should be modeled and represented in a uniform way in the development life-cycle [170, 39, 150, 26]. Therefore, many researchers have been working on the concept of *customizable(configurable) process models* in the context of process-oriented software development [55, 137, 106, 58].

A Customizable reference process model represents a family of similar process models in an integrated way [83, 56].[2] Several approaches and methods have been proposed to model variability in configurable reference process models [83, 56, 55, 132, 137], in which process modeling languages like EPC, YAWL, and BPMN are extended with some notations for supporting variability. The development methodology in these approaches differs from the development methodology of a single process-aware information system in which two lifecycles *customizable process development* and *PAIS development* are defined.

The customizable process development lifecycle (a.k.a Domain Engineering) deals with analyzing the domain and developing customizable process model including the variabilities and configuration options. The core assets of the customizable process model are implemented. This lifecycle generates a customizable process model as well

[2]Some researchers use process lines [161] in analogy with product lines where product lines principles are applied for modeling processes. Both process lines and configurable process models concentrate on modeling variability in process models. Similar to template based approach [34] for implementing product lines, configurable process models can be considered a technique for implementing process lines.

implementation of core process models. The PAIS development lifecycle (a.k.a Application Engineering) deals with capturing the requirements of the target application and deriving target process models based on these requirements. The lifecycle is similar to the methodology mentioned in the previous phase except it reuses the assets developed in the domain engineering lifecycle (i.e. customizable process model and implementation) to develop the target PAIS.

1.2 Research Challenges

In our research we concentrate on the issue related to the development and adaptation of customizable process models. We narrow the scope of the research on modeling phases of these lifecycles i.e. Requirements and Process Analysis and Process (re)design. According to the literature [137, 136, 83] The main issues in developing and customizing process models are: *variability complexity, modeling complexity, configuration complexity, handling delta requirements,* and *customization validation.* In this section we explain the detail of these challenges.

1.2.1 Challenge 1: Variability Complexity

In analogy with product line variability [181], the variability in customizable process models refers to how process variants of a customizable process model may differ from each other [12, 136]. Hence, the variability represents differences among the customized process models, derived from a reference process model, in terms of their process elements. Different variability patterns [12] (e.g., alternative, OR, optional) may occur between different process elements. For example, alternative variability pattern between a set of activities represents that each customized process model has only one of the alternative activities. Also, control-flow variability shows differences of control-flow patterns, defined over the same set of activities, in customized process models. Therefore, many complex variabilities may exist in the reference process model and efficient techniques are required to capture and model those variabilities.

1.2.2 Challenge 2: Modeling complexity

Many of customizable process modeling approaches integrate variability into process models via extending the process modeling language by variability specific constructs [55,

83]. For example, Gottschalk et al. [55] extended YAWL (Yet Another Workflow Language) with hide and block constructs to model the variability and customization options in the workflow models. Incorporating variant-specific information in the constructs of a reference process language [65] overloads process models with selection options (i.e., variation points and variants) and dependencies (i.e., integrity constraints). Consequently, the development and customization complexity of customizable process models dramatically increases. To this end, there are needs for the approaches to consider variability as first class concepts in process modeling and decouple variability modeling from process modeling [58, 106].

1.2.3 Challenge 3: Configuration Complexity

Selecting the right and most important process elements for each individual customer requires understanding his/her requirements as well as the relations of configuration options with the given requirements. Considering the high complexity of customizable models due to including process and variability logic and diversity in customers' requirements, configuring a process is a cumbersome task for process engineers. Specifically, a process engineer needs to take following factors into account when deriving a process from customizable process model.

- Industrial customizable process models may consist of hundreds of configuration options which lead to the exponential growth of the number of possible configured process models. Manually deriving an optimal process model among high number of possible options demands high cognitive load from process engineers.

- Activities may have different quality attributes [17] such as *performance* and *cost* which we refer to as non-functional properties. Stakeholders may have several constraints and preferences over non-functional properties during the process derivation. For example, one stakeholder may ask for a configured process model with *high security*, *high customer satisfaction*, and specific *cost*; and can mention that *customer satisfaction* is more important than *security*; which would make the configuration process more difficult and complex [23, 155].

1.2.4 Challenge 4: Delta Requirements

Many of customizable process modeling approaches derive a customized process from a customizable process model by removing process elements from the customizable process model via operations like hide and block [55]. According to the literature [128], it is unlikely that a reference process model covers all the requirements of a target PAIS. Thus, the process engineers have to customize the configured process model to meet the remaining requirements of the stakeholders. These remaining requirements are known as *delta requirements*. The customization approaches should not only provides mechanisms for deriving a customized process model from a customizable process model, but also should provides the developers with mechanisms for making changes to the customized process model to fit it to target application requirements.

1.2.5 Challenge 5: Customization Validation

While customizing a process model for an individual customer, the generated process should adhere to the regulations defined in a customizable process model. Several configuration and behavioral regulations might be enforced when a customizable process model is constructed. However, derivation of a customized process model from a reference process model is error-prone process especially considering changes need to be done for delta requirements. Hence, the customization approach should guarantee the correctness and compliance of every customized process model with respect to the specified configuration and behavioral constraints and inform process engineers of possible inconsistencies.

1.3 Research Method

Basili [20] discusses three research paradigms in software engineering including *analytical paradigm, scientific paradigm* and *engineering paradigm*. The analytical paradigm (Mathematical) employs mathematical frameworks to develop models and understand their boundaries. Scientific paradigm acquires general knowledge about which technology (process, method, technique, language, or tool) is useful for whom to conduct which (software engineering) tasks in which environments. The engineering paradigm is used for the construction of new solutions (processes, models, methodologies, techniques, and

so forth) [20]. One or combination of methods in these paradigms are used by software researchers to produce knowledge which later will be used by other researchers and software engineers. In my research, I adopted research methods of all these research domains to describe a research method dedicated to my domain (i.e., the development and validation of customizable process models). The following describes three phases of my research method.

Analyzing challenges and issues. First I started with analyzing the process customization domain and identifying the challenges in developing and customizing process models. Hence, a set of objectives was defined: Identification and detail analysis of the problem domain; defining a set of analysis criteria for producing the requirements; and determining the scope of the customization framework and delineating its requirements. According to the defined objectives, in the first task, my colleague and I performed a systematic literature review on the problem domain encompassing existing configuration methods in software product lines, customizable process models, and service oriented architectures. This task involved exploring, accumulating, categorizing, and describing existing configuration (customization) approaches. The result led to an understanding of what is desirable and what is undesirable in a customizable process models and existing challenges. I also investigated theoretical perspectives of a customization framework by applying ontological modeling to identify a set of constructs which are required with customization framework in customizable process models. Finally, as a result of our literature review and theoretical investigation, we generated a set of requirements to be satisfied by the target customization framework.

Planning and design of customization and validation framework. After defining the requirements, a solution was planned, designed, and implemented. The planning and design were the basis for the implementation of the solution. An iterative and incremental approach was adopted to develop the solution in which we designed and implemented possible solutions for already identified requirements and revised the requirements based on the computed results. The approach provides the benefit of investigating the feasibility of our ideas and also takes into consideration the feedback of the domain experts. The design of the customization and validation framework includes: 1) identifying the proper modeling languages for variability modeling and process modeling and performing extensions on these languages based on the knowledge acquired in analysis phase; 2) producing the blueprint of customization process in both domain

engineering lifecycle and application engineering lifecycle along with the architecture of required tooling support by applying the requirements; and 3) identifying inconsistency patterns and deciding on modeling languages for logical reasoning.

Implementation and validation of the solution. In this phase I detailed the process blueprint produced in the design phase and implemented the tools. I created a user guideline which defines the detail specification of the customization methodology and integrated the customization methodology into existing methods for developing customizable process models. I implemented tooling support for my approach by reusing existing tooling from the literature. I mainly used existing tools such as feature model plug-in (fmp)[3] as basis for my tool development. Finally, the proposed customization and validation approach was validated using scientific (empirical) methods. According to Ferimunt et al. [51] three most often used strategies in scientific paradigms are controlled experiments, case-studies, and surveys. Several researchers have applied these strategies [151, 192, 113] and provided guidelines on how to apply these methods for evaluating processes and tools [78]. I also used these research strategies in my context to some extent. We applied literature review in the analysis phase to identify the problems and issues in the customization of process models. In this phase, I applied case-studies for evaluating the customization framework. In this regard, I developed two case-studies one related to online shopping and the other related to health care domain. I employed simulation approaches to explore the capabilities of my proposed customization and validation framework from different perspectives.

1.4 Research Contributions

I addressed the challenges mentioned in section 1.2 by proposing a novel customization and validation approach which utilizes the principles of software product lines, description logic formalism, and artificial intelligence planning.

Software product lines community has extensively investigated the problem of model configuration [125]. Product line approaches consider variability as a first class object and developed dedicated languages such as feature models [73] and Orthogonal Variability Model (OVM) [125] for modeling variability and selection dependencies (i.e., integrity constraints).

[3] http://gsd.uwaterloo.ca/fmp

The existing variability modeling approaches widely use feature models to represent variability in software artifacts including requirements, design, and implementation artifacts [155, 106, 34, 21, 23]. Some researchers [161, 106, 112] employed feature models for a unified representation of the variability of reference process models. This helps manage complexity of process models (challenge 1) and employs the existing configuration approaches in SPLE [23, 106, 155] to configure a reference process model. I adopt a similar strategy for modeling process variability and have made the following contribution:

- A feature-oriented customization approach which 1) addresses more complex variability types (challenge 1) in customizable process models by applying feature models to represent activity variability and implicit variability constructs in the process model to represent control-flow variabilities; 2) defines a guideline to derive a final customized process model from a customizable process model based on the stakeholders' requirements for a target application.

My approach applies restriction strategy by configuring feature models and consequently deriving configured process model and applies extension strategy to further customize the process model based on delta requirements. This approach improves reference process customization [55, 137, 65, 132] in several aspects: 1- handles more comprehensive variability types (challenge 1), 2- manages the complexity of modeling variability along with process logic by employing feature models (challenge 2), 3- proposes an automatic easy-to-use configuration process based on AI planning (challenge 3) and 4- deals with delta requirements by providing an extension strategy (challenge 4).

In addition to customization guidelines, my approach provides a description logic based technique which ensures the correctness of customized process models with respect to behavioral and configuration constraints formulated in reference process models. In this regard, my contributions are:

- A set of inconsistency patterns which may occur between customized process models and customizable process models. These patterns are divided into two groups: *behavioral inconsistency* and *configuration inconsistency* patterns. The former group represents possible inconsistencies between a customized process model and a customizable process model and the latter one reflects inconsistencies between a customized process model and a feature model.

- A formal framework that enables the validation of customizations of reference

process models. The formalism uses Description Logic and reasoning technologies to automatically detect the inconsistency patterns between customized process model and customizable process model and feature models.

The framework is implemented and a set of experiments is designed to evaluate the running time of the algorithms and investigate the performance of the algorithms for different sizes and percentages of variability and inconsistency distributions. The experiments results revealed that the running time of the validation framework is tractable in practical customizable process models with various model sizes and inconsistency distributions. Also, my customization technique, unlike many correctness approaches [168, 64, 140, 134] which only concentrate on behavioral correctness of the customized process model, ensures both behavioral and configuration constraints (i.e. variability and integrity constraints).

1.5 Thesis Organization

- **Chapter 2 - Background:** This chapter describes the fundamentals and important concepts used in the development of the customization and validation framework.
- **Chapter 3 - Overview of proposed approach:** This chapter provides an overview of the proposed customization and validation technique. The relation between different publications used in subsequent chapters and the customization approach is explained.
- **Chapter 4 - Theoretical analysis of variability:** This chapter illustrates the results of theoretical analysis of the variability modeling in process models. It utilizes the ontology theory to define different variability patterns between two information systems.
- **Chapter 5 - Pre-configuration of process variability:** This chapter presents a proposed technique to map requirements, represented by goal model, to the variability model (represented by feature model) and develops a pre-configured feature model based on reasoning on high level objectives of target stakeholders.

- **Chapter 6 - Planning based configuration of process models:** This chapter describes the proposed decision making algorithm as well as planning based technique for developing a configured process model from stakeholders' preferences. The approach is evaluated using several experiments.
- **Chapter 7 - Goal model and process model validation:** This chapter introduces the approach for developing customizable process model and mapping it to a family goal model. The possible inconsistencies patterns between a family goal model and a feature model are identified and description logic based technique is employed to check inconsistencies.
- **Chapter 8 - Goal model and feature model validation:** This chapter introduces the notion of product line variability in goal models and proposes a description logic based approach to ensure alignment of the feature model variability with family goal model variability.
- **Chapter 9 - Customized process model validation:** This chapter introduces a technique for the customization of process models from a configured process model. Behavioral and configuration inconsistency patterns which may occurs during the customization are identified and description logic is used to ensure validity of final customized process model.
- **Chapter 10 - Conclusions and future works:** This chapter concludes our research work and point out the limitation and the possible future directions for the approach.

Chapter 2

Background

Variability is a core notion in customizable process models. Therefore, in order to have a deep understanding of variability, we employed ontological theory to investigate the variability in information systems. Section 2.1 explains the principle of ontological theory and introduces Bunge-Wand-Weber(BWW) ontology. The proposed customization and validation approach covers requirements and design phases of developing customizable process models. The approach relies on goal oriented techniques for capturing and modeling stakeholders' requirements, and consequently, the goal models are described in Section 2.2. Additionally, process models are introduced in more details in Section 2.3, which follows with definition of feature models in section 2.4. In order to find an optimal configuration, we utilize the Artificial Intelligence (AI) planning technique which is described in Section 2.5. Section 2.6 introduces Description logic, which is used for the validation purposes in our approach.

2.1 Theoretical Foundation in Information Systems

A growing interest in using conceptual modeling in requirements engineering has motivated work on theoretical analysis of the meaning of modeling constructs. Using ontological theories is one category of approaches employed in such analyses. The application of a theory of ontology in conceptual modeling is based on the representational premise "information systems provide an artifactual representation of real-world systems (as perceived by someone)" [176]. According to this premise, an ontology can provide a basis for evaluating conceptual modeling languages in terms of their ability to describe a domain

for which an information system is being developed. Wand and Weber [176] identified two evaluation criteria based on this idea - ontological completeness and ontological clarity. Ontological completeness implies that "the mapping from ontological constructs to language is total" [179]. Ontological clarity means that "the mapping from language constructs to ontological constructs is total and one-to-one" [179].

Bunge's ontology is widely used for evaluating conceptual modeling languages [47, 179, 129]. In Bunge's ontology [29], the "world is made up of substantial things which possess properties". Examples of things are specific instances of person, car, or book and properties are name, color, or weight. A thing can be either primitive or composite where "a composite thing may be made up of other things" ([176] p. 8). A property can be either intrinsic (i.e., is possessed by one thing) or mutual ("meaningful only in the context of two or more things" [176]). A composite thing can possess emergent properties that are not possessed by any of its components. Since properties cannot be observed directly, attributes are defined to represent the properties of a thing as perceived by an observer. An attribute is a function that maps a thing to a value at some point of time (and possibly additional conditions). "Properties in general are represented by state functions, the values of which express individual properties" ([47] p.4). A functional schema is formed by a set of state functions that are used to describe similar things. A state is "the vector of values for all property functions of a thing" [47]. An event is a change of a state of a thing [47]. A state law defines any restriction on the individual properties (or attributes) of a thing. The process of a thing comprises the events that the thing might undergo and is determined by a set of transition (transformation) laws [176]. A set of things that possess a common property is termed a class. A set of things having several properties in common is called a kind. A natural kind is a kind of things adhering to the same laws. Bung's ontology defines more concepts that will be mentioned when are needed in the thesis.

2.2 Goal Models

To date, several languages for goal modeling have been proposed, e.g., i*/Tropos [186, 54], NFR [31], KAOS [173] and Goal Requirements Language (GRL) [68]. We adopted GRL, a part of the recent Recommendation of the International Telecommunications Union named User Requirements Notation (URN) [68]. GRL is a language that integrates core concepts of i* and NFR [7].

A goal model consists of *actors*, *intentional elements*, and *links* [68]. *Intentional elements* are *(hard) goals*, *soft goals*, *tasks*, and *resources*. A (hard) goal is a state of affairs that an actor would like to achieve. *Soft goals* are similar to hard goals, but without a clear-cut criteria if the condition is achieved or not. *Tasks* specify conceptual solutions in order to achieve a particular intentional element. *Resources* are informational or physical entities, which are available to actors. *Actors* are entities and refer to stakeholders or systems. Actors can have intentions. In this case, we say an actor has a scope and activities of this actor are within the actor's scope.

A concrete goal model is depicted in Figure 2.1. The goal model is part of an online shopping case-study, which is used throughout this chapter. The goal model contains five actors, namely Front Store, Back Store, Customer, Shipment, and Supplier. When representing user intentions for a system or a part of a system, actors are either *internal* or *external* actors. *Internal* actors are part of the system-to-be, while *external* actors represent stakeholders that interact with system actors. In the goal model in Figure 2.1, only the intentional elements within the scope of the *internal* actor Back Store are depicted.

We distinguish between two kinds of links. (1) *Dependency* links connect intentional elements and/or actors. (2) *Decompositions* and *contributions* link intentional elements[1].

Dependencies describe inter-actor relationships. They express how a source actor (depender) depends on a destination actor (dependee) for an intentional element (dependum) [68]. Dependency relations are only defined between different actors and/or intentional elements of different actors, i.e., intentional elements in the scope of different actors. Hence, a dependency link is not defined between intentional elements of one actor. According to the type of the dependum, dependencies are called task, (hard) goal, soft goal and resource dependencies [189]. For example, Front Store actor depends on Back Store actor for achieving Process Order objective, and the Supplier actor depends on Back Store actor for the soft goal Continuing Business.

Decompositions specify which intentional elements a certain intentional element needs for its satisfaction. For instance, the goal Ship & Bill is satisfied if both tasks Ship Order and Bill Preparation are fulfilled. GRL supports *AND*, *IOR* and *XOR* decompositions.

[1]According to the GRL standard [68] it is also possible to specify correlation links. Correlation links are similar to contribution links, except that they model side effects. From the goal satisfaction perspective both of these links are the same. For this reason, we restrict ourselves only to contribution links.

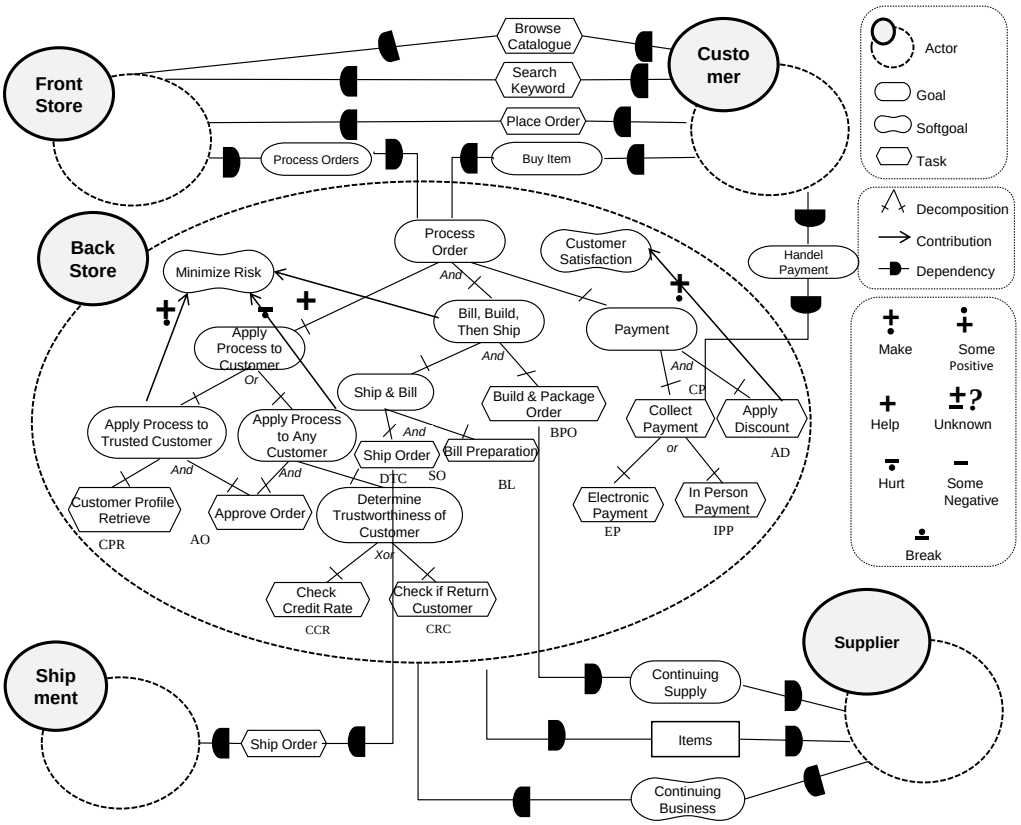


Figure 2.1: A part of goal model for online shopping system

An *AND* decomposition specifies that all source intentional elements need to be satisfied for the target intentional element to be satisfied. *IOR* is used to specify that the satisfaction of at least one source satisfies the target, whereby *XOR* specifies that exactly one of the source elements is necessary to satisfy the target.

Contributions express how an intention contributes to the achievement/fulfillment of another intention. Figure 2.1 shows different contribution types. The Make and Break contributions are respectively positive and negative, and sufficient for the fulfillment of a target element. Help and Hurt contributions are also positive and negative respectively, but insufficient. The extent of the contribution of SomePositive and SomeNegative is unknown. Finally, for the Unknown contribution link, both the extent and degree (positive or negative) of the contribution are unknown.

Definition 2.1 compactly states the specification of goal models.

Definition 2.1 (Goal Model) *A goal model is a quintuple* $GM = (\mathcal{A}, \mathcal{G}, \mathcal{P}, \mathcal{D}, \mathcal{C})$. $\mathcal{A}$ *denotes a set of actors.* $\mathcal{G}$ *is a set of intentional elements (also called intentions or goals). Intentions are (hard) goals (*$\mathcal{G}_g$*), tasks (*$\mathcal{G}_t$*), soft goals (*$\mathcal{G}_s$*), and resources(*$\mathcal{G}_r$*) .*

- $\mathcal{P}$ *is a set of dependency links:*
  $\mathcal{P} \subseteq (\mathcal{A} \cup \mathcal{G}) \times \mathcal{G} \times (\mathcal{A} \cup \mathcal{G})$.
- $\mathcal{D}$ *are decompositions of intentional elements:*
  $\mathcal{D} \subseteq \mathcal{G} \times \{AND, IOR, XOR\} \times \mathbb{P}(\mathcal{G})$.[2]
- $\mathcal{C}$ *denotes positive and negative contributions:*
  $\mathcal{C} \subseteq \mathcal{G} \times \{\overset{+}{\bullet}, \overset{\bullet}{+}, +, \pm?, \overset{-}{\bullet}, -, \overset{\bullet}{-}\} \times \mathcal{G}$.

Regarding this definition, dependencies ($\mathcal{P} \subseteq (\mathcal{A} \cup \mathcal{G}) \times \mathcal{G} \times (\mathcal{A} \cup \mathcal{G})$) are relations between intentional elements ($\mathcal{G}$) within the scope of different actors or between actors ($\mathcal{A}$). For instance in Figure 2.1, there is a dependency from actor Front Store to hard goal Process Order, which is in the scope of actor Back Store. The second argument ($\mathcal{G}$) refers to the dependum, e.g., the hard goal Process Orders in Figure 2.1.

In essence, goal models represent user intentions, relationships between intentions, actors, which might have intentions within their scopes, and dependencies between actors.

[2] $\mathbb{P}(\mathcal{G})$ *denotes the power set of* $\mathcal{G}$.

2.3 Business Process Models

There is an emergence and proliferation of process-oriented software development methods for organizations and enterprises. Today, software products as services are designed, built, executed, maintained and evolved by means of business processes on top of Web services technologies [43, 119]. To accomplish this, business process modeling distinguishes between the concepts of *orchestration* and *choreography* [123].

A process *orchestration* describes the control flow of activities from the perspective of one party. It refers to an executable part of a business process that is provided by one party through a central coordination. Activities realize and implement certain intentions that can be specified by a requirement model [48].

Process *choreography* takes inter- and intra-organizational processes and communication between them (e.g., B2B and B2C) into account. This may involve multiple parties that perform different parts of the overall business process. Choreography describes the set and ordering of interactions (i.e., message exchanges) between individual parties and defines the expected behavior among the interacting parties, i.e., a type of procedural contract or protocol.

2.3.1 Process Orchestration

Workflow patterns [169] describe the orchestration of processes. They define how the process flow proceeds in sequences and splits into branches and how branches converge.

Business process models can be designed at different levels of abstraction. A number of graph-based business process modeling languages have been proposed, e.g., Event-driven Process Chain (EPC), Yet Another Workflow Language (YAWL) [167] and UML Activity Diagrams. We use the Business Process Modeling Notation (BPMN)[118] as a base and standard modeling language. Nonetheless, the proposed approach given in the present work is independent of any goal and process modeling languages. Despite of a variance of modeling notations and different semantics, all modeling languages share the common concepts of activities, gateways (or routing nodes) and relationships between them.

Definition 2.2 (Business Process Model) *A business process model is defined as a connected graph* $\Gamma_{PM} = (\mathcal{V}, \mathcal{E})$*, where* $\mathcal{V}$ *depicts a set of vertices, denoting activities* $\mathcal{A}$ *and gateways* $\mathcal{G}$ *(*$\mathcal{V} = \mathcal{A} \uplus \mathcal{G}$*). A gateway* $G \in \mathcal{G}$ *has a type* $T(G)$ *such that* $T(G) \in$

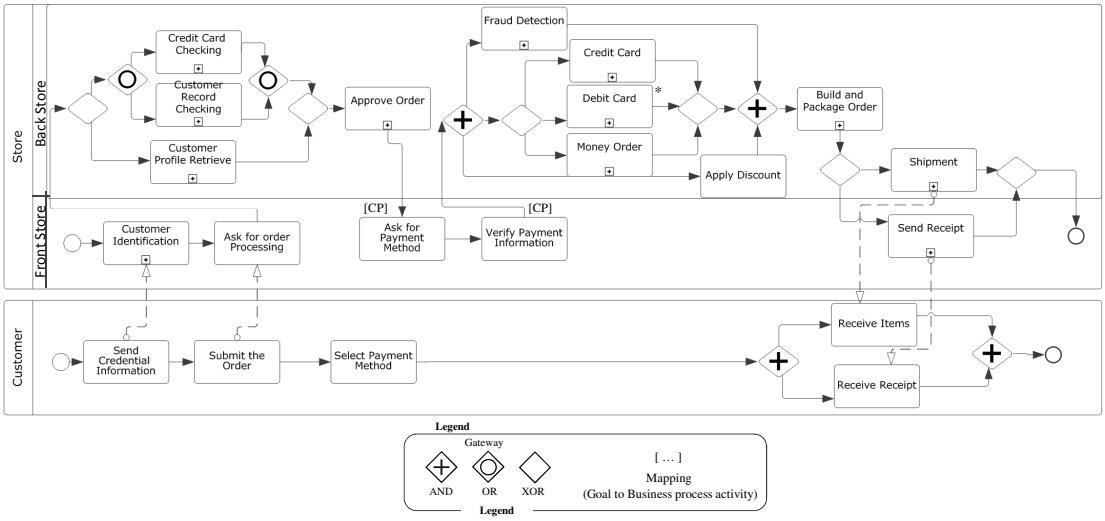


Figure 2.2: A business process model for process order

$\{AND, IOR, XOR, DISC\}$. $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ *denotes a set of edges between vertices.*

To allow the analysis of activities and their relationships with intentions and their corresponding intentional relationships, we focus on structured process models, which are more comprehensible and less error-prone [77]. We consider structural well-formedness conditions for business process models [82, 174], where each process model is decomposed into aggregated SESE (single-entry-single-exit) fragments. SESE fragments can be derived from a process model in linear time (cf. [71]). According to these structural well-formedness conditions, gateways can have either exactly one incoming edge and multiple outgoing edges or multiple incoming edges and exactly one outgoing edge. Furthermore, we assume a materialized representation according to Definition 2.3.

Definition 2.3 (Materialized Business Process Model) *Given a business process model* $\Gamma_{PM} = (\mathcal{V}, \mathcal{E})$*, a materialized process model* PM *is a quintuple* $(\mathcal{A}, \mathcal{G}, \mathcal{E}, \mathcal{F}, \mathcal{E}_A)$*. Vertices* $\mathcal{V}$ *are either activities* $\mathcal{A}$ *or gateways* $\mathcal{G}$ $(\mathcal{V} = \mathcal{A} \uplus \mathcal{G})$*. The set* $\mathcal{F} \subseteq \mathcal{V} \times \mathcal{V} \times \mathcal{B}$ *represents single-entry-single-exit (SESE) fragments, in which* $\mathcal{B}$ *represents a set of branches between entry and exit vertex. A branch* $B \in \mathcal{B}$ *is considered as a set of atomic activities.* $\mathcal{E}_A \subseteq \mathcal{E}^+$ *is a set of sequence edges obtained from the process graph by treating gateways as transparent.*[3]

[3] *The set* $\mathcal{E}^+$ *denotes the transitive closure on the set of edges* $\mathcal{E}$*.* $\mathcal{E}_A$ *denotes the subset of the transitive*

The set $\mathcal{E}_A$ captures the explicit predecessor and successor relationships between atomic activities without gateways. This set is derived from the set of edges $\mathcal{E}$ by neglecting gateways and replacing them by their corresponding next predecessor or successor activity. A business process model is illustrated in Figure 2.2.

Fragments $\mathcal{F} \subseteq \mathcal{V} \times \mathcal{V} \times \mathcal{B}$ (as introduced in Definition 2.3) can be found for instance in the Customer process on the right side, where AND gateways are the start and end point of the fragment and $\mathcal{B}$ consists of two branches (between the opening and closing gateway). Each branch $B \in \mathcal{B}$ contains only one activity (Receive Items and Receive Receipt).

In BPMN, process orchestrations are represented within container called pools. In Figure 2.2, there are two pools: Store and Customer. Both pools contain a business process. A pool refers to a participant and its local process view. In the pool Store, the process is spread over two lanes (FrontStore and BackStore). A lane is a sub-partition of a process.

2.3.2 Process Choreography

A choreography defines the sequence of interactions between participants in terms of message exchange, which appears between pools in BPMN models. A choreography does not exist within a single pool (i.e., in a local context of control). Figure 2.2 illustrates an example of a choreography between the pools Store and Customer. For instance, there is a message sent from the activity Submit the Order in the pool Customer to the activity Ask for Order Processing in the pool Store. Message flows between pools define a choreography [185]. We consider a process choreography as a set of materialized business process models and a set of message flows between activities, formally defined as follow.

Definition 2.4 (Process Choreography) *Given a set $\mathcal{P}$ of materialized business process models. The set $\mathcal{M} \subseteq \hat{\mathcal{A}} \times \hat{\mathcal{A}}$, with $\hat{\mathcal{A}} = \bigcup_{P \in \mathcal{P} \wedge A = activities(P)} A$ is a set of message flows. A process choreography is a tuple $Ch = (\mathcal{P}, \mathcal{M})$ with the condition: $\forall (A_1, A_2) \in \mathcal{M} : \Rightarrow A_1 \in activities(P_1) \wedge A_2 \in activities(P_2) \wedge P_1 \neq P_2$.*[4]

closure such that activities are either directly connected or transitively connected, but only with gateways in-between them.

[4] *$A_i \in activities(P_i)$ is an abbreviation to denote that activity $A_i \in \mathcal{A}$ occurs in process $P_i = (\mathcal{A}, \mathcal{G}, \mathcal{E}, \mathcal{F}, \mathcal{E}_A)$.*

For each message exchange between A_1 and A_2 ($(A_1, A_2) \in M$) it is required that A_1 and A_2 are activities of different processes. We require that each process is in a pool and each pool contains one process.

2.4 Extended Feature Model

Feature models are the most common language used in software product line engineering for representing variability and commonality in a variety of development artifacts including requirements, design, and test [21]. The core notion in feature models is the *feature*. Bosch et al. [27] defined a feature as "a logical unit of behavior specified by a set of functional and non-functional requirements." We consider this definition as we intend to customize process models based on both functional and non-functional requirements. A feature model provides a formal and graphical representation of features as well as the variability relations, constraints, and dependencies defined over the product line features. It has a tree-like structure [34] in which features are typically classified as:

- **Mandatory feature:** if a parent feature is selected, its mandatory child feature must be also selected in the configuration process.
- **Optional feature:** if a parent feature is selected, its optional child feature may or may not be selected in the configuration process.
- **OR feature group:** one or more features in the OR feature group must be selected in the final configuration of the feature model.
- **XOR feature group:** one and only one of the features in the XOR feature group must be selected in the final configuration of the feature model.

In our approach, we assume that atomic features (i.e., leaf features) in a feature model have concrete implementations (they are mapped to the activities in process models). Non-atomic (i.e., non-leaf) features are used for variability and composition relationships of the atomic features. Hence, non-functional properties are defined for the leaf features. If an intermediate feature contains implementations (i.e. map to subprocess with implementation), we create a mandatory child feature for the intermediate feature and map the implementation to the child feature.

In addition to the relations between a parent feature and its child features, a number of relations are defined to represent mutual inter-dependencies (also referred to as

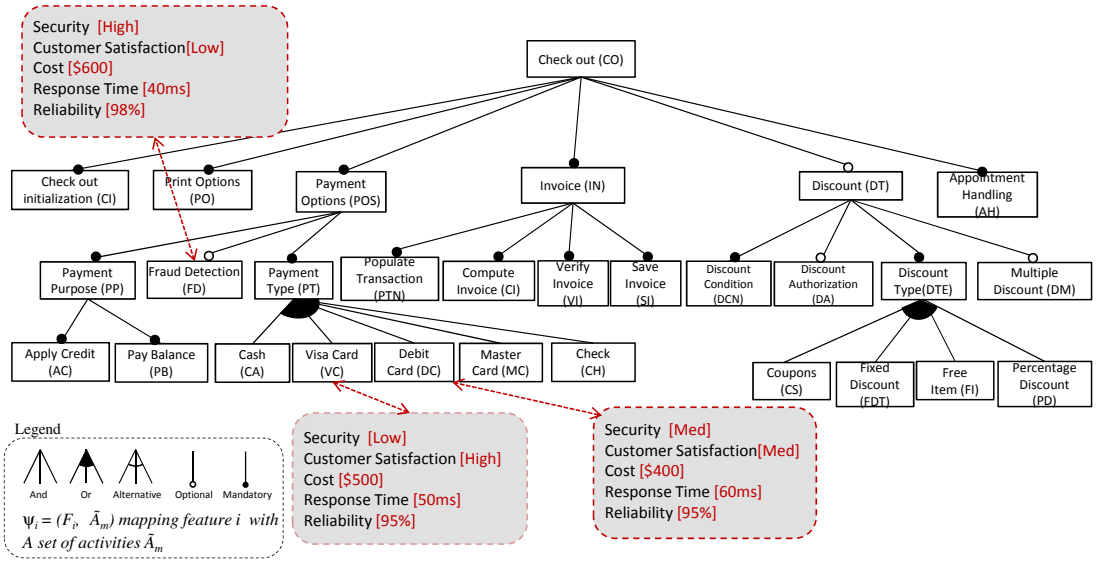


Figure 2.3: Integrated Feature Model with Non-Functional Properties

integrity constraints) between features. The two most widely used integrity constraints are [34]: *requires*-the presence of a given (set of) feature(s) requires an inclusion of another (set of) feature(s); and *excludes*-the presence of a given (set of) feature(s) requires an exclusion of another (set of) feature(s). Batory [21] argued the need for more complex integrity constraints and generalized the constraints into propositional formulas defined over features. In our work we adopt Batory's view over integrity constraints. In feature model tree, features without any children called *atomic* (or leaf) features and features which are decomposed into sub-feature(s) called *non-atomic* (or intermediate) features.

In feature oriented software product line, feature models are mainly used for representing functional variability between different products. Some researchers have extended the feature model notation with non-functional properties to represent the non-functionality aspect [23, 148, 155]. In the extended feature model each feature can be annotated with several non-functional properties and corresponding values of those properties. The non-functional properties are divided into qualitative non-functional properties and quantitative non-functional properties.

Figure 2.3 shows non-functional properties employed in the on-line shopping product line. *Response time*, *cost*, and *reliability* are quantitative non-functional properties and c*ustomer satisfaction* and *security* are the qualitative non-functional properties. For

qualitative non-functional properties such as *security*, *international sale*, based on existing domain knowledge, the impact of each feature on non-functional properties can be identified and proper qualifier tags can be assigned to each feature's non-functional properties. On the other hand, quantitative non-functional properties for the features can be measured using a suitable metric and assigned to the features.

For example, as shown in Figure 2.3, feature Visa Card is annotated with *low security* and *high customer satisfaction* and its estimated *cost*, *response time*, and *reliability* are \$500, 50ms, and 95%, respectively.

2.5 Planning Definition Model

Planning Domain Definition Language (PDDL) is a standard language for expressing the planning tasks and problems [49]. PDDL describes several constructs and features which are used in different planning contexts. Hence, we only introduce the main concepts which we used for the configuration context.

The main concept in the PDDL is an action which contains precondition and post condition to define the applicability and effect of the action [49]. *Domain predicates* define relevant properties in the domain that can be true or false. Domain predicates define the knowledge in the world. *Functions* describe quantitative functions in the domain. *Initial state* and *Goal* describe the state of the world when planning begins and what we want to achieve, respectively.

In the PDDL, planning tasks are separated in two files namely domain file and problem file. Domain file is used to define predicates and actions and a problem file describes the initial state, goal specification, and metrics.

2.6 Description Logic

There are several logical knowledge representation languages like propositional logic, first-order logic and temporal logic. Other formalisms might use traversal algorithms to compare process graphs, while further approaches compare execution traces of process models. We use Description Logics (DL) [16], a decidable subset of first-order logic (FOL), as a common representation formalism. DL offer an expressive representation language, a well-defined semantics and practically efficient reasoning services.

Table 2.1: Constructs and Notations in DL and FOL Syntax. (Legend: $\sqsubseteq$ subsequent, $\sqcup$ Union, $\sqcap$ Intersection, $\neg$ negation, $\forall$ Universal, $\exists$ Existential, $\rightarrow$ implication, $\vee$ logical OR relation, $\wedge$ Logical AND relation.)

Construct Name	**DL Syntax**	**FOL Syntax**
Atomic concept, atomic role	C, R	$C(x)$, $R(x, y)$
Concept inclusion axiom	$C \sqsubseteq D$	$\forall x.C(x) \rightarrow D(x)$
Concept union	$C_1 \sqcup \ldots \sqcup C_n$	$C_1(x) \vee \ldots \vee C_n(x)$
Concept intersection	$C_1 \sqcap \ldots \sqcap C_n$	$C_1(x) \wedge \ldots \wedge C_n(x)$
Concept negation	$\neg C$	$\neg C(x)$
Universal quantification	$\forall P.C$	$\forall y.(P(x, y) \rightarrow C(y))$
Existential quantification	$\exists P.C$	$\exists y.(P(x, y) \wedge C(y))$

DL are a decidable subset of first-order logic. A DL knowledge base consists of a TBox (Terminological Box) and an ABox (Assertional Box). The TBox is used to specify concepts, which denote sets of individuals, and roles defining binary relations between individuals. The main syntactic constructs are depicted in Table 2.1, supplemented by the corresponding FOL expressions. Concept inclusion axioms $C \sqsubseteq D$ mean that each individual of the concept C is also an individual of D. There are two special concepts, namely the universal concept (top concept) $\top$ and the bottom concept $\bot$. The top concept $\top$ is the superconcept of all concepts, i.e., $C \sqsubseteq \top$ holds for each concept C. $\bot$ is an unsatisfiable concept. A concept equivalence (or definition) $C \equiv D$ is an abbreviation for two concept inclusion axioms $C \sqsubseteq D$ and $D \sqsubseteq C$. Reasoning services of DL rely on the well-defined semantics. We use subsumption checking and concept classification as basic reasoning services to detect inconsistencies between models.

Chapter 3

Approach Overview and Discussion

In this chapter, I introduce different parts of my proposed approach that addresses challenges identified in the introduction chapter. Next, I discuss how my published articles relate to different parts of proposed approach.

3.1 Approach Overview

The main contribution of this research is an approach for developing, adapting, and validating customizable process models. The proposed approach covers requirements analysis and design phases and utilizes three modeling paradigms namely goal modeling, process modeling, and feature modeling. A goal model represents the domain requirements in terms of main actors, their intentions and tasks, and their dependencies. A process model represents the control-flow of activities within a process aware information system, and a feature model depicts the variability options in both requirements (i.e. goal model) and design models (i.e. process models). As shown in Figure 3.1, all these three models are related via traceability links (i.e. mapping).

Figure 3.2 shows our proposed approach. The approach is developed in the context of a research project in the Laboratory for Ontological Research at Simon Fraser University. As shown in figures 3.2, the proposed approach consists of *domain engineering* and *application engineering* lifecycles. The domain engineering lifecycle concentrate on developing and implementing customizable process models. In order to implement

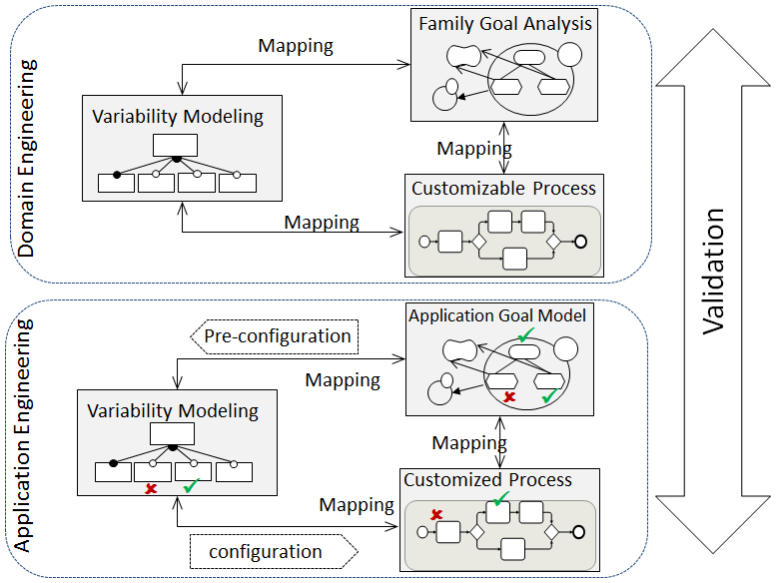


Figure 3.1: Product overview of the proposed approach

a customizable process model, Service Oriented Architecture(SOA) paradigm can be employed where activities are realized by services. In the application engineering lifecycle, a new process-aware information system is built by adapting the customizable process model and tailoring the implementations to fit to the customers requirements. In this research, we focus on the analysis and design phases of the approach and the implementation is outside the scope of our research.

In the domain engineering (reference model development) lifecycle, the overall requirements of a domain are captured and formulated in a family goal model. The family goal model is an extension on the standard goal modeling which discriminates between software variability (variability in a single software system [102]) and product line variability (variability between different software systems [102]) in the intentional space. Product line variability in an intentional space means variation in the objectives of different customers. Later, the product line variability and software variabilities in the family goal models are converted into variability and commonality relations in feature models. According to the intentional and behavioral relations as well as existing tasks in the family goal model, a customizable process model is created. The product line variabilities in the customizable process model as well as selection dependencies between activities are inserted into the feature model.

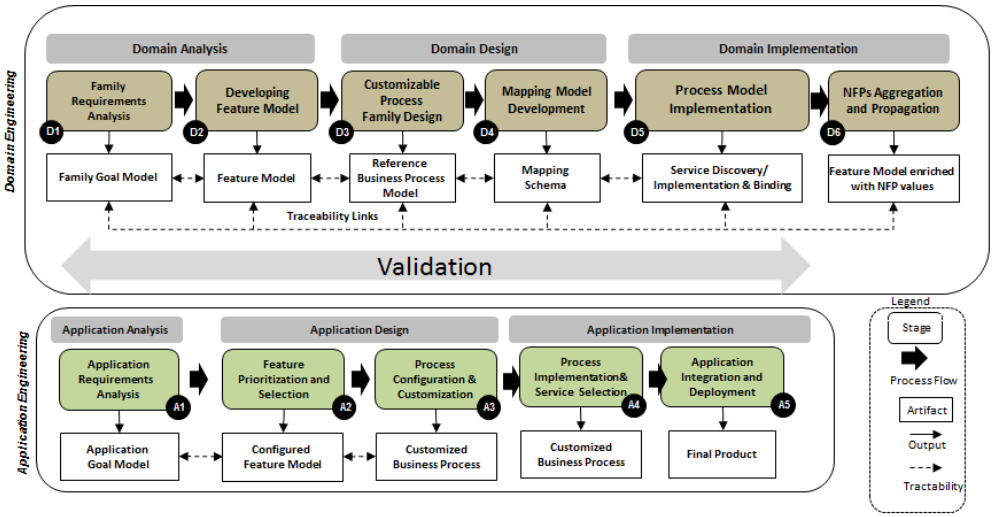


Figure 3.2: The proposed method

In the application engineering (customized process development) lifecycle, a customizable process model is adapted based on the requirements of customers. First, the objectives and preferences of customers are captured and the product line variabilities in the family goal model are resolved. Next, according to the objectives, the feature model is preconfigured by deselecting the features which are mapped to denied goals. Afterwards, the customers' preferences and constraints are used to reach a fully-configured feature model. Based on the mapping between the feature model and the customizable process model, activities corresponding to deselected features are removed from the customizable process model. Finally, the process model is further adapted to satisfy the requirements which are not covered with the customizable process model. In the remaining parts of this section, we explain the phases and artifacts of our approach for developing and customizing process models.

3.1.1 Domain Engineering(Reference Model Development)

The core notion in developing customizable process models is managing variability. Several researchers have investigated the notion of variability in the context of software product lines with different perspectives and assumptions. Mainly, existing points of view on the notion of variability can be categorized into two classes:

- Bosch et al [172] and Sinnema et al [40] defined a variability as "the ability to be

changed, customized, configured, or extended for use in a specific context",

- Weiss et al [181] defined variability as "an assumption about how members of a family may differ from each other".

Mentzger et al [102] referred to the first class as a software variability and the second class as a product line variability. Software variability documents the design and realization/implementation decisions such as classes with a common interface and different implementations or executable files with alternative object modules. On the other hand, product line variability represents differences between the systems that belong to a product line in terms of properties and qualities [102]. Product line variability differs from software variability; product line variability is an explicit decision of product management about what should vary between the systems in a product line and what should not, while software variability is an inherent property of the developed software[102]. For example, in an on-line shopping system, the product line manager can decide to include all three types of payment methods including credit card, debit card, and cash; and hence they cannot be varied between different members of the product line, they can be considered as software variability, but not as product line variability (product line commonality). On the other hand, if the product manager considers variation of payment methods in product line members, both product line variability and software variability exist. In our proposed technique we utilized a similar approach for variability in the process aware information systems and customizable process models. We define *product line variability* and *behavioral variability* classes. The product line variability refers to differences between systems in a family of process aware information systems, which may exist among their requirements, design models and implementation models [181]. On the other hand, behavioral variability represents the various ways that a single process aware information system may be used by its users [92]. For example, workflow patterns in process modeling languages such as BPMN, BPEL, and activity diagrams provide mechanisms for representing variability in behavior of a single process aware information system.

Having clarified the product line variability and the behavioral variability, in the phases of the domain engineering lifecycle, we discriminate these two types of variabilities and use different mechanisms for representing them. In the remaining of this section, we describe our proposed family requirements analysis and customizable process model design phases along with the proposed extensions on modeling languages.

Family Requirements Analysis

The basic idea of process aware information system development is to first understand the requirements, and then develop process models which fit these requirements. Hence, the efficient management of requirements is highly important in the success of a process aware information system development. On the other hand, design and implementation of process aware information systems concentrates on the technical and implementation aspects. The requirements can be lost in the technical architecture of the systems if traceability between design and requirements is not considered. When developing a customizable process model for a family of process aware information systems, variability in customers' requirements signifies the criticality of requirements engineering phase for the success of developing a family of process aware information system. In order to effectively manage variability in a customizable process model, we need to understand the possible differences between the requirements of different customers. In the remaining of this section, the activities for developing goal model and analyzing product line variabilities in requirements are described.

The requirements model for a process aware information system is derived from the potential business goals and objectives of organizations. Developing a reference requirements model mainly consists of *requirements engineering activities* and *variability development activities*. The former concentrates on capturing and modelling the business goals and objectives in the domain and later identifies product line variability in the requirements. The product line variability in requirements can be represented in a separate dimension and realized in the requirements model.

Develop goal model: since we adopted goal-oriented approach for capturing requirements model, we follow existing guidelines for identifying the objectives and illustrate the relation between objectives in a goal model. The process starts with identifying the main actors in a domain and dependency relations between those actors. Next, the high level goals and soft-goals for each actor are discovered and then the goals and soft-goals are decomposed into lower level goals and finally tasks. Goal decomposition is done by following the framework proposed by Liaskos et al. [91] where intentional variability concerns (i.e. software variabilities) are recognized for each goal. Then, the goals are refined according to the variability concerns. After refining goals, requirements engineers analyze the impacts of each sub-goal on the soft-goals and model their impacts using contribution links.

Analyze product line variability: In order to incorporate product line variability in requirements (i.e. variability in the objectives of customers), we extended goal models with some notations. In family goal model, OR and XOR relations can be annotated with *VP* label to show if the relation is product line variability. Additionally, for AND-decomposed intentional elements, we defined the notion of optional which shows whether the source intentional element is necessary for the achievement of the target intentional element. Therefore, using these extensions on the goal model, we analyse the goal model with respect to product line variability and annotate the goals with the proper annotations.

Customizable Process Development

In the context of process aware information systems, process models are employed to represent the high level system design. A customizable process model represents a family of process models in a domain where all the members of a family are merged in one process model (i.e., the customizable process model realizes a superset of all members of the family). Customizable process models describe comprehensive business logic for orchestration and choreography of activities (which are then assigned to the services) and are used to support the construction of customized process models [137, 163]. Execution semantics of reference process models is described using control flow patterns [169]. In the remaining of this section we describe product line variability in customizable process models.

A customizable process model is designed according to the family goal model. Hence, the existing intentional relations in the family goal model can be utilized to guide process engineer to create activities and workflow relations in the customizable process model.

Analyze product line variability: In order to support the development of different customized process models from a customizable process model, the product line variability must be included in the customizable process models.There are two common product line variability types in a customizable process models [13]: *activity variability* and *work flow variability*.

Activity variability is concerned with variability in a customizable process model activities. These are the activities (called optional activities) which may be included/excluded for different customized process models according to the preferences and requirements of a target application. The complex variability relations may exist between a set of

optional activities encapsulated in a sub-process of a customized process model such as exactly one of optional activities in a sub-process must be included in the customized process model.

Workflow variability refers to the type of variability that may appears in the business process logic (i.e., workflow patterns) of customized process models. Hence, in the customizable process model, typical workflow pattern is created in business process which is later customized into special cases according to the customers' requirements. Further investigation of sequential patterns reveals another type of variability in reference process models, which is referred to as order variability [12]. This type of variability exists when a set of activities must be executed in sequential order, but the order of their executions differs in various customized processes. In our context, we employ Business Process Modeling Notation (BPMN) for representing customizable process models. The workflow variability is shown in the customizable process model by annotating workflow patterns with VP notation. The order variability is illustrated using ad-hoc processes in the BPMN.

Feature Model Development

In order to represent a holistic view of product line variability in a separate dimension, we use feature models. Feature models are the most common variability modeling language. I use feature models to represent the variability in both family goal models and customizable process models. Developing of feature models is executed concurrently with the variability analysis activities in previous two phases (i.e., family goal model and customizable process model development phases). In the following, I describe the main activities for formulating product line variabilities in feature models.

Formulate requirements product line variability in feature model: In addition to representing variability in the family goal model, we represent product line variability in a feature model to address the challenge one. Therefore, the feature model is generated from the family goal model by converting product line variability into variability relations in the feature model. The process for developing a feature model from a family goal model consists of: 1) designing the features for the tasks in the goal models; 2) mapping features into the corresponding tasks; and 3) and developing intermediate features according to the existing variability relations in the goal model (i.e. modeling variability). The feature model can be semi-automatically generated from the family

goal model. Additionally, the constraints over features should be captured and modelled in the feature model. Dependency relations capture integrity constraints which involve: require — the presence of a given (set of) features(s) requires the inclusion of another (set of) features(s) — and exclude — the presence of a given (set of) feature(s) requires the exclusion of another (set of) features(s).

Formulate process product line variability in feature model: This activity is performed concurrently with product variability analysis in customizable process model development. Since both the feature model and the customizable process model are derived from the family goal model, a part of variability of the customizable process model is already depicted in the feature model. This step captures the product line variability which may be introduced in the customizable process models in design phase and add these variabilities into the existing feature model.

Mapping Model Development

Developing a mapping model is performed concurrently by the activities in the previous phases. The mapping process builds *correspondence* between the elements of a source model and a target model. The mapping is developed between the goal model, the process model, and the feature model.

Develop goal model and process model mapping: The process models are designed to realize the business objectives of stakeholders which are formulated in family goal models. Tasks in the family goal model are those intentional elements that need to be performed in order to achieve the specified requirements. The basis for aligning process models with goal models is to map *tasks* of the goal model to *activities* (atomic activities or subprocesses) in the process model in order to express the implementation of a goal by an activity. We cover the following mapping rules between elements in the goal models and process models, which follow basic mapping and realization principles in the literature [79, 75, 96]:

- *Actor mapping:* The actors within a goal model are mapped to parties that execute activities in the business process, i.e., to swim-lanes and pools. In our case, each pool contains one process.
  - Internal actors (actors that belong to the system-to-be) are mapped to the lanes of a pool. If a pool consists of multiple lanes the activities of a process (in this pool) might be distributed over the lanes of the pool.

  - External actors (actors that interact with the system-to-be) are mapped to pools.

- *Intentional element mapping:* Among the intentional elements, tasks are implemented by activities in a process. This is expressed by mappings.
  - Tasks in the goal model are mapped to the activities (either atomic or composite, i.e., sub-processes).

These mapping principles defined above follow the modeling intuition of goal and process models: (i) Actors in a goal model represent active entities that perform actions to achieve their goals. These goals are within the actor's scope. Likewise in the process model, lanes group activities of a process and pools group processes that refer to different roles. (ii) Tasks in the goal model are achieved by executing activities (atomic or composite activities) in a process.

Definition 3.1 formally define the goal-process mapping model.

Definition 3.1 (Goal-Process Mapping Model) *Let* $FGM = \langle \mathcal{A}, \mathcal{G}, \mathcal{P}, \mathcal{D}, \mathcal{C} \rangle$ *be a family goal model where* $\mathcal{G} = (\mathcal{G}_g \cup \mathcal{G}_t \cup \mathcal{G}_s)$ *and* $PM = (\mathcal{V}, \mathcal{E}, \mathcal{F}, \mathcal{E}_A)$ *be a customizable process model defined by a structured process model. a goal-process mapping model is* $\Phi_{G-P} \subseteq FGM \times PM = \{\Phi_i(G_i, \mathcal{A}_i) : G_i \in (\mathcal{G}_g \cup \mathcal{G}_t)$ *and* $\mathcal{A}_i \subset \mathcal{A} \subseteq \mathcal{V}\}$.

Develop goal model and feature model mapping: As we explained before a family goal model captures both product line and behavioral variability in the requirements. In order to provide a holistic variability representation, we transfer product line variability in requirements and process models into the feature model. To this end, features are mapped to tasks in the goal model and represent the realization of tasks. Mapping features to tasks is aligned with the feature definition given in the section 2.4 where a feature defined as a logical unit of behavior. The definition 3.2 describes the goal-feature mapping model.

Definition 3.2 (Goal-Feature Mapping Model) *Let* $FGM = \langle \mathcal{A}, \mathcal{G}, \mathcal{P}, \mathcal{D}, \mathcal{C} \rangle$ *be a family goal model where* $\mathcal{G} = (\mathcal{G}_g \cup \mathcal{G}_t \cup \mathcal{G}_s)$ *and* $FM = \langle \mathcal{F}, \mathcal{F}_M, \mathcal{F}_O, \mathcal{F}_{IOR}, \mathcal{F}_{XOR}, \mathcal{F}_{incl}, \mathcal{F}_{excl} \rangle$ *a feature model. A goal-feature mapping model is* $\Phi_{G-F} \subseteq FGM \times FM = \{\Phi_i(G_i, \mathcal{F}_i) : G_i \in (\mathcal{G}_g \cup \mathcal{G}_t)$ *and* $\mathcal{F}_i \subset \mathcal{F}\}$.

Develop feature model and process model mapping: The customizable process models describe the orchestration and choreography of activities in a family and the

feature model encapsulates configuration knowledge and enables the derivation of different customized processes of a customizable process model [106, 105, 131, 152]. Mappings link features of the feature model to the corresponding activities of the reference process model. We define the feature-process mapping as the following:

Definition 3.3 (Feature-Process Mapping Model) *Let* $PM = (\mathcal{V}, \mathcal{E}, \mathcal{F}, \mathcal{E}_A)$ *be a customizable process model defined by a structured process model and* $FM = (\mathcal{F}, \mathcal{F}_M, \mathcal{F}_O, \mathcal{F}_{IOR}, \mathcal{F}_{XOR}, \mathcal{F}_{incl}, \mathcal{F}_{excl})$ *be a feature model. A feature-process mapping model is* $\Phi_{F-P} \subseteq FM \times PM = \{\Phi_i(F_i, \mathcal{A}_i)\text{: } F_i \in \mathcal{F} \text{ and } \mathcal{A}_i \subseteq \mathcal{A} \subseteq \mathcal{V}\}$.

We consider many-to-many mappings by naming convention $\boldsymbol{\phi_i}(F_i, \mathcal{A}_i)$ where $\mathcal{A}_i$ is a subset of activities in a process model. If an activity is mapped into more than one features,the activity presents in the corresponding subsets $\mathcal{A}_i$ of all those features.

3.1.2 Customized Process Development

After developing the customizable process model, similar to software product line, a customized process model is derived from the customizable process model. In order to develop a customized process model, product line variability must be resolved which is referred as a configuration process. Hence, a configuration process is concerned with selecting appropriate variant(s) for each variation point by considering several factors including technical limitations, implementation costs, customers requests and expectations as well as constraints and dependency defined over a feature model. We proposed an approach for customizing the process model which consists of 1) developing application requirements and pre-configuring the feature model based on stakeholders' goals, 2) feature model configuration, and 3) process model customization. In the reminder of this section, we describe these phases.

Application Requirements Analysis

When deriving a product from the product line, both functional and non-functional requirements need to be taken into account. Functional and non-functional requirements do not have the same priority in all domains and situations. Customers may define preferences and constraints over functional and non-functional requirements in terms of different importance and/or limit on the maximum or minimum value for non-functional requirements. Therefore, we capture the customers' objectives and derive

their requirements according to their objectives. An application engineer communicates and understands the customers' needs and requirements by identifying their objectives. The family goal model is used as a reference model for communicating with the customers and capturing their goals. In many cases it would be impossible to fully satisfy all soft-goals. For example, a full satisfaction of performance, security, maintainability, and usability goals at the same-time is impossible. Therefore, the desire level of satisfaction for each soft-goal and the preferences between the soft-goals are elicited from stakeholders. Afterwards, by setting the satisfaction level of high-level goals and executing the backward reasoning algorithm [53], the leaves goals (plans) are selected. We adapted backward reasoning in a way that product line variabilities are taken into account. Next, a pre-configuration process is executed and features which are not based on the current stakeholders objectives are filtered out from feature model, by preserving the feature model constraints.

Feature Model Configuration

After the pre-configuration of the feature model, further specialization of feature model based on the preferences and requirements is performed. This is called configuration process which is performed by a stepwise specialization of the feature model. Specialization is known as a process where some configuration choices are eliminated in each stage of specialization [1]. For instance, customers select and/or deselect features of the feature model based on their preferences and business objectives. Several researchers in product line engineering have proposed techniques to automatically configure the feature model based on the requirements of stakeholders. Some techniques addressed the problem by transforming a feature model configuration into Constraints Satisfaction Problem (CSP), and using CSP solvers to identify optimal configuration [23, 148]. We proposed a decision making framework which adapts an existing decision making algorithm (Analytical Hierarchy Process (AHP)) and applies planning techniques to create a final configuration based on customers preferences and constraints. Figure 3.3 shows the configuration framework.

First, we capture the priority between non-functional properties through their relative importance relations. Relative importance is defined as follow [115].

Definition 3.4 ***(Relative Importance)***. *Relative importance between non-functional*

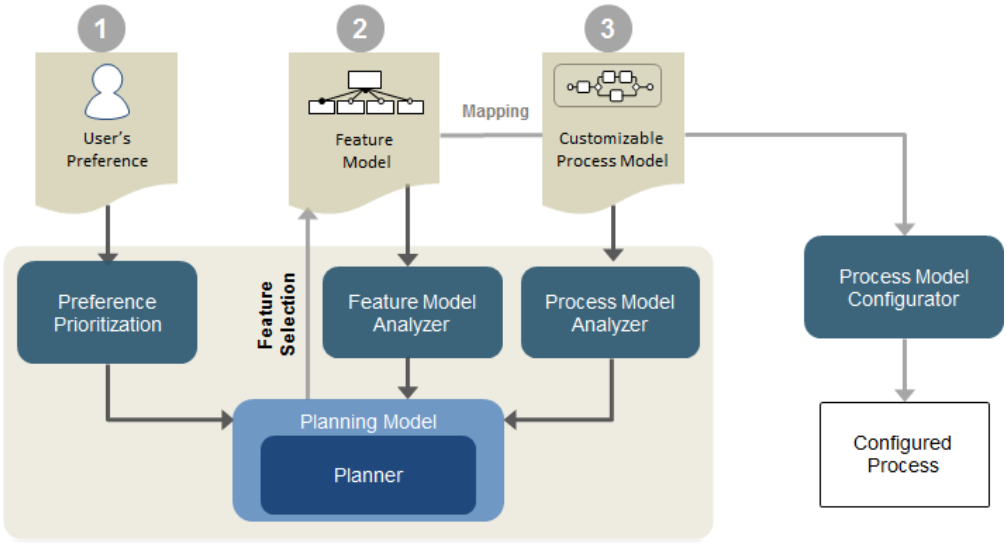


Figure 3.3: The proposed configuration framework

properties a and b is: $a \succ^{\alpha} b$ if non-functional property a is more important than non-functional property b with coefficient α.

Usually, the degree of importance of options (non-functional properties) is represented using values 1, 3, 5, 7, and 9 corresponding to equality, slight value, strong value, very strong and extreme value, respectively [139]. For example, relation *Security* $\succ^{3}$ *Performance* represents that *security* is slightly more important than *performance*. We refer to the relative importance between non-functional properties as stakeholders' preferences. Afterward, we propose an extended algorithm called Stratified Analytical Hierarchy Process(S-AHP) to compute the weight of non-functional properties and feature ranks. The S-AHP utilizes the AHP algorithm and a utility function to compute the rank of features.

A set of transformation rules convert the feature model, the computed features ranks, and constraints over non-functional properties into PDDL structure. Next, the existing planners are employed to find an optimal plan. According to actions in the optimal plan, the features in the feature model are selected. Finally, according to the established mappings between features and activities of the reference process model, activities mapped to the deselected features are removed. As a result, a configured process model is automatically generated.

Process Model Customization

In many cases, it is rare that the configured process model satisfies all the requirements of the target application [128]. Hence, customization steps further adapt and extend the configured process model according to the remaining requirements which are referred as to *delta requirements* [128]. This process includes adding, removing, and refining some activities and gateways into the model. During this step process engineers iteratively apply the the following change operations to the configured process model to derive a customized process model.

- INSERT: This operation adds a process element (i.e. an activity, gateway, or sub-process) into process model.
- DELETE: This operation removes a process element (i.e. an activity, gateway, or sub-process) from a process model.
- MOVE: This operation changes the execution order of activities in a process model.
- MODIFY: This operation changes the execution patterns of a process model. The MODIFY operation changes the work flow patterns and converts a pattern into another pattern.
- REPLACE: This operation replaces a process element, an activity, or subprocess with another activity or sub-process in a business process model.

According to [65] the above change operations are the most relevant operations needed for developing a customized process model from a customizable process model. Also these operations provide the required capabilities to handle the binding of different types of variability exist in the customizable process development.

3.1.3 Validation

Model validation is performed in both domain engineering and application engineering lifecycles. Inconsistency detection is based on mapping (traceability relations) between models (i.e. the family goal model, the feature model, and the customizable process model). In this section, we investigate the possible inconsistency patterns between these models and an approach for detecting these inconsistencies.

Goal Model and Process Model Inconsistencies

According to the mapping definition, tasks in the scope of each actor are mapped to activities inside pools and lanes. Therefore, the execution relations between activities (i.e., workflow patterns) in process models must be consistent with intentional relations between intentional elements in goal models. Mappings "carry" intentional relations to the process model. Thus, there might be inconsistencies between the relations given by the workflow pattern of an activity and the corresponding intentional relations of its mapped intentional element. We distinguish between two types of orchestration inconsistencies: 1) *strong goal-process inconsistency* and 2) *potential goal-process inconsistency* (cf. Definitions 3.5 and 3.6).

Definition 3.5 (Strong Goal-Process Inconsistency) *Assume a workflow pattern WF_A for an activity $A \in \mathcal{A}$ is specified over activities $A_1, \ldots, A_n \in \mathcal{A}$ and an intentional relation IR_G for $G \in \mathcal{G}$, defined over intentional elements $G_1, \ldots, G_m \in \mathcal{G}$. Activities $A, A_1, \ldots, A_n$ are realizations of intentional elements $G, G_1, \ldots, G_m$. A strong inconsistency between WF_A and IR_G occurs if there is no execution combination of activities that leads to the fulfillment of the intentional relation IR_G.*

A strong inconsistency says that for no execution that is expressed by workflow pattern/relation WF_A there is a goal fulfillment/achievement of IR_G possible. A weaker notion is the potential inconsistency. There can exist an allowed execution WF_A in which the corresponding intentional relation IR_G is not fulfilled.

Definition 3.6 (Potential Goal-Process Inconsistency) *Assume a workflow pattern WF_A for an activity $A \in \mathcal{A}$ is specified over activities $A_1, \ldots, A_n \in \mathcal{A}$ and an intentional relation IR_G for an intention G is defined over intentional elements $G_1, \ldots, G_m \in \mathcal{G}$. Activities $A, A_1, \ldots, A_n$ are realizations of intentional elements $G, G_1, \ldots, G_m$. We define a potential inconsistency between WF_A and IR_G if some execution combinations of activities lead to the fulfillment of the intentional relation IR_G and some execution combinations of activities do not lead to the fulfillment of IR_G.*

In the goal model, dependency links are utilized to represent the interactions between actors. Interactions between actors are realized in the corresponding process models in two ways. Firstly, if actors are mapped to pools, dependencies are realized through message exchange between pools, or more precisely between the processes in the different

pools. Secondly, in the case the actors are mapped to lanes, dependencies between intentions (of different actors) are realized through control flow relations [79, 75]. Therefore, there must exist a service interaction pattern between activities that are mapped to the intentional elements of a depender and activities that are mapped to the intentional elements of a dependee. Accordingly, there might be a choreography inconsistency in the process model with respect to actor dependencies. A choreography inconsistency is defined as the following:

Definition 3.7 (Choreography Inconsistency) *Let $FGM = (\mathcal{A}, \mathcal{G}, \mathcal{P}, \mathcal{D}, \mathcal{C})$ be a family goal model and $G_1 \in \mathcal{G}$, $G_2 \in \mathcal{G}$ be intentional elements, $Act_1, Act_2 \in \mathcal{A}$ actors and G_1 is in the scope of actor Act_1 ($G_1 \in Scope(Act_1)$) and goal G_2 is in the scope of actor Act_2 ($G_2 \in Scope(Act_2)$).*

Let $Ch = (\mathcal{P}, \mathcal{M})$ be a process choreography. Assume there are mappings between intentional elements G_1 and $A_1 \in activities(P_1)$ and between G_2 and $A_2 \in activities(P_2)$ ($P_1, P_2 \in \mathcal{P}$). Furthermore, there is a dependency relation $(G_1, G_x, G_2) \in \mathcal{P}$ (G_1 is the depender, G_2 is the dependee and G_x is the dependum).[1]

There is a choreography inconsistency if there is no service interaction patterns between activities A_1 and A_2 (i.e., $(A_1, A_2) \notin \mathcal{M}$ and $(A_2, A_1) \notin \mathcal{M}$).

In order to detect the aforementioned inconsistency patterns, we provide a transformation algorithms that convert the family goal model, customizable process model, and mapping relations into description logic formalisms. Additionally, we add a set validation concepts based on the subsumption test to identify different inconsistencies. The details can be found in [57].

Goal Model and Feature Model Inconsistencies

Considering a family goal model represents the objectives of stakeholders in the domain, the feature relations originate from intentional relations in the family goal model. This means that variability and commonality in feature models should be aligned with product line variability relations defined in the family goal models. Therefore, there would be an inconsistency in a family requirements model if intentional relations do not coincide with the feature relations.

[1] *From a logical point of view, a dependency relation depicts a relationship between actors (depender and dependee) or between intentions of their scope, but there is no relationship imposed to the dependum.*

Assume FR is a feature relation, with target feature F and source features $F_1, \ldots, F_n$, i.e., F depends on $F_1, \ldots, F_n$. Likewise, IR is an intentional relation with respect to product line variability (i.e., if annotated with VP, variability, otherwise commonality) with target element $G \in \mathcal{G}$ and source intentional elements $G_1, \ldots, G_m \in \mathcal{G}$. The fulfillment of G depends on the fulfillment of $G_1, \ldots, G_m$. The mapping provides the correspondence between child features and child intentional elements and between feature F and intentional element G. We define potential and strong goal-feature inconsistencies.

Definition 3.8 (Potential Goal-Feature Inconsistency) *A permissible satisfaction of the intentional element G, which depends on the satisfaction of $G_1, \ldots, G_m$, might lead to an incorrect configuration of feature F, while F depends on $F_1, \ldots, F_n$.*

Definition 3.9 (Strong Goal-Feature Inconsistency) *All permissible satisfactions of G, which depend on the satisfaction of $G_1, \ldots, G_m$ with respect to IR, lead to an incorrect configuration of feature F (F depends on $F_1, \ldots, F_n$).*

In order to detect the aforementioned inconsistency patterns, we provide a transformation algorithms that convert the family goal model, feature model, and mapping relations into description logic formalisms. While transforming the family goal model into description logic, if an intentional relation is annotated with VP, the relation is converted into OR relation. If relation is not annotated with variability notations, the relation converted into AND relation. Finally, a set validation concepts based on the subsumption test are added to identify different inconsistencies. The details can be found in [14].

Customization Inconsistencies

Also, during developing a customized process model, the model must preserve the behavioral relations (business logic) specified by a customizable process model and the variability (configuration) relations expressed by a feature model. In order to investigate the behavioral inconsistencies, we use trace semantics of processes. The trace semantics describe a process in terms of all possible executions (traces). A trace is a sequence of activities. A customized process model should also be valid with respect to the configuration relationships that are imposed by the feature model. These relationships include hierarchical relations (i.e., Mandatory, OR, Alternative, and Optional

Relation) and integrity constraints (Include, exclude, and corequiste). The configuration step does not cause inconsistencies in the derived process model (i.e., configured process model), because we perform configuration of the feature model and then according to the mapping between features and activities, the configured process model is automatically generated. Moreover, several approaches and algorithms exist which ensure validity of a feature model configuration. However, during customization step when a process engineer applies change operations, configuration constraints might be violated.

In order to identify the possible inconsistency patterns, first we define an allowable execution combination set as the following:

Definition 3.10 (Allowable Execution Combination Set) *Let WR be a workflow relation defined over activities* $A_1, \ldots, A_n$*. The allowable execution combinations is the set of all execution combinations of activities* $A_1, \ldots, A_n$ *(Shown as* $\Gamma(WR)$*) that do not violate constraints defined by the workflow relation WR.*

According to our analysis based on industrial test cases, two types of behavioral inconsistency may occur in a customized process model with respect to a reference process model: 1) *strong behavioral inconsistency* and 2) *potential behavioral inconsistency.*

Definition 3.11 (Strong Behavioral Inconsistency) *Assume there is a mapping between activities of workflows WR and* WR'*. A strong behavioral inconsistency between WR and* WR' *occurs if there is no intersection between the allowable execution combination set of WR and the allowable execution combination set of* WR'*, i.e.,* $\Gamma(WR) \bigcap \Gamma(WR') = \emptyset$

Definition 3.12 (Potential Behavioral Inconsistency) *Assume there is a mapping between activities of workflows WR and* WR'*. A potential behavioral inconsistency between WR and* WR' *occurs if the allowable execution combination set of WR is a subset of the allowable execution combination set of* WR'*, but not equal to* WR'*, i.e.,* $\Gamma(WR) \subset \Gamma(WR')$ *and* $\Gamma(WR') \neq \Gamma(WR)$*.*

In addition to behavioral inconsistencies, customization may cause configuration inconsistencies where the configuration constraints formulated in feature model is violated. Schobbens et al. [145] and Gue et al. [60] defined well-formedness rules for feature models using Free Feature Diagrams (FFD). Among the well-formedness rules, we adopt those

rules which are relevant for the configuration of feature models: 1) there is a unique feature root that is a direct or indirect parent of all features in the feature model; and 2) all features, except the root feature, have one parent feature. Considering these two well-formedness rules, similar to Gue et al. [60], we define the following concepts in the feature models:

Definition 3.13 (Mandatory Path) *A mandatory path for a feature F is a path between the feature F and the root feature in a feature model in which each intermediate node (feature) is either a mandatory feature or the sole child in an Alternative- or Or-group.*

Definition 3.14 (Inclusion Chain) *An inclusion chain for feature F is a chain from start feature F where all features are connected by the include relations to each other.*

Mappings describe a concrete implementation of a feature by one or more activities. As the customization might add and remove activities, the selection and deselection of features in the feature model specialization changes correspondingly. In order to ensure both well-formedness rules for feature models and their specializations after the customization operations, we check configuration inconsistencies in the feature model specialization. We distinguish between *hierarchical inconsistency* (Definition 3.15) and *integrity inconsistency* (Definition 3.16). Both definitions rely on the mandatory path and inclusion chain that are given by integrity constraints, according to Definitions 3.13 and 3.14.

In these two definitions a *deselected feature* refer to a feature whose corresponding activity (the activity mapped to the feature) does not exist in the customized process model. A *selected feature* is a feature whose corresponding activity exist in the customized process model.

Definition 3.15 (Hierarchical Inconsistency) *A deselected feature F is hierarchically inconsistent if either F: a) is mandatory and there is a mandatory path from the feature F to the root feature; or b) is an OR- or alternative feature and there is a mandatory path from the feature F to the root feature and all feature with the same direct parent with the feature F are deselected.*

Definition 3.16 (Integrity Inconsistency) *There is an integrity inconsistency in a selected feature F if: a) there is a deselected feature in the inclusion chain of the feature F; or b) there is a selected feature which is excluded by feature F.*

In order to detect the aforementioned inconsistency patterns, we provide a transformation algorithms that convert the feature model, customizable process model, customized process model, and mapping relations into description logic formalisms. Additionally, we add a set validation concepts based on the subsumption test to identify different inconsistencies. The details can be found in [13].

3.2 Publications and Discussion

In this section, I point out the publications that were created during this research endeavor and relates them to the overview of the approach.

3.2.1 Publications

This research leads to sixteen papers published in different software engineering journals, conferences, and workshops. I divide the publications into two subgroups: the former group are the papers which directly used in the following chapters of thesis; and second group are the papers which developed as a result of the research, but due to space limitation we decided to not include in the thesis.

Publications as Thesis Chapters

- *(P1)* **Asadi, M.**, Gasevic, Wand, Y., Hatala, M. (2012) Deriving Variability Patterns in Software Product Lines by Ontological Considerations. *In Proc. Of the 31st International Conference on Conceptual Modeling*, LNCS 7532, Italy: 397-408.

- *(P2)* **Asadi, M.**, Bagheri, E., Gasevic, D., Hatala, M. (2011). Goal-driven software product line engineering. In Proc. Of the 26th ACM Symposium on Applied Computing (SAC 2011), Taiwan: 691-698.

- *(P3)* **Asadi, M.**, Gasevic, D., Hatala, M., Bagheri, E., (2014). A Planning-Based Approach for the Configuration of Process Models. to be Submitted to Automated Software Engineering (ASE) Journal.

- *(P4)* **Asadi, M.**, Groner, G., Mobabbati, B., Gasevic, D. (2014) Validation of User Intentions in Feature-Oriented Software Families. Journal of Software Systems and Modeling (SOSYM), 20 pages, in press.

- *(P5)* Groner, G., **Asadi, M.**, Mobabbati, B., Gasevic, D., Boskovic, M. (2014) Validation of User Intentions in Process Orchestration and Choreography, Information Systems Journal, Vol. 43, No 7, pp. 83-99.

- *(P6)* **Asadi, M.**, Mobabbati, B., Groner, G., Gasevic, D. (2014) Validation of Customization of Reference Process Models. Journal of Systems and Software (JSS), Vol. 96, No. 10, pp. 73-92.

Other Publications

- *(P7)* **Asadi, M.**, Soltani, S., Gasevic, D., Hatala, M. The Effects of Visualization and Interaction Techniques on Feature Model Configuration. Journal of Empirical Software Engineering, (in press).

- *(P8)* **Asadi, M.**, Soltani, S., Gasevic, D., Hatala, M., Bagheri, E. (2014) Toward Automated Feature Model Configuration with Optimizing Non-Functional Requirements. Information and Software Technology Journal, pp. 22, in press.

- *(P9)* Mohabbati, B., **Asadi, M.**, Gasevic, D., Hatala, M., Mller, H. (2013) Combining Service-Orientation and Software Product Line Engineering: A Systematic Mapping Study, Information and Software Technology Journal, Vol. 55, No. 11: 18451859.

- *(P10)* Mohabbati, B., **Asadi, M.**, Gasevic, D., Lee, J. (2014) Software Product Line Engineering to Develop Variant-rich Service-Oriented Applications. Springers Web Services Handbook: 535-562.

- *(P11)* Soltani, S., **Asadi, M.**, Gasevic, D., Hatala, M., Bagheri, E. (2012) Automated Planning for Feature Model Configuration based on Functional and Non-Functional Requirements. In Proc. Of the 16th International Software Product Line Conference, Brazil: 56-65.

- *(P12)* Groner, G., **Asadi, M.**, Mobabbati, B., Gasevic, D., Bokovi, M., Silva Parreiras, F.(2012) Validation of User Intentions in Process Models. In Proc. Of the 24th International Conference on Advanced Information Systems Engineering, Poland: 366-381.

- *(P13)* Bagheri, E., **Asadi, M.**, Gasevic, D., Soltani, S. (2011) Stratified Analytic Hierarchy Process: Prioritization and Selection of Software Features. In Proc. Of the 14th International Software Product Lines Conference,South Korea:300-315.

- *(P14)* Bagheri, E., **Asadi, M.**, Ensan, F., Gasevic, D., Mohabbati, B. (2011). Bringing Semantics to Feature Models with SAFMDL. Proc. Of the 2011 Conference of the Centre for Advanced Studies on Collaborative Research (CASCON 2011), Canada, 2011: 287-300.

- *(P15)* **Asadi, M.**, Bagheri, E., Mohabbati, M., Gasevic, D.(2012) Requirements Engineering In Feature Oriented Software Product Lines: An Initial Analytical Study, In Proceedings of the 1st International Workshop on Requirements Engineering Practices on Software Product Line Engineering (REPOS 2012) at SPLC2012, Brazil: 36-44.

- *(P16)* **Asadi, M.**, Mohabbati, B., Kaviani, N., Gasevic, D. Boskovic, M. Hatala, M. (2009). Model-Driven Development of Families of Service-Oriented Architectures. In Proceeding of The First International Workshop on Feature-Oriented Software Development co-located with MODELS 2009, USA: 95-102.

3.2.2 Discussion

Following the research method described in the introduction chapter, first I developed the initial ideas on the notion of variability management and decision making in the process customization which leads to initial method (*P16*) and Stratified Analytical Hierarchy Process (*P13*) technique. Next, I conducted through analysis of variability in product lines domain from both theoretical and practical point of views which resulted theoretical foundation for variability analysis (*P1*) and mapping study (*P9*). The result of analysis helped me in deciding on variability strategy and language (i.e., feature oriented technique), and required extensions on the goal model *(P5)* and process models *(P6)* to support variability.

Simultaneously, I investigated the notion of variability from requirements perspective *(P15)* and proposed initial ideas on requirements driven configuration of variability *(P2)*. Next, further effort has been done to provide a comprehensive configuration technique which resulted in automated techniques for configuration *(P2, P8)*. After developing configuration tool, my colleague and I tried to apply the visualization and interaction

techniques for the further improvement of configuration and conducted an empirical study which showed the usefulness of the proposed visualization and interaction interventions *(P7)*.

The other component of my research concentrated on developing mapping models and validation aspects in our approach. To this end, I analyzed different artifacts in the approach (i.e., family goal model, feature model, and customizable process model) and developed mapping models. We identified different types of inconsistencies and developed proper algorithms for detecting these inconsistency types. The results were published in *(P4, P5, P6, P12)*. Finally, I made an effort to describe the initial holistic view of the proposed method in *(P10)*.

Table 3.1 shows the overall proposed approach and the publication contributions to each part of the proposed method. Also we highlighted the main papers which contribute to each section.

Table 3.1: Relation between publications and different phases of the proposed method.

Lifecycle	**Phase**	**Publication**
Domain Engineering	Family Goal Model Development	***P2***, ***P4***, *P5, P10, P12, P15*
	Customizable Process Development	***P5***, *P9, P10, P12, P14*
	Feature Model Development	***P1***, ***P3***, *P4, P7, P9, P10, P11, P15*
	Developing Mapping Model	***P4***, ***P5***, ***P6***, *P9, P10, P12*
Process Customization	Preconfiguration	***P2***, *P10, P15*
	Configuration	***P3***, *P7, P8, P9, P10, P11, P13*
	Customization	***P6***, *P9, P10*
Validation	Goal-process Validation	***P5***, *P9, P10*
	Goal-Feature Validation	***P4***, *P9, P10*
	Customized Process Validation	***P6***, *P9, P10*

Chapter 4

Theoretical Analysis of Variability

This chapter consists of "Deriving Variability Patterns in Software Product Lines by Ontological Considerations" paper published in *proceedings of the 31st international conference on conceptual modeling*.[1] The paper presents the results of theoretical analysis of the variability in the context of product lines. It utilizes the ontological theory to define different variability patterns between two information systems. Next, an ontological framework is developed which is used to evaluate variability modeling languages completeness and clarity.

Author's role - I performed the analysis of the variability and developed variability patterns. Also, I devised the ontological evaluation framework and evaluated feature models and the Orthogonal Variability Modeling (OVM) language using the developed framework. I wrote the paper and it was revised by my co-authors.

[1]The paper reprinted (with minor adjustment to formatting), with permission from Springer Press.

Asadi, M., Gasevic, Wand, Y., Hatala, M. (2012) Deriving Variability Patterns in Software Product Lines by Ontological Considerations. *In Proceedings of the 31st International Conference on Conceptual Modeling*, LNCS 7532, Italy: 397-408.

4.1 Abstract

Variability modeling is widely used in software product line engineering to support reusability. Specifically, it is used in the derivation of concrete software products from a reusable solution within a family of products. To help manage variability, several modeling languages have been proposed for representing variability within a family of products. The study and evaluation of languages to model variability has so far focused on practical aspects of such languages. Less attention has been paid to more theoretical approaches to the analysis of variability modeling languages. In developing such approaches it would be of particular interest to explore the ability of variability modeling to represent the information about the real world (application) domain for which the product family is designed. In information systems research, evaluation of expressiveness of conceptual modeling languages has been done based on ontological theories. This paper describes a framework for general analysis of types of variability based on Bunges ontology and derives a variability framework which is used to evaluate variability modeling languages.

4.2 Introduction

Software Product Line Engineering (SPLE) is an approach to developing a set of software systems which satisfy requirements of a specific domain and share common features [73]. The key factor for the success of SPLE is variability which can be categorized into [126]: *essential variability* (i.e., variability from the system usage perspective) and *technical variability* (i.e., variability related to the realization/implementation of essential variability and/or variability related to the IT-infrastructure) [126]. Essential variability is variability in domain knowledge and can be represented by *conceptual models*. Conceptual models are representations of static (e.g., things and their properties) and dynamic (e.g., process and events) phenomena in a domain of interest [176]. Variability languages should be able to represent both essential variability (manifested in the conceptual models) and technical variability (manifested in the implementation

models). Several variability languages have been proposed which use approaches such as feature modeling [73] and orthogonal variability modeling [125]. The evaluation and improvement of expressiveness of variability languages have attracted some attention in SPLE research [67, 42]. However, theoretical analysis of variability modeling languages with respect to their ability to represent variability in real world domain models has yet not been studied in detail. Theoretical analysis of variability languages help in better understanding of expressiveness of these languages and consequently changes in these languages for proper representation of variability between conceptual models

In this paper, we investigate the use of ontological theories for theoretical analysis of variability modeling languages. An ontological theory defines constructs required for describing the structure and processes of the world in general. Ontological theories have been used to evaluate modeling languages in terms of correspondence of ontological concepts to modeling constructs. Bunge's ontology (as adapted by Wand and Weber [176]) has been used to evaluate several conceptual modeling languages [176, 47]. This ontology includes a set of high level constructs for representing real world phenomena. The evaluation of modeling languages is based on the assumption that an information system is an artifact that represents a real-world domain. Since Bunge's ontology provides concepts for representing real world phenomena, we presume that it is appropriate for analyzing the essential variability of software systems which represents variability of real-world domains. Therefore, in this paper, we analyze essential variability of products in a software product line by investigating variability between real-world domains they are intended to represent. More specifically, we employ concepts from Bunge's ontology (as adapted for information systems [176]) as the theoretical framework for identifying possible variability patterns among products based on real-world domains variability. Afterward, we develop a theoretical framework for variability and apply the framework for evaluating expressiveness of variability languages.

4.3 Background

4.3.1 Variability in Software Product Line Engineering

Variability is the central concept in SPLE [126] and have been investigated with different perspectives and assumptions. Mainly, existing points of views on the variability notion can be categorized into two classes:

- Bosch et al. [63] defines variability as "the ability to be changed, customized, configured, or extended for use in a specific context".
- Weiss et al. [181] define variability as "an assumption about how members of a family may differ from each other".

Mentzger et al. [102] refer to software variability as the first class of variability and product line variability as the second class. Product line variability differs from software variability; product line variability is an explicit decision of product management about what should vary between the systems in a product line and what should not [126]; software variability is an inherent property of the software under development and represents different behavior of the software. Product line variability originates from differences among real-world domains which are represented by the products of a product line. For example, in a shopping domain, real-world shopping systems may vary in their types of payment methods where one may include *credit card* and *debit card* and another may have *debit card* and *cash*. This variation leads to product line variability for their corresponding information systems. But, in a shopping domain, if all shopping systems have all three types of payment methods and hence, the payment methods cannot be varied among different members of the product line; that is, there may be software variability but there is no product line variability.

Since software products are representation of real-world domains, one can analyze the sources and types of variability which exist among real-world domains by using an ontological theory. In this paper, we concentrate on the ontological analysis of essential variability to identify variability patterns that can exist among different products.

4.3.2 An Illustrative Example

To exemplify the concepts in the remainder of the paper, we use a standard case study commonly used in comparative analyses of information system methodologies [117]. This is the IFIP working conference case study ([117] pp. 8-9). We assume three conference examples named conference A, conference B, and conference C, and we analyze their variability. Figure 4.1 shows these three conferences.

	Conference A	Conference B	Conference C
Things	Specific instances of Program Committee, Organizing Committee, Participants, Papers	Specific instances of Program Committee, Organizing Committee, People Involved, Papers, Demos	Specific instances of Program Committee, Organizing Committee, Attendees, Papers, Art Works,
Properties (attributes)	Paper (Submission ID, Title, Author(s) Name, Quality, Type, Status), Authors(Name, Papers(s), Affiliation, Role)	Paper (Paper ID, Title, Author(s) Name, Quality, Category, Status), Authors (AName, Paper(s), Affiliation, Role, Conflict)	Paper (Paper ID, Title, Author(s) Name, Quality, Category, Status), Authors(AName, Papers(s), Affiliation, Role)
Lawful State Spaces	Paper-Status (Submitted, Accepted, Rejected, Short Papers) Paper-type (Experience, Research, Evaluation. Ideas)	Paper-Status (Submitted, Accepted, Rejected, Conditionally Accepted) Paper Category (Theoretical Foundation)	Paper-Status (submitted, Accepted, Rejected, Conditionally Accepted) Paper-Category (Theory, Practice)
Lawful Events	Paper {(Submitted→Accepted) (Submitted → Rejected), (Submitted →Short Paper Accepted)	Paper {(Submitted→Accepted) (Submitted → Rejected), (Submitted → Conditionally Accepted), (Conditionally Accepted→Accepted), (Conditionally Accepted →Rejected)},	Paper {(Submitted→Accepted) (Submitted → Rejected), (Submitted → Conditionally Accepted), (Conditionally Accepted→Accepted), (Conditionally Accepted →Rejected) }

Figure 4.1: Three imaginary conferences based on the IFIP working conference example

4.4 Ontological Analysis of Essential Variability

According to the representation premise of Bunge's ontology, we can conclude that every product (i.e. information system) in a product line family is a representation of a real domain. For example, a software system developed for managing conference A is a representation of conference A in the real-world. Thus, the software system's conceptual model represents static and dynamic phenomena of the real-world domain of the system under study (i.e., conference A). As already indicated, essential variability represents variability among knowledge of real-world domains, and can be represented by different conceptual models of the products. Hence, we investigate essential variability by exploring variability of domains which these products are intended to represent. To investigate variability between different real world domains corresponding to different products, we assume that we can map the phenomena in one domain to the phenomena in another domain.

Mapping Premise: We assume that we can establish corresponding mappings between *things, attributes, states,* and *events* in one domain and *things, attributes, states,*

and *events* in another domain.

For example, for the IFIP conference context, we can establish mappings between property *Submission ID* of thing *Paper* $\in$ *Conference A* with property *Paper ID* of thing *Paper* $\in$ *Conference B*. The mapping is denoted as *A(Paper.Submission ID)* $\longleftrightarrow$ *B(Paper.Paper ID)*.

To investigate variability among different domains, first we analyze similarity patterns among things in these domains. Next, we consider variability as opposite to similarity and define variability patterns between different real-world systems

4.4.1 Variability Patterns among Sets of Phenomena

Before introducing variability classes in terms of Bunge's concepts (i.e., thing, property, state, and event), we introduce a set of general variability patterns between two sets of phenomena (By phenomena we refer to any possible observation that can be made about the domain or part of it). Afterwards, considering these variability patterns and Bunge's concepts, a set of variability classes among two domains are defined.

Assume $S = \{s_1, s_2, , s_m\}$ is a set of phenomena belonging to domain D_1 and $T = \{t'_1, t'_2, , t'_n\}$ is a set of phenomena belonging to domain D_2. Now, we have one of following situations with respect to similarity between these two sets.

Definition 1 (Equivalent Sets of Phenomena): S is equivalent to T (denoted as $S \equiv T$), if and only if there is a mapping between elements in S and elements in T.

Definition 2 (Similar Sets of Phenomena): S is similar to T with respect to p (denoted as $S \cong_p T$) if and only if there is a subset of S (i.e., $S' \subset S$) and of T (i.e., $T' \subset T$) which are equivalent $S' \equiv T'$. p is equivalent subset i.e. $p = S' = T'$.

Definition 3 (Completely Dissimilar Set of Phenomena): S is completely dissimilar to T with respect to (denoted as $S \neq T$) if and only if there are no subsets of S(i.e., $S' \subset S$) and of T (i.e. $T' \subset T$) that are equivalent.

Based on Definitions 1-3, we can define the following similarity patterns between two different sets of phenomena:

- *Full similarity double side* - when a set of phenomena S and set of phenomena T are equivalent (i.e., $S \equiv T$).

- *Full similarity one side* - when a set of phenomena S and a set of phenomena T are similar (i.e.,$S \cong_p T$) and when we have either $S' \subset S$ and $S' \equiv T$ or $T' \subset T$

and $S \equiv T'$.

- *Partial similarity* - when a set of phenomena S and a set of phenomena T are similar (i.e., $S \cong_p T$) and there is no subset of one functional schema that is equivalent to the other functional schema.
- *Complete Dissimilarity* - when two sets of phenomena are completely dissimilar.

One of the above similarity patterns happens between attribute sets of phenomena in different real-world domains. All the above patterns, except *full similarity double side*, represent possible variability between two sets of phenomena. To investigate variability between more than two sets which belong to different real-world domains, we can explore the variability patterns between each set and the rest of the sets.

4.4.2 Variability among Things of Different Real-World Domains

According to Bunge's ontology, a real-world domain is comprised of things. To investigate variability between domains, we need to explore variability among things in the product domains in terms of the variability of their structure and processes. The structure and processes of things is defined in terms of a combination of their properties, states, laws and events. We analyze variability among things by analyzing variability among their *attributes (properties representation), lawful state space* and *lawful event space*. To investigate variability among two things in terms of their attributes, we form functional schemas which are sets of attributes for defining those things. Figure 4.2 shows different similarity classes of two things in different real-world domains.

For instance for IFIP conferences, we have a full similarity double side among functional schema defining paper in conference A and functional schema defining paper in conference C. Moreover, there is a subset of the functional schema defining authors of conference B which is equivalent to the functional schema describing authors of conference A. Hence, there is a full similarity one side between the functional schemas of the authors in the two conferences.

4.4.3 Variability among Real-World Domains

After exploring variability among *things* of different product domains, we can define variability classes between two *product domains* belonging to a product line. Variability

among product domains may occur in both static (structure) and dynamics (processes) of the domains. The structure of a domain is defined based on things in the domain. Considering Bunge's ontology, the structure of a domain can be shown using functional schemas and a lawful state space of the whole domain. Hence, to investigate structural variability between two domains, we need to consider combinations of variability classes defined for functional schemas and lawful state spaces of these two domains. A difference between the functional schemas implies a difference between the conceivable state spaces and lawful state spaces of the two product domains. Therefore, variability between functional schemas of the domains leads to the variability of the state spaces of the domains. However, an equivalence of the two functional schemas means the equivalence of their conceivable state space, but does not mean an equivalence of their lawful states because different laws may govern their properties. Hence, there may be structural variability between domains in terms of their lawful state spaces, even though there is a full-similarity double side between their functional schemas. For instance for IFIP conferences, we have a full similarity double side among the *functional schema* defining paper of *conference A* and the *functional schema* defining the paper in the *conference C*, but possible values for *status* of the paper in conference A are *Submitted, Accepted, Rejected, Short Papers* and in the conference C are *Submitted, Accepted, Rejected, Conditionally Accepted*. This shows variability in the lawful state spaces, even though their functional schemas are completely similar.

On the other hand, when considering dynamics of the domain a process can be defined in terms of a *sequence of changes* (i.e. events) [176]. These changes may happen within things (internal events of the things) and between things (i.e. interactions between things) of a domain. The processes of a domain can be shown using the *lawful event space* of the domain and the *ordering* between these events. Hence, to investigate variability between processes of different product domains, we need to explore variability classes between their lawful event spaces of the domains and their sequences. As an example for variability in ordering of events, assume that determining the program of conference A involves an order *Workshop* $\rightarrow$ *Tutorial* $\rightarrow$ *Main Conference* and for conference B *Main Conference* $\rightarrow$ *Workshop* $\rightarrow$ *Tutorial*. Although both the conferences contain the same set of events, the ordering of events is different.

Figure 4.2 shows similarity classes between two different product domains. All of the above similarity classes, except full similarity double side, show variability classes among

Patterns	Class name	Description
Full similarity	*Full similarity among functional schemas*	Equivalent functional schemas
	Full similarity among lawful state spaces	Equivalent lawful state spaces
	Full similarity among lawful event spaces	Equivalent event spaces
Full similarity one side	*Full similarity one side among functional schemas*	Similar functional schemas and a subset of one functional schema is equivalent of the other functional schema
	Full similarity one side among lawful state spaces	Similar lawful state spaces and a subset of one lawful state space is equivalent of the other lawful state space.
	Full similarity one side among lawful event spaces	Similar lawful event spaces and a subset of one of the lawful event spaces is equivalent of the other lawful event space.
Partial similarity	*Partial similarity among functional schemas*	Similar functional schema and there is no subset of one functional schema that is equivalent to the other one.
	Partial similarity among lawful state spaces	Similar lawful state spaces and no subset of one lawful state space that is equivalent to the other lawful state space.
	Partial similarity among lawful event spaces	Similar lawful event space and there is no subset of one lawful event space that is equivalent to the other event space.
Complete dissimilarity	*Complete dissimilarity among functional schemas*	Dissimilar functional schemas
	Complete dissimilarity among lawful state spaces	Dissimilar lawful state space
	Complete dissimilarity among lawful event spaces	Dissimilar Lawful event space

Figure 4.2: Similarity classes between things of different product domains

domains. When investigating more than two domains, combination of these classes may exist between one product domain and the rest of product domains.

4.4.4 Variability Framework Derived From Bunge's Ontology

Having identified variability classes in section 4.4.3, we describe a framework for evaluating variability languages by using the ontological theory. The evaluation framework is based on the assumption that variability languages must be ontologically expressive and must be able to represent all the variability classes which may happen among the elements of conceptual models, represented in figure 4.3.

By investigating the variability classes in section 4.4.3, we specify two main concepts for variability framework: *variability sources* and *variability patterns*. A *variability source* shows elements in which variability may happen. A *variability pattern* shows a different recurring type of variability between sets of phenomena of different product domains. Considering Bunge's ontology, *structure of a domain* including *things, properties (attributes),* and *lawful state space* and processes of the domain including lawful event space and time of occurrence are variability sources. The common variability patterns for both the structure and process of domains are *full-similarity one side, partial similarity*, and *complete dissimilarity*. Additionally, the *ordering* variability pattern is

Similarity among structures of Domains			Similarity among Processes of Domains		
Functional Schema	**Lawful State Space**	**Class Name**	**Lawful Event Space**	**Sequence Difference**	**Class Name**
Full Similarity – Double Side	Full Similarity - Double side	Completely similar structures	Full Similarity – Double Side	No	Full Similarity-Double side among Process
	Full Similarity – One Side	Complete similar macro structure, different micro structure		Yes	Full Similarity-Double side among events and different order
	Partial Similarity		Full Similarity –One Side	No	Full Similarity –one Side among Process
	Dissimilarity	Complete similar macro structure and complete dissimilar micro structure		Yes	Full Similarity –one side among events and different order
Full Similarity – One Side	Full Similarity – one side	High similar macro and micro structure	Partial Similarity	No	Partial Similarity among Process
	Partial Similarity			Yes	Partial Similarity among Process and different order
	Dissimilarity	High similar macro and complete dissimilar micro structures	Dissimilarity	NA	Complete Dissimilarity among process
Partial Similarity	Partial Similarity	Medium similar macro and micro structure			
	Dissimilarity	Medium similar macro and complete dissimilar micro structure			
Dissimilarity	Dissimilarity	Complete Dissimilar Structure			

Figure 4.3: Similarity Classes between two product domains

dedicated to the processes of domain and shows another aspect of difference between the sets of lawful event spaces.

Similar to ontological analysis for conceptual modeling languages [176], we identify two evaluation criteria for assessing the capability of languages to model variability - variability completeness and variability clarity. Variability completeness is concerned with investigating if modeling languages have constructs for representing all the variability patterns and consider variability in all possible sources. Variability clarity means that there is a one-to-one mapping among variability constructs in variability languages and variability patterns of the framework.

4.5 Analysis of Variability Languages with Variability Framework

In this section, we analyze two variability languages, i.e. feature models [73] and Orthogonal Variability Models (OVMs) [126] using the proposed variability framework.

Concepts in the Framework \ Variability Languages			Feature models		OVM	
				Explanation		Explanation
Variability Source	Structure	Things	√	relating features to natural kind in Bunge's Ontology	√	relating variation point (subject) and variants (object) to natural kind in Bunge's Ontology
		Properties	√		√	
		Lawful state Space	√		√	
	Process	Event Space	√		√	
		Time	√		√	
Variability Patterns	Full Similarity One-side		√	Optional relations	√	Optional relations
	Partial Similarity		±	OR relation	±	Cardinality [m..n]
	Dissimilarity		√	Alternative Relation	√	Cardinality [1..1]
	Ordering Variability		×	Not Considered	×	Not Considered

Figure 4.4: Analysis results of feature models and OVM using variability framework (✓) match, (×) not match, (±) ambiguity

Feature models are widely employed in SPLE to model variability and provide representations for variability relations. A central notion in identifying and modeling variability in feature-oriented software product lines is feature, which is defined as follows:

"Important distinguishing aspects, qualities, or characteristics of a family of systems" (Kang et al. [73]); and "a logical unit of behavior specified by a set of functional and non-functional requirements" Bosch [27]. In the feature model, variability is represented through the following variability relations: *Optional feature, Alternative feature group,* and *Or feature group.* OVMs represent variability using the variation points (VP) and variants (V) constructs [126]. A variation point is a representation of variability *subject*, "a variable item of the real-world or a variable property of such item" [126]. A variant is a representation of a variability *object*, a particular instance of a variability subject [126]. For example, a variable subject color is a property of a real-world item (e.g. Car) and variable objects for *color* are *green, red,* and *blue.* Then, color is a variation point and green, red, and blue are variants. In OVM, variability is specified using relations defined between variation points and variants. These relations are *optional* and *alternative choice* with cardinality *min..max* [102].

In order to analyze these two languages, based on the evaluation criteria defined in our framework, we derive a research question: *What are the representational shortcomings of feature models and OVM in light of the theoretical variability framework?*

To answer the research question, we established a mapping between constructs of these languages and the concepts in the variability framework (see Figure 4.4).

With respect to completeness criteria, the variability languages should encompass constructs for presenting variability sources and variability patterns. Hence, we investigate if feature models and OVM can represent variability among all different sources of variability (i.e., things, properties, lawful state space, and lawful event space).

Variability sources: Variability in feature models is represented in terms of difference among features and in OVM in terms of variation points and variants. According to definitions for features, we can conclude a feature is a particular set of properties or processes of one or more products in a product family. To interpret features based on Bunge's ontology, we can relate features to *natural kinds* because natural kinds are used to define things with a set of common properties that adhere to the same laws including both transition and state laws [47]. Hence, the natural kinds similar to features can be used to represent both the processes (lawful event spaces and time) and structure (things, properties, and lawful state space) of the domain. Considering the ways features and natural kinds can be related, we propose that a good set of features is required to represent all sources of conceptual variability including things, properties, lawful state space, and lawful event space. Similarly in OVM, a variation point and a variant can represent both processes (lawful event spaces and time) and structure (things, properties, and lawful state spaces) of the real-world domain. Therefore, the variation point and variant can be related to natural kinds. This means that OVM can represents all sources of conceptual variability.

Variability patterns: To interpret variability relations in feature models and OVM, using our variability patterns, for simplicity we consider a product line with three products (e.g., conferences A, B, and C). However, the overall discussion and interpretation is applicable for product lines with any number of products. We make two assumptions. First, if a product has a feature (in feature models) or a variant (in OVM) *f*, then there are things in the real-world domain of the product belonging to natural kind nk_f. Second, we assume that all products involved in a variability relation, contain thing(s) in their corresponding real-world domain which are mapped to thing(s) in the real-world domain of the other involved products.

Optional relation: If a feature (variant in the OVM) f is optional then one or two of the products in the example product line contain *f*. Consequently, things in the real-world domain of products containing feature (variant in OVM) f belong to natural kind nk_f. However, things in the real-world domains of the other products (i.e., products

without optional feature or variant f) do not have a set of properties and processes defined by nk_f. This means there is a full similarity one side with respect to nk_f among things which have mapping to the real-world domains of the products.

Alternative relation (alternative choice with cardinality 1..1): In an alternative relation (i.e. XOR group f1, f2, f3), from a group of alternative features (variants in the OVM), each product contains only one of the features (variants in the OVM). This means that there are things in the domain of products involved in the alternative group such that these things belong to natural kinds nk_{f1}, nk_{f2} and nk_{f3}. The mapped things of different products involved in the alternative relation are completely dissimilar with respect to the properties and laws defined in nk_{f1}, nk_{f2},and nk_{f3}.

OR relation (alternative choice with cardinality min..max): A group of features connected by the OR relation means that products in the group have one or more features. Similarly, in OVM, a group of variants with cardinality *min..max* refers the selection of at least *min* and at most *max* number of variants. To interpret these patterns, all the variability patterns, including complete dissimilarity, partial similarity, and full-similarity one side may happen (but do not have to) among mapped things of each pair of products. Therefore, an OR group in feature models and alternative choice with cardinality *min..max* in OVM can be mapped into complete dissimilarity, partial similarity, and full-similarity one side variability patterns.

Ordering variability: Neither feature models nor OVM has constructs for representing ordering variability between features and variants, respectively. Consequently, feature models and OVM cannot represent part of process variability classes shown in figure 4.3.

Based on our analysis, we conclude feature models and OVM have the same representational expressiveness for modeling conceptual variability. Also, both languages have the lack of variability completeness as they do not have any construct for representing ordering variability. Finally, due to the ambiguity in OR and alternative choice with cardinality *min..max*, we cannot establish oneto-one mapping between variability patterns in our framework and these *variability* relations. This is a lack of variability clarity with respect to our ontological framework.

4.6 Related Work

Similar to our work, Reinhartz-Berger et al. [133] conducted an ontological analysis of variability modeling using Bunges ontology. They analyzed process variability and identified a set of variability patterns in the behavior of products. They used their variability framework to perform a formal operational feasibility analysis. In our study, we have investigated both structural and behavior variability which need to be modeled using variability languages. Moreover, our framework is used to evaluate expressiveness of variability languages for representing types of differences among products.

Several frameworks have been employed for evaluating variability languages from practical points of view. Specifically, two main works have been done to evaluate feature models [67, 42]. Diebbi et al. [42] performed a study in an company and established a set of criteria by studying software engineers main expectations for a variability notation. Heymans et al. [67] developed a formal quality method based on SEQUAL (SEmiotic-QUALity) for evaluating the expressiveness of feature models. In comparison to these works, instead of practical consideration we employed a theory-based approach which can better reveal complete and non-redundant set of variability types.

4.7 Conclusion

Variability plays a pivotal role in systematic reusability in SPLE. This prompts a need for new methods for detailed and formal analysis of the variability notion. However, existing variability formalizations lack theoretical foundation. Trying to address this challenge, our work explores the notion of product line variability using the ontological theory and provides a theoretical analysis of variability modeling. Ontological understanding of variability is beneficial from several aspects. First, based on the results of ontological analysis, software developers can clearly identify sources of essential variability in information systems and consider those sources during domain engineering lifecycle. Second, the possible patterns of variability (i.e., types of differences) among products in terms of structure and processes can be identified. This can support the evaluation of variability modeling languages and the improvement of their expressiveness in representing variability. This work is part of continuing work for developing theoretical foundations for variability modeling and configuration decisions in software product lines. We intend to enrich the variability framework with patterns and sources of technical variability.

Chapter 5

Developing Feature Model From Intentions

This chapter consists of "Goal-driven software product line engineering" paper which is published in *proceedings of the 26th ACM Symposium on Applied Computing (SAC 2011)*.[1] The approach presents a proposed technique to map requirements (represented by goal model) to the variability model (represented by feature model) and develop a pre-configured feature model based on reasoning on high level objectives of target stakeholders.

Author's role - I investigated the goal model and feature model and their differences and commonalities. Also, I developed the mapping relations and pre-configuration process. Furthermore, I created the case-study and wrote the text in the paper, co-authors contributed in shaping the ideas and revising the written text.

[1]The paper reprinted (with minor adjustment to formatting), with permission from ACM Press.

Asadi, M., Bagheri, E., Gasevic, D., Hatala, M. (2011). Goal-driven software product line engineering. In Proceedings Of the 26th ACM Symposium on Applied Computing (SAC 2011), Taiwan: 691-698.

5.1 Abstract

Feature Models encapsulate functionalities and quality properties of a product family. The employment of feature models for managing variability and commonality of large-scale product families raises an important question: on what basis should the features of a product family be selected for a target software application, which is going to be derived from the product family. Thus, the selection of the most suitable features for a specific application requires the understanding of its stakeholders intentions and also the relationship between their intentions and the available software features. To address this important issue, we adopt a standard goal-oriented requirements engineering framework, i.e., the i* framework, for identifying stakeholders intentions and propose an approach for explicitly mapping and bridging between the features of a product family and the goals and objectives of the stakeholders. We propose a novel approach to automatically pre-configure a given feature model based on the objectives of the target product stakeholders. Also, our approach is able to elucidate the rationale behind the selection of the most important features of a family for a target application.

5.2 Introduction

A software product line (SPL) covers the feasible space of all possible software products for a given domain of interest. In other words, it provides the means for capturing the commonalities of all possible products of a given domain and also addresses variability by covering a comprehensive set of dissimilarities between the products. In SPLs, characteristics of a software system mostly including its functionalities are represented by *Features* [125]. Software product lines have two main lifecycles, namely domain engineering and application engineering. The *Domain Engineering* lifecycle is concerned with representing all of the features of a set of similar/related software systems as a Feature Model. Later, the *Application Engineering* lifecycle involves the elicitation of the needs, requirements and expectations of the stakeholders to develop a suitable final

product based on a given feature model. As the target domain becomes more complex, the structure of the feature model grows to be more complicated with many more features and interactions between its features.

Given large-scale software product families (modeled using feature models), the main important question is how and what features should be selected for the next product that is going to be derived from this product family.

The process of selecting the appropriate features for a product from the feature model is referred to as the *Configuration Process*. This process requires the consideration of many factors such as technical limitations, implementation costs, and stakeholders expectations. Moreover, stakeholders are not always familiar with the structure of feature models and the available feature. In addition, stakeholders are often more comfortable to express their needs in terms of their goals and objectives. Therefore, in order to be able to select the best set of features based on the stakeholders intentions and expectations, the software practitioners should understand the relations between the available SPL features and stakeholders goals and intentions. It is not easy for the stakeholders to view a feature model and decide which set of features are the ones that they are interested in or which ones are the most beneficial and useful for their purpose. Even more, it is not enough to know what features exist and can be selected, what their interactions are and how they are able to perform their tasks, but it is also important for the stakeholders to know why these features essentially exist, interact and perform in the way that they do. Knowing the why behind the existence of these feature model elements can easily speak to the intentions and objectives (goals) of the stakeholders [188].

A goal is defined as something that the stakeholders hope to achieve as a result of the development of a software product. In other words, it is the high-level objective of the business, organization or system owned, managed, or operated by the stakeholders [9]. Stakeholder goals have been widely captured through goal models within the requirements engineering domain [10, 85, 191]. Goal models are graph-like representations of the relationship between stakeholders goals and their operational plans. They are often used to represent the realistic space of stakeholders intentions and objectives [188]. Goal models are fundamentally built over three important concepts, namely *goals, soft-goals*, and *plans* (aka scenarios or tasks). Goals are objectives related to the functional aspects of the system. In contrast, soft-goals refer to quality attributes of the system. Furthermore, plans are ways to operationalize stakeholders goals.

The idea of employing goal models within the software product family has recently gained focus [190, 191, 25]. The proposed approaches have mainly attempted to use goal models for representing the variability of software products and transforming goal models into corresponding feature models. Although this idea provides the means for representing the features of a product family in terms of the stakeholders objective, it confines variability to those defined based on the stakeholders point of view (referred to as *essential variability*) and ignores technical and infrastructure variability (e.g., variability related to hardware, and operating systems). With this issue in mind, the main research questions that we are interested in addressing in this paper are: how are the most suitable features for a target application selected based on stakeholders needs; how do the stakeholders needs and goals relate with the available features within a feature model and variabilities (essential and technical), in other words, what is the relationship between the feature space and the intention space; and finally, how can a relationship between the stakeholders goals and the SPL features be formulated to select the best set of features for a target application of interest.

In this paper, we investigate how the most suitable set of features can be selected for the product configuration process by examining the stakeholders needs and requirements gathered through a standard goal-oriented requirements engineering process. The stakeholders intentions and goals are important within the software product line feature selection and prioritization process as they ensure that: 1) a complete and comprehensive set of initial features from the set of available features is selected (that can be passed into automated feature model configuration processes), which is due to the fact that we can make sure that all stakeholders intentions, objectives and concerns are covered and addressed by the selected features; 2) irrelevant superfluous features are not included in the selected features, since features that do not correspond with at least one of the stakeholders goals can be considered irrelevant for the target application and be not considered; and 3) the rationale behind the feature selection process is clear for the stakeholders. This becomes very important as stakeholders may not be able to clearly understand the utility of the selected features, but are able to analyze and see the importance of these features in relation to their objectives.

We propose explicit mappings to link features of a SPL feature model to the stakeholders goals and objectives defined in goal models. We support this process by appropriate tooling and implementation technology (Section 5.4). These mappings help

software practitioners move from the stakeholders goals and expectations towards feature model selection decisions in such a way that a more desirable product is developed. Additionally, through the mapping mechanisms, practitioners can conduct reasoning on the stakeholders preferences and objectives, and hence the most relevant features to stakeholders objectives are configured automatically. Finally, by creating goal models and mapping them onto feature models, application engineers can communicate with the stakeholders based on their own jargon and abstract them away from realization details. To explain the proposed approach, we describe a case study 5.3 and develop the case study using our proposed approach in Section 5.5. After acknowledging the related works in Section 5.6, we conclude the paper with a discussion on the advantages and limitations of our work as well as outlining future work.

5.3 An Illustrative Example

Before describing our proposed approach in detail, we introduce the case-study that is used for evaluating our approach. This benchmark case study consists of designing a family of *e-shop systems* that mainly supports ordering of products and shipment of the products to the customers. In our e-shop family, we assume that in all the e-shop systems, a common scenario is as follows: customers select some products and add them to their cart. They then perform the purchase process and the system verifies the customers order and if the order is valid, the product(s) is/are shipped to the customers. To support this general scenario, the system should first show the products with their available quantities to the customers. It should also provide facilities for the customers to create an account, manage their account, and do online shopping. Also, the system should verify customer orders from the perspectives of validity and correctness. After approving the order, the system processes the payment. Finally, the products are shipped to the customers along with the issued bill. Different variations for the general scenario based on the specific stakeholders functional and non-functional requirements can be defined. Additionally, each e-shop variant may include some additional scenarios in order to offer more to their customers. The e-shop family case-study has been used in both SPL [33] and goal-oriented requirements engineering [190]. We adopted the description in these resources and developed a case-study which combines the pro-posed case-studies in the literature. Due to space limitation, we only use a subset of the e-shop system, i.e., the "supply customer order" part.

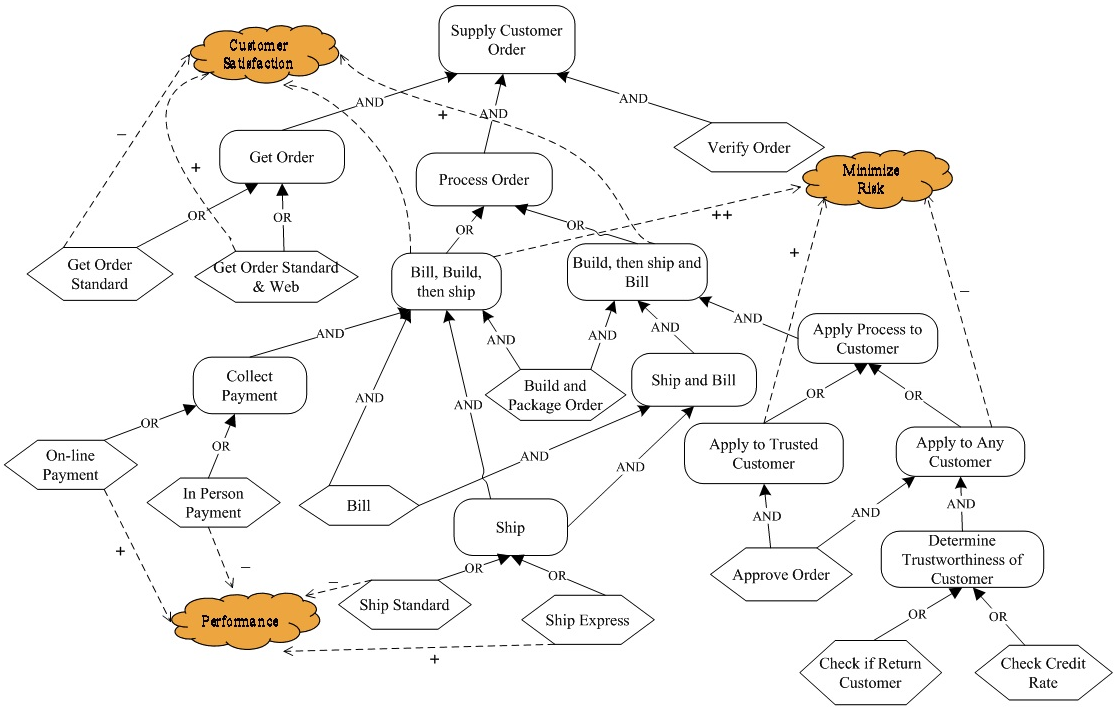


Figure 5.1: Part of the e-shop goal model

5.4 Goal-Feature Model Mappings

For mapping goal and feature models, we adopt an approach similar to the template-based approach proposed by Czarnezki et al. [33]. An overview of the mapping approach is shown in Figure 5.3.

This figure shows the main products involved in the mapping process and also the activities that are performed to produce a pre-configured feature model. A family goal model represents the objectives of the family members with their relations and the feature model represents a hierarchy of features and the defined constraints between features. Through the developed mapping mechanism (discussed in Section 5.4.2), developers can annotate features with references to goals and create the mappings. Next, when a new product is needed, reasoning, based on the stakeholders high level goals is performed on the family goal model. Afterwards, an automatic pre-configuration of the feature model can be performed through the proposed algorithm introduced in Section 5.4.3 to produce the feature model which conforms to the stakeholders goals. Our proposed

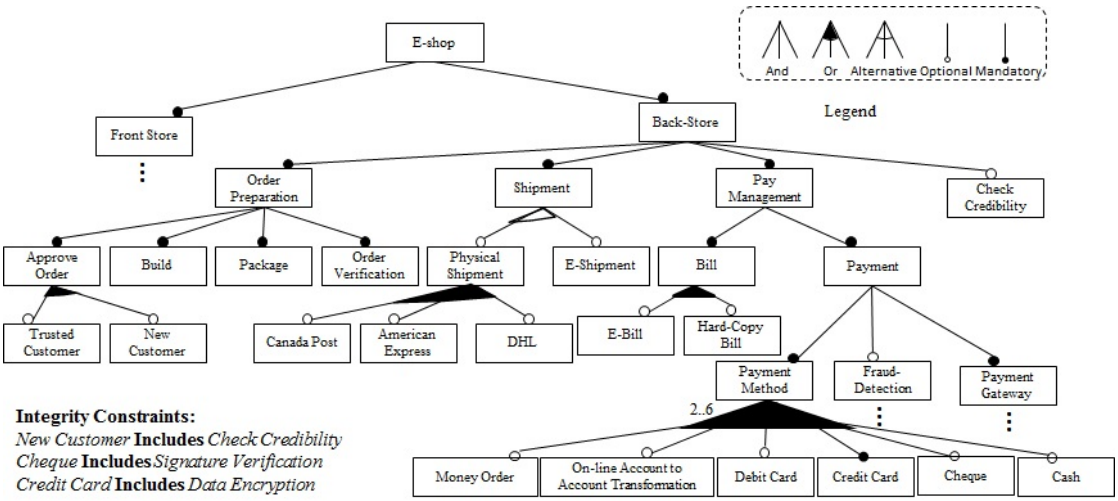


Figure 5.2: Part of the e-shop feature model

approach ensures that the pre-configured feature model contains features which are based on the stakeholders goals and preferences (soft-goals). Before describing the technical details of the mappings, we describe the steps that need to be per-formed in order to develop mappings by the domain engineers in the following.

5.4.1 Develop and Map Feature Model

In order to formulate the mappings, a requirements engineer needs to have both the feature model and the goal model available. Many techniques have been proposed by researchers in goal oriented community and SPL community to independently develop a goal model [125, 85] and feature model [88]. Additionally, feature model derivation from goal model has been explored by Yu et al [191] and Uno et al [165]. Developing a feature model from a goal model ensures traceability between features and stakeholders intensions and facilitates the mapping process between the goal model and feature model. On the other hand, since we adopt Batorys definition of features [21] (i.e., features are incremental pieces of functionality), goal models (especially goals) can be used as a reference to define features and develop a feature model. Therefore, we assume the goal models have been developed by an existing approach. We propose steps to create a feature model based on the goal model and map them to the relevant goals.

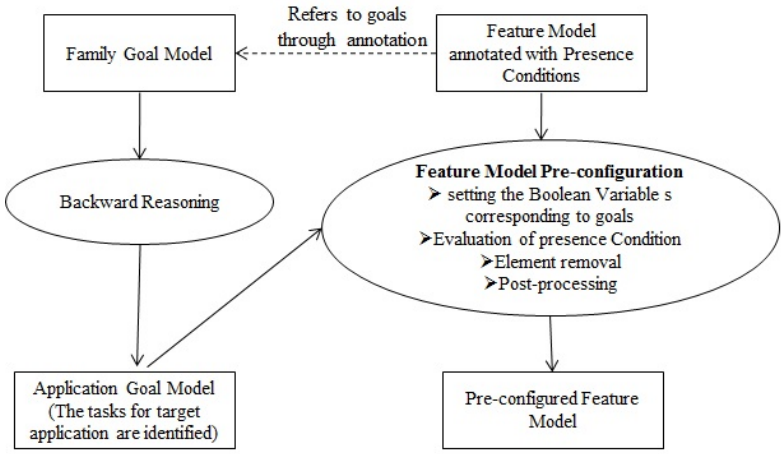


Figure 5.3: Overview of mapping process in application engineering for pre-configuring feature models (adapted from [33])

- *Capture features related to tasks*, domain engineers need to examine low-level-goals (i.e., plans or task) and try to define features that can realize these tasks. For defining features for a task, the task, technical infrastructure related to the task and its environment constraints are considered to describe the features. For example, in the e-shop example, we have analyzed task On-line Payment as well as its required infrastructure and environment constraints. We identified some features including, Debit Card, Credit Card, On-line Account to Account Transformation. Furthermore, by analyzing required infrastructure and environment constraints (i.e. technical details) of these features, the features Payment Gateway and Fraud Detection are discovered. Feature Payment Gateway is OR-decomposed to features Authorize.Net, Paraddata, Skipjack, VeriSign, and Payflow Pro. Feature Fraud Detection is also OR-decomposed into Authentication, Signature Verification, and Data-Encryption. Finally, feature Data Encryption is analyzed and is XOR-decomposed into features DES/3DES Algorithm, RCA, and IDEA. Apparently, many features can be defined that reflect differences in the feature model on technical issues, about which some groups of stakeholders might necessarily not be concerned at that technical level. Note also that most of the features mentioned here are not shown in Figure 6.2 due to the space limit, although they are part of the feature model.

- *Define parent features:* After extracting all atomic features related to the tasks in the goal model, domain engineers develop a feature model through iteratively grouping features with lower granularity to higher granularity by using existing relations in the feature model (i.e. OR, AND, Alternative) as well as the integrity constraints between features. In order to structure atomic features, domain engineers group features which are relevant to each other. Relevance of features is defined according to the relation between goals which they realize. Features realizing the goals which have the same direct or indirect parent can be considered to be relevant by domain engineers. Existing relations between goals (i.e., AND or OR-decomposition) are used to define proper relations between features in a feature model.

- *Define mappings between features and goals:* This step can be performed either in parallel with the previous step or after the previous step is finished. Using the mapping structure (i.e. annotating features with goals that they realize) given in Section 5.4.2, domain engineers can create mappings between goal and feature models. At the end of this stage, the feature model and the mappings which map features to goals are created.

5.4.2 Mapping Infrastructure

Similar to the template-based approach [33], we employ the concept of *Presence Conditions (PCs)* to annotate features of feature models with the goals available in the related goal models. PCs are defined in terms of the goals and are evaluated based on their satisfaction degree (FD, PD, PS, FS). When a feature is annotated with a PC, the PC indicates whether features should remain in or be removed from the feature model during the pre-configuration process. PCs are expressed through the use of Boolean formulas over sets of variables, where each variable corresponds to a goal in the goal model. Since we consider features as functional increments, we only define features PCs based on the goals and task (not soft-goals). After reasoning on the goal model and determining the goal satisfaction/denial, the appropriate value is assigned to the variable. The variable is true if and only if the corresponding goal is labeled with FS, PS, and PD; and the variable is false if the corresponding goal is labeled with FD.

For example, consider Figure 5.4(a), which illustrates the Collect Payment goal along with the features realizing the tasks of the goal. A boolean variable is defined for each

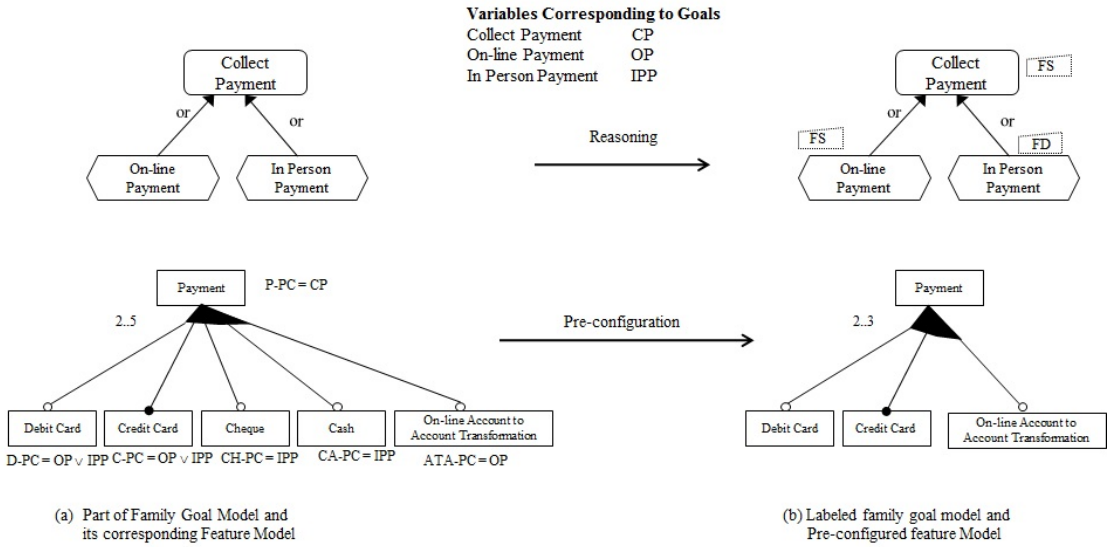


Figure 5.4: An example of mapping between goal and feature models where a reasoning algorithm. The algorithm selects the On-line payment plan based on the high-level objectives of the stakeholders. Consequently, the appropriate features are kept in the feature model.

task (CP variable for Collect Payment, OP variable for On-line Payment, and IPP for In Person Payment). Next, according to goals which are mapped to features, PCs of the features are formulated as boolean expressions of variables corresponding to the goals. For example, the On line Payment and In Person Payment goals are mapped to the Debit Card Payment feature, so the features PC is expressed as Debit Card-PC = OP∨IPP . The PCs are constructed automatically when domain engineers map features to goals through the use of the provided tooling support. Having features mapped to goals, the reasoning process over the goal model is able to label the goal model tasks based on the stakeholders high-level objectives. As shown in Figure 5.4(b), the On-line Payment and In Person Payment tasks are labeled with Fully Satisfied (FS) and Fully Denied (FD), respectively. Due to the goal labels, the OP and IPP variables are valued to true and false, respectively. Consequently, the PCs of the features are evaluated and their final Boolean values are determined. After checking the rules, introduced in the next section, to ensure that feature model constraints have not been violated, the features Money Order, Cheque and Cash are removed from the feature model.

We should note that if a features PC is set to true then its corresponding sub-features PCs are evaluated; otherwise if a features PC is false (i.e. it is determined that the feature should be removed from the feature model), then all its sub-features are removed except for sub-features which are involved in some integrity constraints such as *includes* for which suitable actions are defined. For example, let us assume the feature Fraud Detections PC is evaluated "false". So, the feature as well as its sub-features must be removed from feature model. However, the feature Credit Card includes the sub-feature Data Encryption of the feature Fraud Detection. On the other hand, the feature Cheque requires feature Signature Verification to validate the customer signature on the cheque. Hence, all other sub-features are removed except features Data-encryption, Signature Verification and the parent feature (i.e. Fraud-Detection). Moreover, existing feature relations in the feature model are removed when their corresponding features are deleted from the feature model. So, we do not need to annotate them or map them to the elements in the goal model. Since the mappings between features and goals are not one-to-one corresponding, PCs may become more complex logical expressions. When a feature is annotated with more than one goal, meaning the feature is used for realizing more than one goal, we define $\vee$ (Or) relation among corresponding variables of the goals. Also, one or more features may be used for realizing one goal. In such a case, all of them are annotated with that goal.

5.4.3 Feature Model Pre-Configuration

Through the use of the mapping mechanisms defined in the previous section, developers can map features to goals by annotating features with Boolean variables that correspond to the goals. Next, during configuration, the pre-configured feature model is generated automatically. The pre-configuration process is a *model-to-model transformation* where both the input and output models conform to the feature metamodel [35]. The process involves assigning true or false values to variables based on their corresponding goal labels (i.e., FS, PS, PD, FD), evaluating the PCs according to the variables and constraints in the feature model, and possibly employing some additional simplification processes.

After backward reasoning on the goal model, it can be seen that the goals with FD labels are not based on a high level objective of the current application stakeholders. Hence, features realizing such goals can be removed. That is, we considered the Boolean variable corresponding to each goal (hard-goal and plan) and each features PC is defined

as a logical expression of these variables. Therefore, by assigning false values to variables whose corresponding goal is fully denied, we can evaluate the features PC. Then, features with values of PCs equal to false are candidates to be removed from the feature model. However, uncontrolled deletion of features may violate general constraints (e.g., removing a mandatory feature in an *And* grouped feature) or integrity constraints (e.g., removing one feature involved in an *include* relation while the other feature remains in the feature model). Therefore, during the pre-configuration process, we should make sure that these constraints are not violated. We discuss each of the constraints and describe the proper mechanisms (rules) for managing possible violations. All of these rules are checked automatically on the feature model and proper actions are adopted based on the case under investigation. In the remainder of this section, we call a feature a *Candidate Feature*, when the features PC is evaluated to *false*, which shows that the feature could be removed.

With respect to general constraints in the feature model, we discuss this for each relation (i.e., Or, And, Alternative) and type of features (i.e., mandatory or optional).

- *And, Or* and *Alternative* relations: If a candidate feature involved in an AND relation is a mandatory feature, the feature cannot be removed from the feature model, but it is labeled as a denied feature. For example in Figure 6.2, the Order Preparation feature cannot be removed even if its PC is evaluated to false, since it violates the feature model constraints. However, if a candidate feature is an optional feature, it can be removed from the feature model after checking its integrity constraints and group cardinality constraints. For instance, the Customer Verification feature in Figure 6.2 can be removed if its PC is evaluated to false. It still should be checked w.r.t the other constraints such as integrity and cardinality before the removal.

- *Group cardinality ⟨ n1, n2 ⟩:* If a feature is a candidate feature and the removal of the feature causes group cardinality violation (i.e. n1 ⟩ n2), first all siblings of the feature which are candidates to be removed are counted. If the deletion of the candidates violates the group cardinality constraint, they are kept in the feature model and are labeled as denied. For example, in Figure 5.4, removal of the Cash and Cheque features does not violate the feature cardinality constraint.

- *Feature cardinality:* feature cardinality does not impose any limitations on the

removal of features from the feature model.

In all aforementioned cases, if a feature, which is removed from the feature model, is not an atomic feature, all of its children independent of their PC are removed from the feature model. In addition to the general constraints, integrity constraints may also affect the deletion of the candidate feature. These constraints include:

- *F1 includes F2*: if F1 is not a candidate feature and F2 is a candidate feature, according to the *includes* constraint, feature F2 cannot be removed from the feature model. Therefore, we keep feature F2 and label it as a denied feature. For example, in Figure 6.2, the New Customer feature includes Check Credibility feature. Assume the PC of the New Customer feature is evaluated to true and the PC of Check Credibility is evaluated to false. By applying the general rule and since it does not violate any of the general feature model constraints, it can be removed. Based on the integrity constraints, it cannot be removed and we should keep it and label the feature as denied.

- *F1 excludes F2*: independent of this integrity constraint, based on values of the PCs of these features and other constraints, a proper action is adopted. If the PC of each feature is false, the feature is removed after checking other constraints.

The *simplification* step is conducted on the feature model after the removal of the features in order to correct grouped cardinalities and remove the extra constraints such as *includes* and *excludes* when one of their features is removed. Additionally, for relations that only have one sub-feature and the other sub-features are removed; the parent feature is replaced with its sub-feature. The pre-configuration algorithm can be summarized as follows:

1. *PCs evaluation*: The Boolean variables are evaluated to true or false based on their goals labels. Next, a depth-first algorithm is performed on the feature model and the PC of each visited feature is evaluated.

2. *Removal analysis:* For each features with their PCs evaluated to false, according to the situations that may occur, one of the above defined mechanisms is executed. At the end of this step, features with their PC evaluated to false are either removed from the feature model or are labeled as denied features.

3. *Simplification step*: Simplification of the feature model is done.

Feature model pre-configuration can be executed before the configuration processes such as staged configuration [35] and S-AHP [17]. In order to conduct pre-configuration the following steps should be done by application engineers: First, in the capture *stakeholders high-level goals* step, application engineers communicate and understand stakeholders needs by identifying their objectives. The family goal model is used as a reference model for communicating with the stakeholders and capturing their goals. In many cases, it would be impossible to fully satisfy all soft-goals [187]. For example, the full and simultaneous satisfaction of performance and security, goals is not possible. So, the desired level of satisfaction for each soft-goal is elicited by the stakeholders. Next, by setting the satisfaction level of high-level goals and executing the backward reasoning algorithm [53], the leaf goals (plans) are selected. Finally, the pre-configuration process is executed and the features, which are not based on the current stakeholders objectives, are filtered out from the feature model. The features which are not based on stakeholders goals, and have not been removed due to feature constraints are labeled as denied. Another scenario for the use of mappings is during the configuration process when an application engineer intends to further specialize the feature model. In this case, application engineers can trace back features to the goals in the goal model using the mapping infrastructure. Therefore, using the forward propagation algorithm, application engineers can find out the influence of removing a feature in satisfaction of high level objectives of the stakeholders.

5.4.4 Tooling Support

In order to provide tooling support, we have been customizing the *fmp2rms* plug-in [33], a tool that integrates the Feature Model plug-in with the Rational Software Modeler (RSM) and provides facilities to map feature models to implementation models. In our context, we reuse this tool for mapping feature models to goal models. That is, we developed a primitive prototype named *OpenME2fmp* that integrates the OpenME plug-in (an Eclipse plug-in used to develop goal models) and fmp (an Eclipse plug-in used to develop feature models). The *OpenME2fmp* utilizes the provided interface of *fmp2rms* and implements the pre-configuration algorithm. For the implementation of the Presence Conditions (PC), we consider boolean formulas in Disjunctive Normal Form. Thus, the only logical connector, that we support in PCs, is the Or relation ($\vee$). With

respect to visualization of mappings, we present both feature models and goal models in the mapping space, by utilizing *fmp* and *OpenME*. We also provide the list view for the goal model which represents the variables corresponding to the goals and tasks. Domain engineers can switch between these views during the mapping process. Besides the mapping area, we provide an area where the mappings are presented (mapping View) from both the goal view and the feature view. That is, in the goal view, we list the goals and put the feature(s) which realize the goal as its sub-items and in the feature view. We also list the features and show the goals which are realized by the features as their sub-items.

5.5 Descriptive Case Study

In this section, we discuss our experience in applying the proposed approach to the e-shop case study. Since our objective is to evaluate and exemplify our approach mappings between goal and feature models, we adapt and use an existing goal model in the literature [190]. Due to space limitation, we just show an application of our approach on a part of the case-study (Supply Order Management). As shown in Figure 5.1, the high-level objectives are *Support Customer Order, Get Order, Process Order,* and Verify Order. All goals identified on this level are common between different products of the e-shop family and required to be fully satisfied. However, different levels of satisfaction of soft-goals (i.e. Minimize Risk, Customer Satisfaction, and Performance.) for the products are identified. Relations between these high-level goals are represented with contribution links.

First, the feature model is developed through analyzing the goal model. For developing the feature model we follow steps introduced in Section 5.4.1. So, for each task, features required to implement the task are defined. By analyzing tasks w.r.t functionalities, environment constraints, and required infrastructure. The features shown in Table 1 are identified. Some of those features are common among different goals (e.g. Credit card is common between On-line and In-person Payment). Also, sometimes two or more goals may be realized just by one feature (e.g. Check credit rate and Check if return customer tasks are realized by the feature Check credibility). The supply order process goal model has 12 tasks, from which 47 features are derived.

Afterwards, the features are grouped with appropriate relation in feature model (i.e. And, Xor, Or). The grouping process start with features related to a task and then

the relations in the goal model are used as direction for defining relation within feature model. For example, derived features from goal On-line payment are analyzed and different approaches of payment are grouped with the OR-relation to form a feature named Payment Method. According to the family description, a cardinality constraint on Payment Method is defined and mandatory and optional features are identified. Let us assume in our application domain all the stakeholders need at least two methods of payment and they want to have Credit Card payment as mandatory feature in all applications. So, we define cardinality constraints [2..n], which means that at least two should be selected for every product in the family. Additionally, features Fraud Detection is decomposed into Data-encryption and Authentication. Payment Gateway is OR-decomposed into the sub-features Authorized.Net, para-data, Skipjack, and Versing Payflow Pro. The features Payment Method, Payment Gateway, and Fraud Detection form an And-grouped feature named Payment. All these features are mapped to and thus annotated with the On-line payment task. By considering in-person payment task, the feature payment method is updated and three more sub-features (i.e. Cash, Cheque, and Money Order) added to this feature and mapped to the goal In-Person payment. Moreover, the feature Signature Verification is inserted to the sub-features of Fraud Detection and mapped to the In-person payment task. Finally, Payment Gateway, Fraud Detection and Payment Methods are mapped into In-person Payment. The feature model integrity constraints are defined as well. For example it, the feature Credit Card includes the feature Encryption. The process follows to develop feature model and map it to the goal model. The part of feature model is illustrated in Figure 6.2.

Tasks	**Related Features**
Get Order Standard task	Paper Catalog, Cart (Item Order Management, Customer, Information Management), Fax, Phone
Get Order Standard and web	All the features of Get Order Standard task plus E-catalog (Search, Brows, Custom view, Item list)
On line Payment	Credit Card, Debit Card, On-line Account to Account Transformation, Fraud Detection (Data encryption, authentication, DES/3DES Algorithm, RCA, and IDEA), Payment Gateway (Autho-rized.Net, Paradata, Skipjack, Versing Payflow Pro)
In Person Payment	Cash, Cheque, Money order, Fraud Detection (signa-ture Verification), credit card, debit card.
Ship Standard	Ship preparation, shipping carrier (Canada Post)
Ship Express	All the features in Ship Standard task Plus Shipping Carrier (FedeX, UPS, USPS, Custom Shipping Gateway), Tracking System
Bill task	E-bill, Paper Bill
Approve Order	Approve Order
Check Credit rate	Check Credibility
Check if return Customer	Check Credibility
Build and Package	Order fulfillment (Physical item fulfillment, electrical item fulfillment, Service fulfillment)
Verify Order	Order management

Having developed the feature model, we can specify the high-level goals for each product and also the satisfaction level for each soft-goal. Then, the reasoning algorithm is

applied to the goal model, based on which the pre-configured feature model is produced. As shown in Figure 5.4, for instance, after reasoning on the goal model, On-line Payment is labeled *fully satisfied* and In-Person Payment is labeled *fully denied*. Thus, Credit Card, Debit Card, and On-line Account to Account Transformation are kept in the feature model and Cheque and Cash are removed.

5.6 Related Work

Goal-oriented software configuration has become an emerging area of research [10, 85, 191, 90]. Here, we review some of the most relevant works on the theme of the paper. Liaskos et al [90] propose an approach which first identifies and understands configuration options of personal software; then they develop a goal model based on the configuration options and use it as a mediator between users and configuration options. Therefore, they automatically transform users high-level goals into a configuration that satisfies the users goals. Lapouchnian et al. [85] use goal models in designing and implementing autonomic systems which are able to select the best behavior according to the context. These systems are able to do self-optimization and self-healing. Our approach differs from these approaches in terms of the domain (i.e. our approach addresses the domain of SPL engineering which has a higher level of variability types and complexity). SPL covers both essential variability (i.e. Variability of the product family from stakeholders point of view) and technical variability (Variability of the product family from the realization point of view). The goal model is used for representing and managing essential variability. Using goal models for technical variability diverges from their primary purpose understandability for stakeholders. So, we used both goal and feature models for managing variability to facilitate the variability binding for stakeholders and developers.

Moreover, Yu et al employed goal models to develop *generic software* "software solutions that can accommodate many/all possible functionalities that fulfill stakeholder goal" [191]. They extended the notations of the standard goal model with sequential, parallel, exclusive, non-system notations and define a set of heuristic rules which are employed to generate design models including feature models, state models, and component models. In [190], Yu et al. have proposed a dedicated tool for creating an initial feature model from a goal model by applying the similar transformation rules in [191]. The main issues of the approach are that first, they had to extend goal models with new notation elements and limit the structure of feature models to the structure of goal

models. Although developers can change the feature model structure later, the main structure is still based on that of a goal model. Yet, we do not add any additional notation elements to goal models and our aim is to map feature models to goal models, not automate the generation of feature models from goal models.

Further, Antonio et al, [10] proposed ISARLPS, which defines a very high level process for developing feature models from goal-model represented by i* diagrams (i.e. Strategy Dependency [SD], and Strategy Rational [SR]). The steps in their approach are defined from a very high level and the approach is a transformation rather than mapping approach. Borba et al [25] also followed the approach similar to Antonio et al define another set of heuristic rules for transforming goal models to feature models. Our approach is different from the above approaches in the policy which is applied to relations between feature models and goal models. We map goal models to feature models instead of transforming them into each other. First, we found out that goals are not the sole source of variability. Second, the structure of goal models is different from feature models. Third, goal models and feature models are designed to represent conceptually different information, and hence in many cases direct transformation does not provide rational results. Finally, we do not want to add new notation elements to the existing goal modeling language. As shown in Section 3.1, a feature model is created based on goal models and all rules defined by related research can be employed as suitable guidelines to for-mulate an initial feature model.

In SPLE, many approaches have been developed to map feature models to design and implementation models [33, 194, 66]. Czarnecki et al [33] have proposed a template based approach which we adopted for mapping goal models to feature models. We had some other alternatives. For example, VML* proposed by Zschaler et al [194] is another approach for mapping features with solution artifacts. Another approach is used in FeatureMapper [66] that allows for mapping feature models to solution space models.

5.7 Conclusion

We promote the consideration of early requirements information in the form of stakeholders goals and objectives in the process of selecting the right features for a particular software product. Such an approach is simple yet effective for ensuring that the best set of features has been selected for a target application. The introduced approach can

serve as a best practice guideline that provides a convenient way for capturing stakeholders intentions, mapping them as concrete requirements into software features of the whole product line family, and feeding the process of the SPL configuration with a more accurate set of stakeholders desired features. The main aim of this paper is to introduce the general idea of identifying the most suitable set of features of a software product feature model for the software practitioners. We argued that stakeholders goals are among the suitable decision rationale for choosing the most suitable features to be included in the final product. We have proposed a framework to help select desired features to be included in the final software product.

As future work, we are going to enhance the support of non-functional properties in product lines by utilizing soft-goals. Currently, we are investigating to first define non-functional requirements based on soft-goals and quality properties and add them to the features through annotations. This is similar to the approach that we have already proposed in our feature selection technique called S-AHP [17].

Chapter 6

Planning Based Process Model Configuration

This chapter consists of "A Planning Based Approach for the Configuration of Process Models" paper which is submitted to *Automated Software Engineering journal.* The paper describes the proposed decision making algorithm (i.e. Stratified Analytical Hierarchy Process) as well as a PDDL planning based technique for developing a configured process model from stakeholders' preferences. A set of transformation rules is defined, which transfers feature models and configuration constraints into a planning domain. The approach is evaluated using several experiments.

Author's role - I was one of the main contributors in developing stratified analytical hierarchy process. Also, I developed transformation rules and implemented the rules in the vis-fmp tool[1]. Finally, I developed a simulation framework and performed a comparative analysis of the proposed approach with the related work. I wrote the paper and my co-authors revised the paper.

[1] vis-fmp is an extension of feature model plugin tool (fmp), which is developed by the author and his colleague in the Laboratory of Ontological Research at Simon Fraser University

Asadi, M., Gasevic, D., Hatala, M., Bagheri, E., (2014). A Planning Based Approach for the Configuration of Process Models. Submitted to Automated Software Engineering (ASE) journal.

6.1 Abstract

Due to the diversity of contexts and stakeholders' requirements, several variants of a process model may coexist in an organization. Configurable process models provide a holistic representation of process variants in an integrated model. Many techniques have been proposed, which provide guidelines for developing and adapting configurable process models. Although these approaches contribute to the development of configurable process models, they lack efficient variability modeling and automated support for adapting configurable process models based on functional and non-functional requirements. In this paper, we propose a configuration approach which formalizes the variability in configurable process models into feature models and provides a planing based technique to automatically adapt configurable process models based functional and non-functional requirements of stakeholders. A set of experiments has been conducted to evaluate the scalability of the approach for different process model sizes and different variability distributions. The results of experiments reveal that the approach is scalable for industrial size configurable process models.

6.2 Introduction

The increasing complexity and interdependencies of today's business operations have motivated organizations to enforce a process-centered alignment of organizations information systems and IT infrastructure. Hence, Process Aware Information Systems (PAIS) were introduced to manage and execute operational processes involving people, applications, and/or information sources on the basis of process models [45]. A process model consists of all activities of a business process and their attributes (e.g. cost and time) as well as control and data flows between activities. Process models are usually represented using a visual language such as Business Process Management Notation (BPMN) and Event Process Chain (EPC).

Process models are typically instantiated several times and every process instance

may differently be executed based on the context and the requirements of target stakeholders [136]. Hence, several variants of a process model may coexist in an organization which urge the organization to support flexible processes to cope with process variabilities [136]. Motivated by the need of flexible process models, a number of approaches have been proposed for the development of *configurable process models* [55, 84, 137]. A configurable process model is an integrated set of all possible process configurations as well as a guideline for leading stakeholders to a solution fitting to their requirements [55]. Although the existing research on the configurable process models improves developing and configuring process models, two main challenges still exist:

Challenge 1 (Modeling complexity): Many configurable process model approaches incorporate variability into process models by introducing new constructs over process model languages [55]. For example, Gottschalk et al. [55] and Rosemann [137] extended EPC and YAWL with configurable connectors and hide and block constructs, respectively. In these approaches, variability is not considered as a first class object and variant-specific information is maintained in the constructs of a customizable process language [65]. Hence, process models are overloaded with selection options (i.e., variation points and variants) and dependencies (i.e., integrity constraints) which increases the complexity of configurable process models during the development of these models.

Challenge 2 (Configuration Complexity): Selecting the right and most important process elements for each individual customer requires understanding his/her requirements as well as the relations of configuration options with the given requirements. Considering the high complexity of configurable models due to including process and variability logic and the diversity of stakeholders' requirements, configuring a process model is a cumbersome task for process engineers. Specifically, a process engineer needs to take following factors into account when deriving a process from a configurable process model.

- Industrial configurable process models may contain hundreds of configuration options which leads to the exponential growth of the number of possible configured process models. Manually deriving an optimal process model from the high number of possible options demands high cognitive load from process engineers.

- Activities may have different quality attributes [17] such as *performance* and *cost* which we refer to as non-functional properties. The non-functional properties can

be quantitative or qualitative. On the other hand, stakeholders may have several constraints and preferences over non-functional properties during the process derivation. For example, one stakeholder may ask for a configured process model with *high security*, *high customer satisfaction*, and a specific *cost*; and can mention that *customer satisfaction* is more important than *security*; which would make the configuration process more difficult and complex [23, 155].

In order to address the first challenge, inspired from software product lines community, several researchers have considered variability as a first class object and adopted variability modeling languages such as feature models [73] for modeling variability and selection dependencies (i.e., integrity constraints) in process models [106, 59, 112]. We adopt a similar strategy for modeling the process variability and use feature models for representing the variability of configurable process models. A traceability model is developed which maps activities and their attributes (e.g. quality attributes) in configurable process models to features in the feature models.

Having represented the variability of configurable process models in the feature models, we propose an automatic configuration technique based on Planning Domain Definition Language (PDDL) which identifies the optimal configuration based on the given requirements. The proposed technique receives stakeholders' preferences and computes the ranks of features. Next, the feature model is transferred into a PDDL-based planning domain and constraints and an objective function (optimizing preferences) are transferred into a planning problem. Afterward, existing planners are utilized to find an optimal plan which corresponds to the optimal feature model configuration. Finally, activities mapped to the deselected features are removed from the configurable process model and a configured process model is generated. Previously, we applied a Hierarchical Task Network (HTN) planning technique to produce the feature model configuration [155]. However, due to limitations in the existing planners for HTN planning, we decided to concentrate on classic planning techniques which provide richer constructs and stronger planners. Furthermore, in this work, our contribution concentrates on the process model configuration which includes modeling the mapping between feature models and configurable process models and process derivation. The formalization for the requirements, the non-functional properties, and the overall configuration approach is similar to our previous proposed work [155].

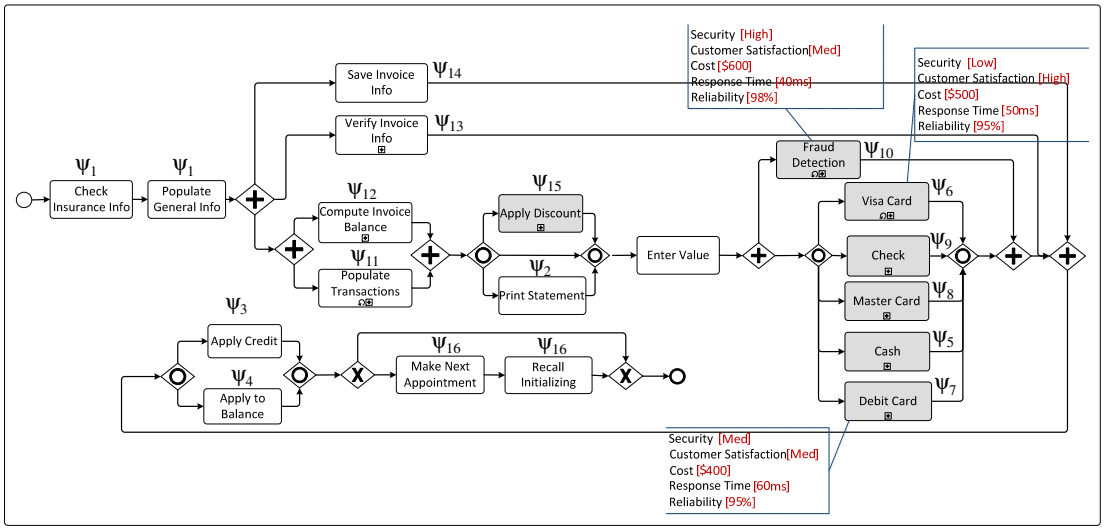


Figure 6.1: A part of the reference process model: Check out Subprocess.

Section 6.3 describes the running example which is used in the paper. The formulation of configurable process models variability into feature models and the mapping model are explained in section 6.4, which is followed with the description of the proposed configuration process in section 6.5. The performance evaluation results and comparisons with related work are described in sections 6.6 and 6.7, respectively. Finally, we conclude the paper and highlight the future directions in section 6.8.

6.3 Case Study Description

We exemplify our approach by a running example from the health care domain. In collaboration with our industrial partner, we produced a process model of their information system. The company's system is used to assist optometrists in their daily activities such as conducting different types of eye exams and managing patients. The system supports most of the common processes including checking a patient in, examination, and checking out processes. Figure 6.1 shows part of a check out process model of the system.

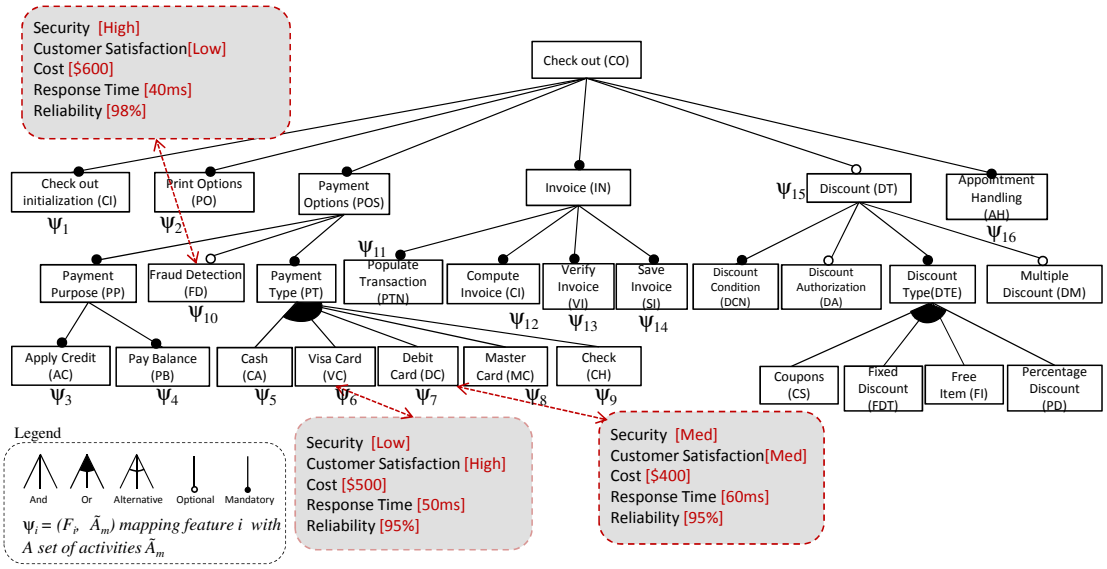


Figure 6.2: Feature model corresponding to check out sub-process

6.4 Modeling Variability in Configurable Process Models

In configurable process models, the variability represents the differences among configured process models in terms of their process elements. The variability is based on *variation points*, where the differences happen, and *variants*, possible options for a given variation point [125]. Different variability patterns [12] (e.g., alternative, OR, optional) may occur between activities. For example, *alternative variability* pattern among a set of activities represents that each configured process model must have only one of the alternative activities. *OR variability* pattern shows that one or more activities among a set of activities in a configurable process model may appear in each configured process model. For example, in Figure 6.1, one or more payment methods may be selected for each configured check out model. In many configurable process modeling languages, variant-specific information is maintained in the constructs of configurable process models [65] which overloads process models with the selection options (i.e., variation points and variants) and dependencies (i.e., integrity constraints). Incorporating both variability and process logic in a single model, especially for large size process models, increases the complexity of customizable process models for the development and customization of these models.

Due to abovementioned problems, we decided to represent variability in a separate dimension and used feature models for this purpose. In the context of configurable process models, we consider features as abstraction over process models where a feature is mapped to *one or more* activities and their quality attributes. Accordingly, feature non-functional properties represent the aggregation over the corresponding quality attributes of activities mapped to the feature. Figure 6.2 shows the corresponding feature model for check out sub-process in our running example.

In order to represent mappings, we adopted the annotation based approach proposed by Czarnezki et al. [33]. Therefore, each activity in a configurable process model is annotated with the concept of Presence Condition (PC) which indicates if the activity should be removed from the configurable process model during the configuration process. Presence conditions are expressed through the use of Boolean formulas over sets of variables, where each variable corresponds to a feature in the feature model. For example, the figure 6.2 shows check out initialization (CI) feature is mapped to check insurance info (CII) and populate general info (PGI) activities from Figure 6.1 which leads to annotating them with Ψ_{CI}.

6.5 Configuration Method

A configuration refers to the specialization of a process model into a configured process model by resolving its variability [136]. During a configuration, a process engineer makes several decisions based on the stakeholders' goals and preferences. For each decision making situation, a process engineer evaluates alternatives and makes intuitive decisions. For large size problems with several dependencies, intuitive decision making would not be efficient and may not lead to an optimal solution [148]. Method based decision making can help handle the complexity of large scale decision making problems via machine support. In this section, we introduce a planning based configuration method to systematically capture the stakeholders' requirements and assist process engineers in finding a process instance fitted to the stakeholders' requirements. The proposed method is comprised of: *Capturing stakeholders' requirements, computing features ranks, planning based configuration,* and *deriving a configured process model.*

6.5.1 Capturing Stakeholders' Requirements

Stakeholders' requirements include both functional and non-functional requirements. In the context of our work, the functional requirements refer to sub-processes or activities which are requested by the stakeholders. For example, a stakeholder's functional requirement might be including a credit card payment sub-process into a configured process model.

Regarding the non-functional requirements, we consider *constraints* and *preferences*. Constraints are the limits that a stakeholder indicates over the total values of non-functional properties. For example, a stakeholder may ask that a configured process model must have medium *customer satisfaction* and a *cost* less than 2000$. Thus, the final process model should not have any sub-process (i.e., feature) that provides *customer satisfaction* less than the medium level. We assume that a process model has some level of a qualitative non-functional property (e.g., medium *customer satisfaction*), if all of its sub-processes have at least that level of non-functionality (i.e., all sub-processes must have at least medium *customer satisfaction*). Also, the total cost of the configured process model should be less than 2000$.

Non-functional properties may not have the same importance for a stakeholder. That is, a stakeholder may be more interested to a subset of non-functional properties. Moreover, in some situations, there might exist conflicts between requirements defined over non-functional properties that have opposite behaviors. For example, request for high *security* may conflict with the request for high *performance* in the systems. In these situations, the stakeholder needs to choose between the competing options [155]. One approach for specifying the priority between non-functional properties is through formalizing their relative importance relations. Relative importance is defined as follow [155].

Definition 6.1 ***(Relative Importance)***. *Relative importance between non-functional properties a and b is: $a \succ^{\alpha} b$ if the non-functional property a is more important than the non-functional property b with coefficient α.*

Usually, values 1, 3, 5, 7, and 9 corresponding to equality, slight value, strong value, very strong and extreme value are used to represent the degree of importance of options (non-functional properties) [139]. For example, relation $Security \succ^{3} Performance$ represents that *security* is slightly more important than *performance*. We refer to the relative

importance between non-functional properties as stakeholders' preferences.

6.5.2 Computing Feature Ranks

After capturing the constraints and preferences, we calculate the ranks of features. For this purpose, we employ Stratified Analytical Hierarchy Process (S-AHP) algorithm, proposed in our previous work [17]. S-AHP is an extension of AHP algorithm, which uses AHP for computing the weight of non-functional properties and employ utility theory to compute the rank of features. The process of S-AHP includes:

Computing the weight of non-functional properties: in this step preferences of stakeholders are received in terms of relative importance relations and then an evaluation matrix $\mathcal{M}(n, n)$ is generated, where n is the number of non-functional properties and $m_{i,j}$ shows the relative importance of property i to property j. The algorithm iteratively computes the eigenvectors until the difference between eigenvectors is less than a given threshold(ϵ). In order to compute an eigenvector, first the algorithm squares the evaluation matrix by multiplying the matrix to itself. Next, the values of each row are summed and accumulated in a result vector. Finally, each value in the matrix is normalized by dividing the value to the corresponding value of the result vector. The computed eigenvector is compared with the previous enigenvector and the algorithm stops if difference is less than the given threshold. The final eigenvector shows the weight of non-functional properties [17]. For example, Table 6.1 shows RI matrix corresponding to a stakholder's preferences. S-AHP computes the final eigenvector which equals to the weights of non-functional.

Table 6.1: Relative importance of non-functional properties.

	Cost	Sec.	Resp.	Rel.
Cost	1	3	7	5
Security	1/3	1	3	3
Response Time	1/7	1/3	1	1/3
Reliability	1/5	1/3	3	1

$$\begin{Bmatrix} Cost & 0.58 \\ Security & 0.23 \\ Response\ time & 0.06 \\ Reliability & 0.13 \end{Bmatrix}$$

Computing the rank of features: In this step the rank of a feature is computed based on the non-functional properties assigned to the feature. For example, since debit card feature is annotated with *security, customer satisfaction, cost, response time*, and *reliability*, the rank of the feature is computed based on aggregating the impact of these non-functional properties. According to the type of non-functional properties (e.g. qualitative or quantitative) and the preference rule defined for each non-functional property, the impact of the non-functional property is calculated differently. For a qualitative non-functional property, a stakeholder defines the relative importance relations between the property qualifier tags and then S-AHP algorithm is applied to compute the impact of each qualifier tag. Next, the impact of the qualitative non-functional property on a feature is calculated by multiplying the weight of the non-functional property and the qualifier tag assigned to the feature. For example, consider that the weight of *security* non-functional property is 0.2 and qualifier tags low, medium, and high have impacts of 0.1, 0.3, and 0.6, respectively. Then impact of *security* over credit card feature is: $0.2 * 0.6 = 0.12$, because *credit card* feature is annotated with high *security*. For quantitative non-functional properties in order to remove the impact of differences in the scales of non-functional properties, first the value of the non-functional property is normalized using the given formula

$N(q_n(f)) = \frac{q_n(f) - \mu_n}{\sigma_n}$

where μ_n and σ_n are the average value and the standard deviation of the quantitative non-functional property (n) for all atomic features in the feature model.

After normalizing the quantitative non-functional properties and computing the value of qualifier tags of each qualitative non-functional property using S-AHP [17], we compute the ranks of the features using the following utility function [155].

Definition 6.2 ***(Utility Function)***. *Let us assume there are α quantitative non-functional properties to be maximized, β quantitative non-functional properties to be minimized, and θ qualitative non-functional properties whose impact needs to be maximized. The utility function for feature f is defined as:*

$$R(f) = \sum_{i=1}^{\alpha} w_i \times N(q_i(f)) + \sum_{j=1}^{\beta} w_j \times (1 - N(q_j(f))) + \sum_{k=1}^{\theta} w_k \times M_k(QT(f))$$

where w is the weight of each non-functional property calculated by S-AHP such that $0 \leq w_i, w_j, w_k \leq 1$ and $\sum_{i=1}^{\alpha} w_i + \sum_{j=1}^{\beta} w_j + \sum_{k=1}^{\theta} w_k = 1$ and $M_k(QT(f))$ returns

the real number corresponding to quality tags of k-th non-functional property computed by S-AHP.

6.5.3 Planning Based Configuration

After identifying the rank of features via S-AHP algorithm, we select the best set of features which satisfy the constraints over non-functional properties and feature model relations. To this end, we apply PDDL based planning techniques to find an optimal configuration. Feature models and the constraints over non-functional properties are transferred into the planning domain and problem, respectively. Afterward, a planner is utilized to find an optimal plan based on the planning problem and the generated plan is transferred into a configured feature model.

Domain Transformation

We devised two types of rules for generating a domain description from a feature model: *rules for generating domain predicates and functions*; and *rules for generating actions.*

Generating domain predicates and functions. As shown in Figure 6.1, activities in configurable process models are annotated with non-functional properties. Since activities are mapped to features, the non-functional properties are propagated to features. During the configuration process, only features must be selected that do not violate the constraints defined over non-functional properties. That is, the value of a non-functional property for a feature is considered to decide if to include or exclude the feature in the configuration. On the other hand, in the PDDL, precondition of an action is considered to decide if to include or exclude that action in a plan. Action preconditions are formulated in terms of *logical expressions* which are the combination of *logical atoms* and *logical connectives. Logical atoms* involve *predicates* - properties of objects that we are interested in (can be true or false) [49]. According to the correspondence between the role of preconditions in the selection of actions and non-functional properties in the selection of features, we generate domain predicates for non-functional properties. Figure 6.3 illustrates the generated domain predicates. A domain predicate is generated for each qualifier tag of qualitative non-functional property. For example, for *security* property which contains *low, medium,* and *high* qualifier tags, three domain predicates *SecurityLow, SecurityMed,* and *SecurityHigh* are generated. Also, for each quantitative non-functional property a function is defined. Figure 6.3 shows the generated functions

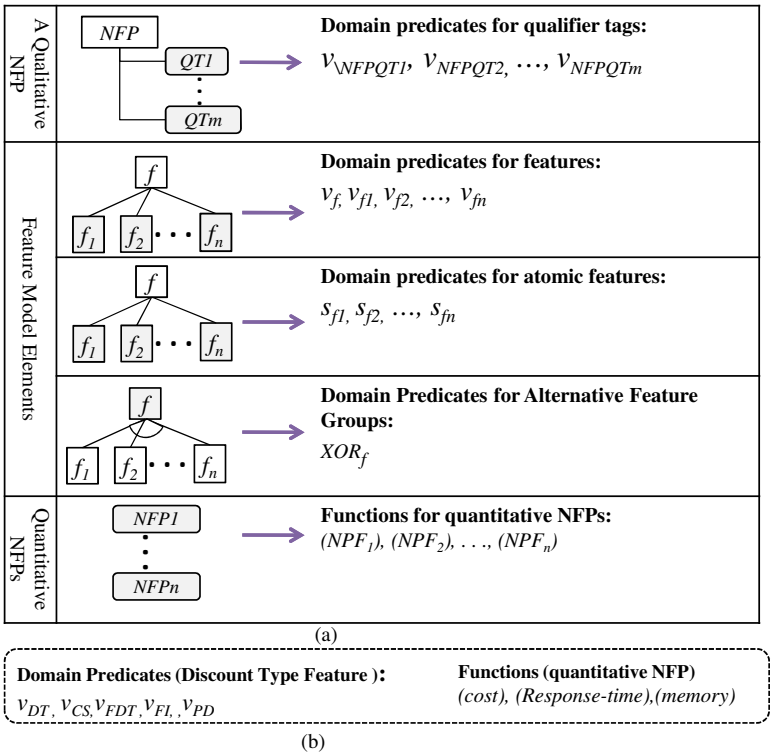


Figure 6.3: Generated domain predicates from feature models

for *cost, response time* and *memory*. In addition to non-functional properties, feature model relations (i.e., hierarchical relations and integrity constraints) affect the selection of features. For example, from an alternative feature group only one child feature must be selected for the configuration. In order to consider the feature model relations during the configuration process, we need to formulate these relations in terms of actions preconditions (see generating actions section). To this end, a domain predicate corresponding to each feature in the feature model (see Figure 6.3) is generated.

Generating actions. Actions represent the operations that can be performed in a domain. We transfer features in the feature model into actions in the PDDL domain. One action is generated for each feature. Figure 6.4 shows the transformation rules for generating action definitions based on the attributes of atomic features. If a feature is annotated with a specific qualifier tag of a qualitative non-functional property, the corresponding domain predicate is added to the precondition of the action. For example,

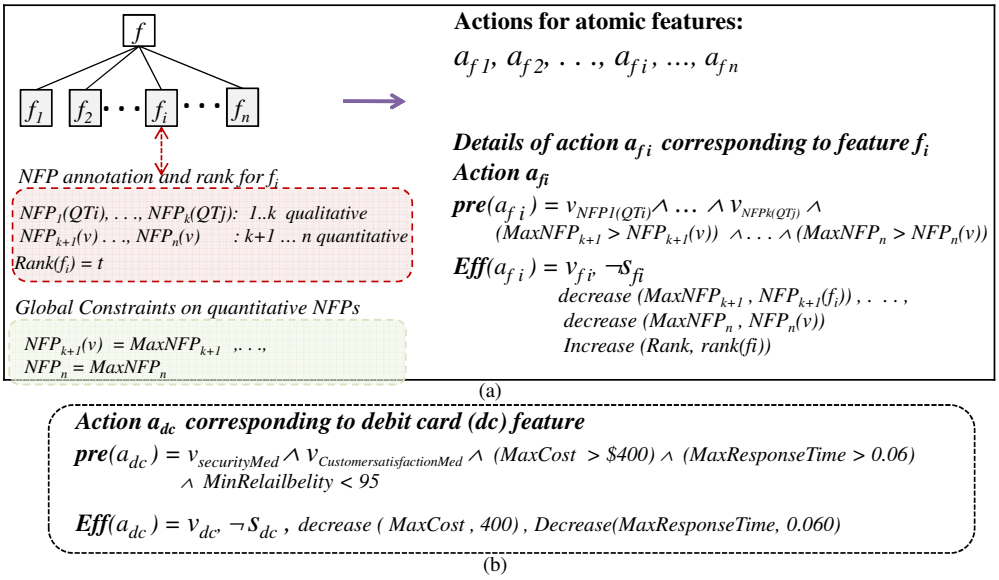


Figure 6.4: Generated actions for atomic features

in Figure 6.4(b), domain predicates $v_{securityMed}, v_{customersatisfactionMed}$ are added to the precondition of action a_{dc}, because the credit card feature is annotated with medium *security* and medium *customer satisfaction*. For quantitative non-functional properties, if a feature is annotated with a specific value for a non-functional property (i.e. $NFP_n(v)$), we add a domain predicate which compares the value of the function corresponds to the non-functional property with the value of non-functional property for the given feature (i.e. $(NFP_n > NFP_n(v))$). For example, *cost* for credit card feature equals to 400$, hence we add $(cost > 400)$ which implies that the value of the cost function should be at least 400$ to select the action a_{dc} for the current plan. In order to ensure that the action (a_{f_i}) corresponding to an atomic feature (f_i) is selected only one time in the plan, we define a domain predicate (s_{f_i}) which initially is set to true. We add s_{f_i} into precondition of f_i to ensure this action is selected for the plan only once.

Effects are used to change the state of the domain after executing the actions. Therefore, we use the effect of actions to update functions and domain predicates corresponding to features. For all quantitative non-functional properties, we update the corresponding function by decreasing the current value of the function by the non-functional value for the current feature. For example, the value of cost function is updated in the effect of

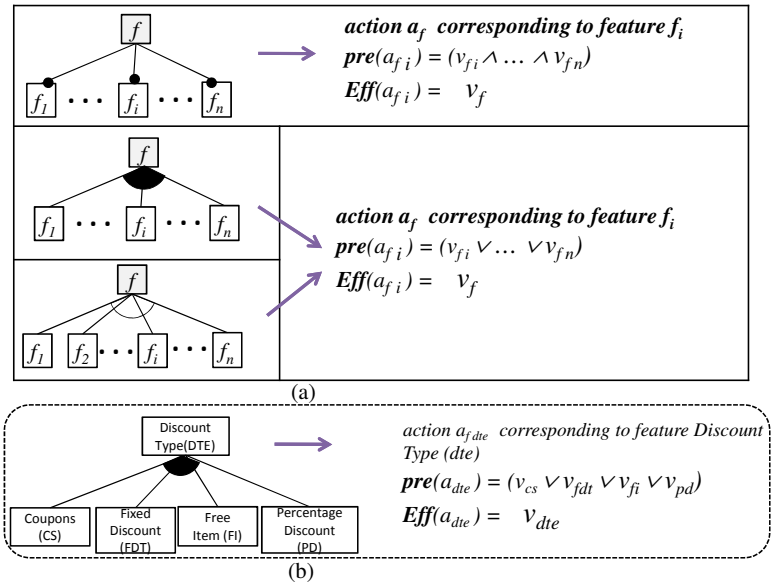


Figure 6.5: Generated actions for intermediate features

a_{dc} (see figure 6.4). Also, the domain predicate corresponding to a feature (i.e. v_{f_i}) is set to true. In the effect of the action s_{f_i} domain predicate is set to false to prevent multiple selections of the same action by a planner. Finally, the rank function is updated with the computed rank of each atomic feature.

An action is created for each intermediate feature. Figure 6.5 shows the transformation rules for different types of intermediate features (i.e. AND, OR, and alternative). For an AND feature group, the precondition is logical AND of domain predicates corresponding to child features. If the AND feature group includes optional features, we do not include the domain predicates corresponding to optional features into the precondition of the parent feature. For OR and alternative feature groups, the precondition of the action is logical OR of the domain predicates correspond to the child features. For example, as shown in the figure 6.5(a), the precondition of the action a_{dte} is $pre(a_{dte}) = (v_{cs} \vee v_{ss} \vee v_{fi} \vee v_{pd})$ which is logical OR of the domain predicates corresponding to Coupons, Fixed Discount, Free item, and Percentage Discount. In the effect of the action a_{dte}, we set the corresponding domain predicate to true ($v_{dte} = true$). The transformation rules in the figure 6.5 do not distinguish between alternative and OR feature groups. In order to ensure that exactly one child feature is selected in alternative feature group we define

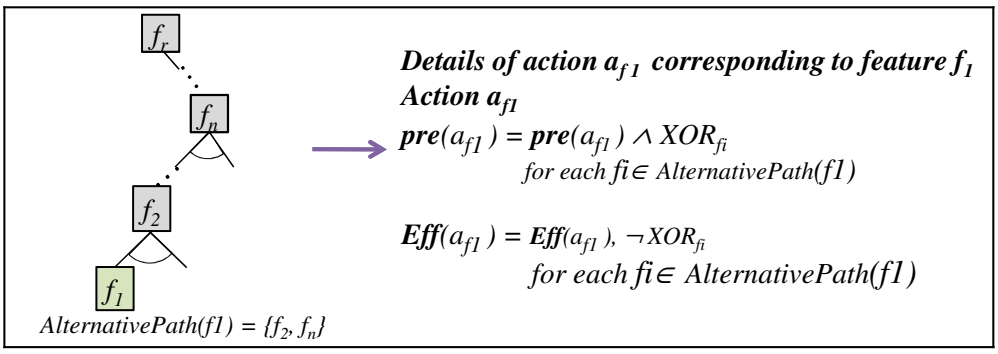


Figure 6.6: An example of alternative path and generated PDDL constructs

a transformation rule which is based on alternative paths. We define an alternative path as follow:

Definition 6.3 ***(Alternative path)***. *An alternative path for an atomic feature f is a set of all alternative features groups from the atomic feature to root of the feature model.*

For example, Figure 6.6 shows the alternative path for f_1 feature ($AlternativePath(f_1) = \{f_2, f_n\}$). Previously, we defined a domain predicate for each alternative feature group (see figure 6.3). Now, to satisfy the alternative constraints, we compute the alternative path for all atomic features and add domain predicates corresponding to alternative feature groups to the precondition of the atomic features. For example, as shown in Figure 6.6, the precondition of action a_{f_1} corresponding to feature f_1 is updated as $pre(a_{f_1}) = pre(a_{f_1}) \wedge XOR_{f_2} \wedge XOR_{f_n}$. Also, the effect of the action a_{f_1} is updated as $eff(a_{f_1}) = eff(a_{f_1}), \neg XOR_{f_2}, \neg XOR_{f_n}$. By using these transformation rules, we make sure that only one child of each alternative feature group will be selected.

In addition to hierarchical constraints in the feature model, we need to transfer integrity constraints into PDDL domain. Figure 6.7 shows the transformation rules for generating suitable action preconditions based on the integrity constraints between features. As we mentioned a feature may *require* or *exclude* a set of features. When a feature f_i requires a set of feature defined by a logical expression over the features(i.e. $\Phi(f_1, \ldots, f_n)$), the precondition of the action a_{f_i} is updated with the corresponding logical expression of domain predicates (i.e. $pre(a_{f_i}) = pre(a_{f_i}) \wedge \Phi(v_{f_1}, \ldots, v_{f_n})$). For exclude relation, a precondition of the action a_{f_i} is updated by a negation of the corresponding logical

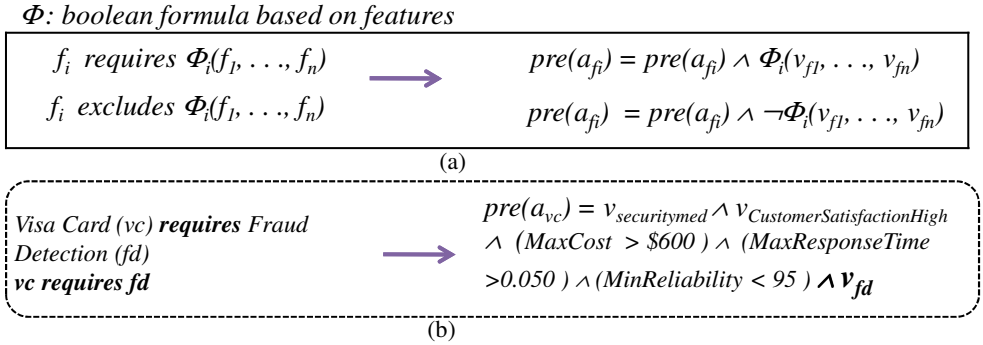


Figure 6.7: Transformation rules for integrity constraints

expression of domain predicates (i.e. $pre(a_{f_i}) = pre(a_{f_i}) \wedge \neg\Phi(v_{f_1}, \ldots, v_{f_n})$).

Problem Transformation

Problem transformation implies generating PDDL problem description based on the constraints over non-function properties. Problem generation includes two main parts: *generating initial state* and *generating goal*.

Generating initial state: In section 6.5.1, we captured the stakeholders' requirements over non-functional properties in terms of constraints over non-functional properties. For each quantitative non-functional property, we generate an initial state where we assign the requested value for the non-functional property by stakeholders to the function corresponding to the non-functional property. For example, if a stakeholder defines that the maximum cost should be less than 2000$, we add the following statement to the initial states ($(= cost 2000)$). For qualitative non-functional properties, if a stakeholder selects a qualifier tag for a non-functional property, the domain predicate corresponding to the qualifier tag and all higher tags are set to true. For example, if a stakeholder asks for medium *customer satisfaction*, we add the states ($customersatisfactionMed$ and $customersatisfactionHigh$). In order to enable the selection of atomic features, we set the domain predicates corresponding to the atomic features (i.e. s_{f_i}) to true. Also, the domain predicates corresponding to alternative feature groups (i.e. XOR_f) are set to true to enable the selection of at least on alternative child for each feature group. Finally, we set the *Rank* function equals to 0.

Generating goal state and objective function: The objective of the configuration process is to select a subset of features in the feature model that satisfies the feature model and non-functional constraints as well as optimizes stakeholders' preferences. Hence, the goal for planning should ensure that the generated plan satisfies the constraints and optimize the ranks assigned to the features. In the domain definition, the precondition of an intermediate feature was formulated based on the relation defined over its children (see figure 6.5) and non-functional constraints. Also in the initial states, we set the domain predicates of the atomic features (i.e. s_{f_i}) to true. Therefore, by setting the selection of the action corresponding to the root feature as a goal state (i.e. : $goal(v_{f_{root}})$), we ensure that the generated plan satisfies both feature model and non-functional constraints. By employing the S-AHP algorithm (see section 6.5.2), we computed the ranks of features based on the preferences of stakeholders. Hence, we define maximizing rank function as a metric.

Identifying the Optimal Plan

After generating the planning domain and planning problem, we employ an existing planner to find an optimal plan. There are several planners which fits for different planning types[2].We use LPG planner which can solve the numeric optimization planning problems. This planner applies different search heuristics to find the plan that satisfies the constraints defined in a domain and reach the objective defined in a planning problem. After the planner generated plan(s), we parse the output of the planner and according to the mapping between features and actions, we select the corresponding features from the feature model.

6.5.4 Deriving Configured Process Model

Features in a feature model are mapped into activities in a configurable process model using presence conditions(see section 6.4). After configuring the feature model by employing S-AHP and planning technique, we instantiate the configurable process model to achieve a stakeholder's fitted process model. Process instantiation is performed in three main steps: evaluating the presence conditions, removing elements from the configurable process model, and post processing. According to the selected and deselected features

[2]International Planning Competition - http://ipc.informatik.uni-freiburg.de/

corresponding presence condition values are computed. Next, activities annotated with presence conditions equal to false are removed from the configurable process model. Further refinements are applied by utilizing the techniques such as *simplification* proposed by Czarnezki et al. [33]. For example, gateways (e.g. OR, XOR) with only one branch are removed from the process model. In the running example, if fraud detection feature is deselected, then fraud detection subprocess is removed from the configurable process model along with input and output flow links. Next, simplification step removes the AND gateway because it only has one branch.

6.6 Evaluation

We aim at evaluating whether the configuration of process models can be performed in a reasonable amount of time. To study the performance and scalability of the configuration algorithm, we derive three research questions:

- RQ1: How does the execution time of the configuration algorithm scale-up as the size of configurable process model increases and variability percentages change?
- RQ2: Does the percentages of integrity constraints have significant impact on the running time of the configuration algorithm?
- RQ3: Does the type of non-functional constraints have significant influence on the running time of the configuration algorithm?

In order to answer these research questions, we conducted a set of experiments and analyzed the results of the evaluations.

6.6.1 Experiment Setting

In order to evaluate our configuration algorithm for several situations, we applied the simulation modeling technique by following guidelines similar to those proposed in [76]. We selected the simulation technique as it is commonly employed in the context of configuration [183, 62, 116, 155]. Moreover, using simulation techniques, we can explore more situations and collect the required data to compare different situations and answer our research questions.

Table 6.2: Independent variables and their values

Size [Features]	variability %	Integrity Cons.%	NFP constraints
[200-1000]	[25, 50, 75]	[10, 20, 30]	[.50, .66, 1] [H-M, L-M, L]

Since the configuration is performed on the feature models, we only generate feature models. To generate feature models with different characteristics (e.g., the number of features, probability of mandatory and optional features, probability of OR and XOR groups, and percentage of integrity constraints), we adapted Betty FM Generator[3], which enables the random generation of highly-customized feature models [147]. The starting point for our analysis and the first independent variable is the *number of features* which corresponds to the number of activities in configurable process models. According to our investigation on existing configurable and reference process models in the literature [101] such as the SAP reference process model and the size of process models of our industrial partner, we selected the range of features from 200 to 1000 features. The second independent variable that we consider in our analysis is the *percentages of variability* in the process models which is distributed from 25%, 50%, and 75%, which means that 25, 50, and 75 precents of activites are variable in configurable process models. When formulating these variabilities in feature models, we equally distribute the variability into OR, alternative and optional features in the feature models. This range of variability distribution is selected to observe the impacts of diverse variability distributions on the execution time of planning. Based on our analysis on SPLOT repository[4], which shows an average of 18% of integrity constraints for real feature models [18], we considered 10%, 20%, and 30% distributions of integrity constraints in the feature models.

The results of an investigation on non-functional properties by Mairiza et al. [98] show that the number of relevant non-functional properties for different application domains is at most 11. Moreover, Sommerville and Sawyer [156] highlighted that the effective number of non-functional properties is around six. Considering these two studies,

[3]Betty Feature Model Generator Version 1.1, http://www.isa.us.es/betty

[4]SPLOT - Software Product Line On-line Tools, http://www.splot-research.org

we defined 6 non-functional properties; three quantitative; and three qualitative non-functional properties. Next, for each qualitative non-functional property three qualifier tags were defined. After defining the non-functional properties, we annotated features with non-functional properties. For quantitative non-functional properties, we applied a random function with normal distribution to produce the non-functional values for each feature. For qualitative non-functional properties, we randomly annotated features with 0 to 3 qualitative non-functional properties with normal distributions, that is, most of the features had one or two non-functional properties. In order to examine the impact of non-functional constraints, we defined *constraints type* as an independent variable which shows three cases: 1) no constraints; 2) expected value of 2/3 of maximum value of a quantitative non-functional and *medium* and *low* qualifier tags for qualitative non-functional; 3) and expected value of 1/2 of maximum value of a quantitative non-functional and *medium* and *high* qualifier tags for qualitative non-functional.

The evaluation was performed on a computer with an Intel Core DUO 2.2 GHZ CPU, 4GB of RAM, Linux, Java Runtime Environment v5.0, and LPG v1.0. We set the timeout for the planner 3600 seconds.

6.6.2 Experiments Results

The results of experiments are illustrated in Figures 6.8-6.11. The results show that for the large size of process models (i.e. 1000) with high percentages of variability the configuration technique can find the solution in a reasonable time. In the following, we discuss the details of the results.

RQ1: Size and variability percentages - Figure 6.8 depicts that as the size of feature models increases the running time increases exponentially. However, the average running time for the models with 1000 features is 153.76 which shows practicality of the approach for the large size process models. According to Figure 6.9, the variability percentage has a significant impact on increasing the running time for finding an optimal solution. The result were expected because when the variability percentage increases, the number of possible selection options increases and the planning algorithm needs to explore a larger solution space. Interestingly, the results reveal that the variability has higher impacts than the model size in which the average time for a model with 600 elements and 75% variability has higher running time than models with the higher number of elements but lower variability percentages.

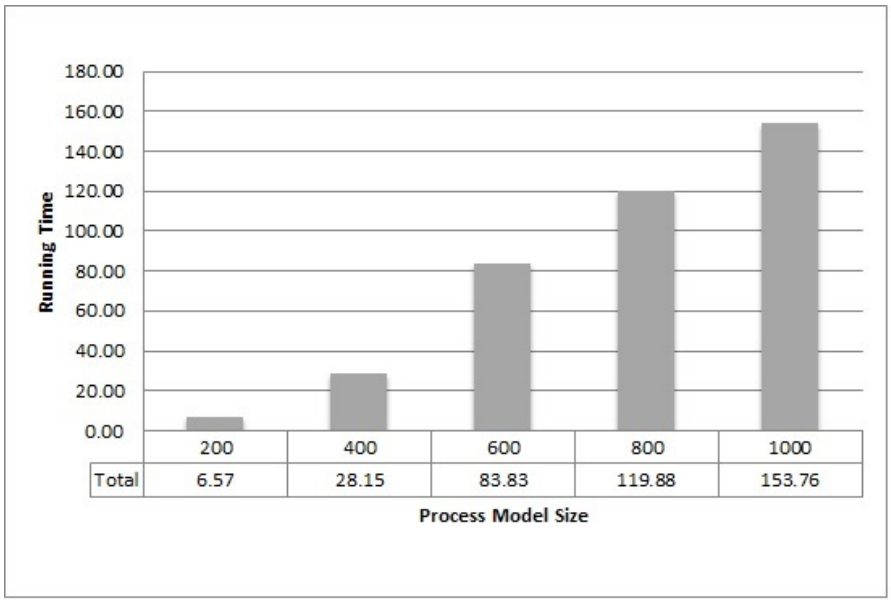


Figure 6.8: Running time for different process sizes

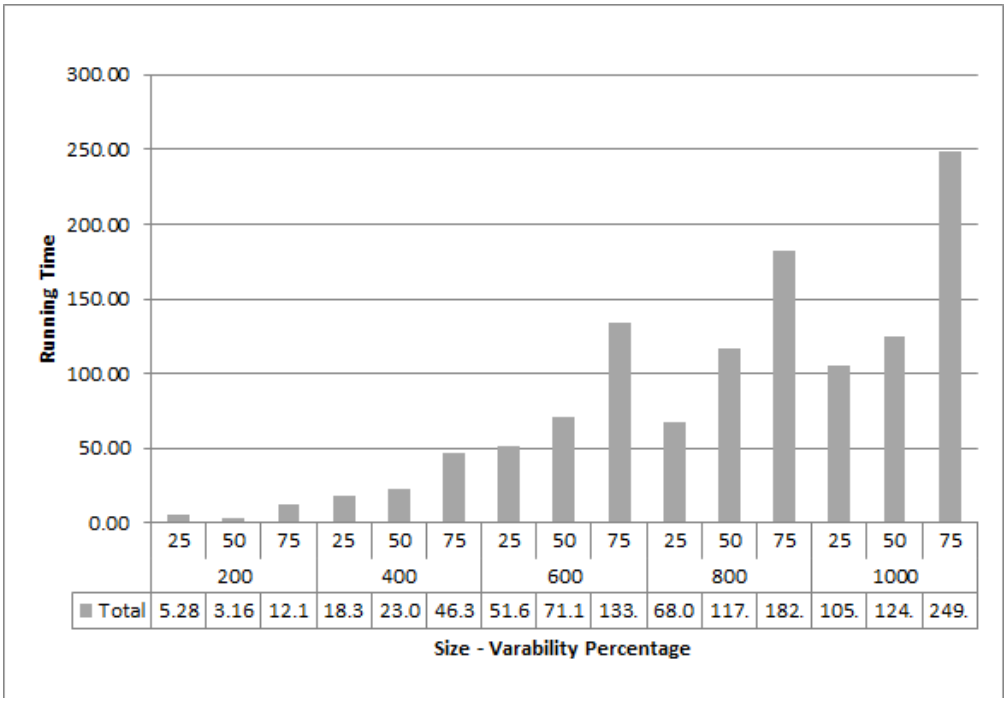


Figure 6.9: Running time for different sizes and variability distributions

RQ2: Integrity constraints percentages - Figure 6.10 shows the average running time for different integrity percentages. The results show that the average running time increases with the increase of the integrity constraints percentages. However, further investigation of size and integrity constraints reveals that the behavior of the planning algorithm varies in different sizes of models. The increase of the integrity constraints reduces the configuration space but increases the complexity of preconditions.

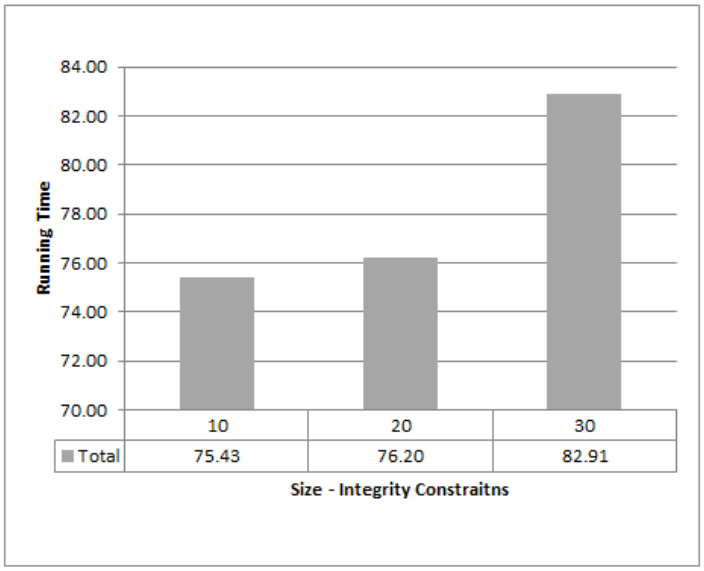
RQ3: Non-functional constraints - Figure 6.11 shows the results for different types of constraints. According to the results, having more constraints (i.e. higher qualitative tags and lower maximum quantitative non-functional values) decreases the running time for finding the optimal solution. This is due to eliminating many actions whose precondition are not satisfied and consequently reducing the search space for finding the optimal solution.

6.6.3 Threats to Validity

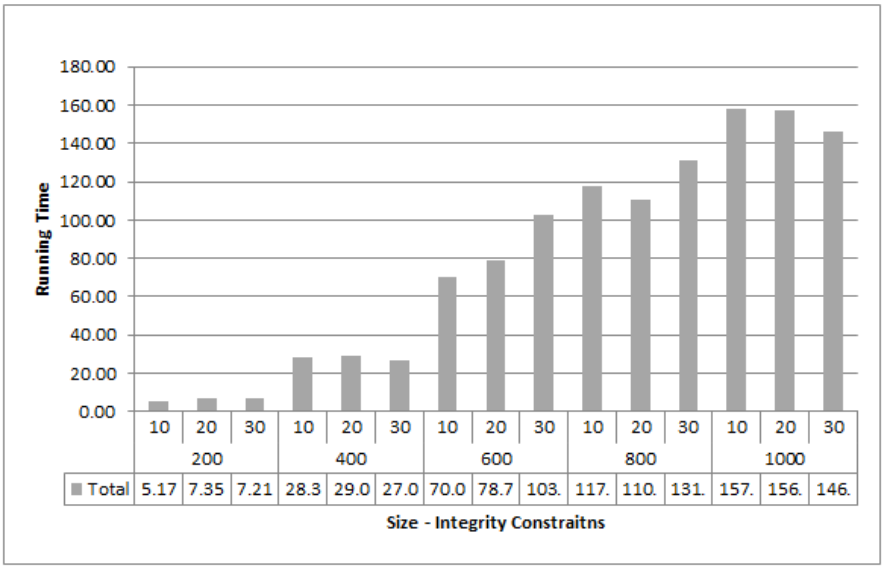
When evaluating a technique using an experimental framework, the results are always subject to different threats including internal and external threats. In this section, we explore the major threats to the validity of our experimental results.

The confounding variables (i.e., variables which are not considered as independent variables and might have impacts on the dependent variable) forms the major concern in internal validity. Therefore, in our experiments we controlled the confounding variables to ensure the internal validity of our results. When we were investigating the impact of independent variables related to feature models (e.g., size of feature models, variability percentages, and integrity constraints percentages) on the running time, we controlled impacts of the non-functional constraints(see figures 6.8, 6.9, 6.10). Similarly, we controlled the feature models characteristics (see figure 6.11) when we were investigating the impact of constraints over non-functional properties. We also generated 100 samples that reduces the impact of the random generations of the models.

In order to ensure the external validity of our experiments, we need to show that the results are generalizable. We identified two main factors influencing the external validity of our experiments: 1) the characteristics of process models and 2) the number of non-functional properties. With respect to the size of process models (feature models), our investigation over industrial process models (e.g., SAP reference process) and observation of our case-study confirmed that the sizes of generated models are aligned with



(a)



(b)

Figure 6.10: Running time for different sizes and integrity constraints

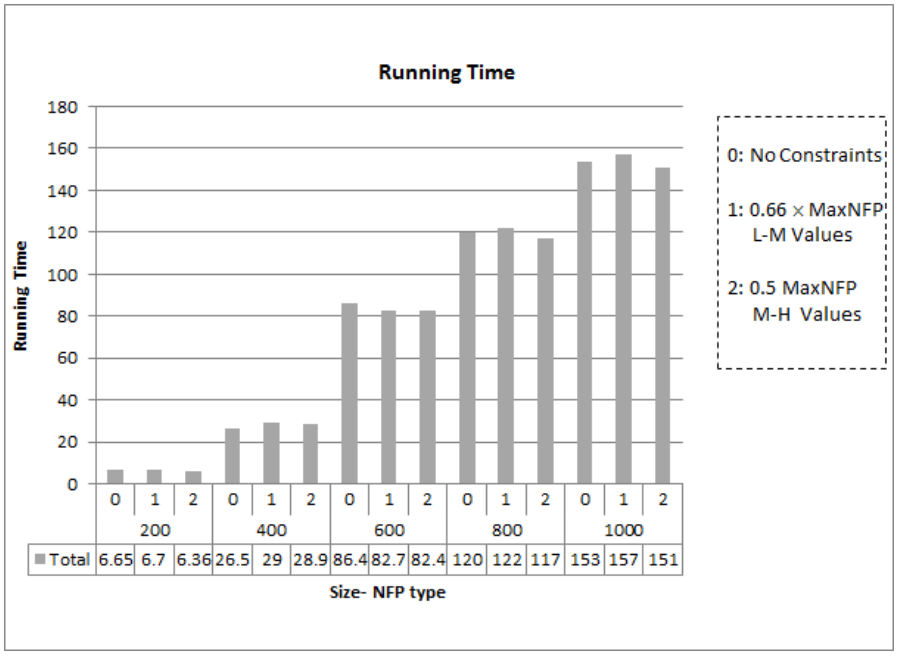


Figure 6.11: Running time for different sizes and non-functional constraints

the size of existing industrial models. The distributions of integrity constraints and variability relations in the feature models backed up by our analysis over SPLOT repository and existing studies [62] on the characteristics of real feature models. Finally, regarding the number of non-functional properties, the existing studies by Mairiza et al. [98] and Sommerville and Sawyer [156] showed that the number of relevant non-functional properties for different application domains is at most 11 and the effective number of non-functional properties is around 6.

6.7 Related Work

La Rosa et al. [136] surveyed configurable process model approaches. According to their results only two approaches, C-YAWL [55] and C-EPC [137], provide techniques to derive a process model from a configurable process model. On the other hand, many techniques have been proposed in the software product line engineering which provide the automatic configuration of feature models.

All automatic configuration techniques transfer the configuration problem into another representation (e.g., planning) and use existing techniques to solve the problem. In order to compare our approach with the existing configuration approaches, we organized these approaches based on the techniques used for the configuration. The techniques are: *questionnaire models, constraint satisfaction problem (CSP), approximation, genetic algorithm, analytical hierarchy process,* and *planning.*

Questionnaire model: These approaches represent the configuration in terms of a questionnaire model in which domain facts are organized into questions. These questions are posed to stakeholders to configure configurable process models. Questions group domain facts based on their content [136]. In [55], Gottschalk et al. applied questionnaire based approach for deriving a process model from workflow models. Similarly, La Rosa et al. [84] employed the questionnaire approach to derive EPC models from C-EPCs. These approaches are interactive and need stakeholders' inputs for binding each variation point or a group of variation points.

CSP models: These approaches convert the configuration problem into a CSP problem and use existing CSP solvers to find an optimal configuration. An initial approach was proposed by Benavides et al. [23] in which an extended feature model (i.e., feature models with extra-functional features) was transferred into a CSP model. Using their extension, they were able to assign extra-functionalities such as price range or time range to features. The purpose of their technique is to find a configuration that satisfies the given constraints. Also, SPL conqueror [148], developed by Siegmund et al., extends feature models with qualitative and quantitative non-functional properties and applies CSP to find an optimal configuration based on a user defined objective function. SPL conqueror differs from Benavidas' approach on types of non-functional properties they manage (i.e. both qualitative and quantitative non-functional properties are taken into account.) White et al. [184] also formalized the stage configuration and introduced a Multi-step Software Configuration probLEm solver (MUSCLE) in which they used CSPs for find optimal configuration in each stage. Non-functional properties such as cost constraints between two configurations are formalized as CSP constraints. The approach is only applicable for multi-stage configuration and concentrates on creating new configurations from an already derived product configuration.

Approximation techniques: These approaches transfer the configuration problem into a well-known optimization problem and apply existing approximation techniques to

find the solution. White et al. [183] used a Filtered Cartesian Flattening (FCF) method to select optimal feature sets according to the resource constraints. In their method, they map the feature selection problem to a multi-dimensional multi-choice knapsack problem (MMKP) [183]. By applying the existing MMKP approximation algorithms, they provide partially optimal feature configurations in polynomial time.

Genetic algorithm: These approaches employ genetic algorithms to find the configuration. In [62], Guo et al. first randomly generate a string with n binary digits (which represents a chromosome so that a value of 1 and 0 for a digit indicates the selected and deselected feature, respectively). Next, the arbitrary feature selection is transformed into a valid feature selection by applying FesTransform (Feature selection Transform) algorithm. The results of their empirical study show that their proposed algorithm can achieve an average of $86 - 90\%$ optimality for feature selection. Ognjanovic et al. [116] represented the variability of process models in feature models and employed the genetic algorithm to generate a configured process model. Their algorithm is similar to Guo et al. [62]. They can handle more complex non-functional properties and preferences (i.e., relative importance).

Analytical hierarchy process:. These approaches apply analytical hierarchy process techniques to configure feature models based on pair comparison of the criteria. Zhang et al. [193] proposed an AHP based method for quality aware configuration of feature models. They considered quality attributes as features in feature models and defined interdependency relation between functional features and non-functional features. Having formed the groups of functional features (also called contributors) for each quality attribute based on interdependency relations, they employed Analytical Hierarchy Process (AHP) to calculate the relative importance of contributors of each quality attribute. Bagheri et al. [17] also applied the AHP algorithm for computing rank of the features based on the preferences of stakeholders over business concerns.

Planning: These approaches transfer a configuration problem into a planning problem and employ the existing planners to find an optimal configuration. Soltani et al. [155] applied Hierarchical Task Network planning technique and used SHOP2 planner to find the configuration. They take into account qualitative and quantitative non-functional properties, preferences, and constraints. Our approach belongs to this category and applies a PDDL based technique to find an optimal solution. Similar to the approach proposed by Soltani et al. [155], we handle the same types of non-functional properties,

preference, and constraints.

6.7.1 Comparison with Related Works

In order to evaluate our approach we employ a criteria-based comparison technique where we use a set of criteria for systematically comparing our approach with existing approaches.

Similar to [143, 136, 155], these criteria were collected by applying top-down and bottom-up approaches. Following the top-down approach, we used an existing survey on configuration of configurable process models [136] and feature model configuration literature. Following the bottom-up approach, we identified various important aspects of process model configuration in the description of existing related work and added them to the criteria set. We do not claim that this criteria set is complete, but it provides appropriate aspects to compare our work with related work.

These criteria include: 1) Variability representation strategy (implicit vs explicit), 2) Variability patterns (optional, OR, alternative), 3) Managing functional and non-functional requirements; 4) Modeling stakeholders' preferences; 5) Optimization; 6) Considering stakeholder constraints; 7) Providing tooling support; 8) Automating configuration process; 9) Ensuring the feature model constraints; 10) Effective representation of results to stakeholders; 9) Time efficiency. The results are shown in Table 6.3[5].

With respect to variability, all feature driven approaches (including our approach) use the explicit representation for variability modeling and cover all variability patterns. This helps mitigate modeling complexity by separating the variability aspect from the process aspect. C-YAWL and C-EPC represent the variability in process models and only cover optional variability. They also lack mechanisms for representing selection dependencies (i.e. integrity constraints) between activities.

All the approaches completely address functional requirements, but the coverage of non-functional requirements varies. The questionnaire based approaches i.e. C-YAWL and C-EPC do not cover non-functional requirements. Benavides [23], MUSCLE[184], GAFES [62], and FCF [183] support the configuration only based on quantitative non-functional requirements, Zhang et al. [193] and Bagheri et al. [17] generate the configuration only based on qualitative non-functional requirements. SPL conqueror [148],

[5]The results of the comparison is an extension of our previous evaluation of related work in [155]

Table 6.3: Comparative analysis of related work ((+) criterion met, (-) criterion not met, (+/-) criterion partially met), expl. explicit, impl. implicit

Approach \ Criteria	Representation	Variability Patterns	FR/NFR	Preference	Constraints	Automation	Integrity constraints	Tooling support	Optimization	Time efficiency
C-YAWL - Gottschalk et al. [55]	impl.	+/-	+/-	–	–	+/-	–	+	–	– (no result was reported)
C-EPC La Rosa et al. [137]	impl.	+/-	+/-	–	–	+/-	–	+	–	– (no result was reported)
CSP - Benavides et al. [23]	expl.	+	+/-	–	+	+	–	+	+	– (up to 52 features)
Conqueror Siegmund et al. [148]	expl.	+	+	+/-	+	+	+	+	+	+/- (no result was reported)
MUSCLE White et al. [184]	expl.	+	+/-	+/-	+	+	+	-	+	+/- (up to 500 features)
FCF White et al. [183]	expl.	+	+/-	+/-	+	+	+	-	(90%)	+ (up to 10,000 features)
GAFES Guo et al. [62]	expl.	+	+/-	+/-	+	+	+	+	(90%)	+ (up to 10,000 features)
Ognjanovic et al. [116]	expl.	+	+	+	+	+	+	–	(90%)	+/-(up to 287 features)
Zhang et al. [193]	expl.	+	+	+	+/-	+	+	+	+/-	– (no result was reported)
Bagheri et al. [17]	expl.	+	+/-	+	–	–	–	+	+/-	– (no result was reported)
Soltani et al. [155]	expl.	+	+	+	+	+	+	+	+	+/- (up to 250 features)
Our approach	expl.	+	+	+	+	+	+	+	+	+ (up to 1000 features)

Ognjanovic et al. [116], Soltani et al. [155], and our approach handle both quantitative and qualitative non-functional requirements.

With respect to preferences over non-functional properties and optimization, the studied approaches capture preferences either using a user defined objective function [23, 148, 184, 183, 62] or employing a relative importance [17, 155, 193]. The later approaches define some utility function to optimize the stakeholders' preferences. Our approach belongs to the second category. Questionnaire based approaches i.e., C-EPC and C-YAWL do not address optimization and preference capturing.

Stakeholders' constraints over quantitative non-functional properties are supported by FCF [183], GAFES [62], Ognjanovic et al. [116], CSP based approaches [23, 148, 184], Zhang et al. [193], and Soltani et al. [155]. Constraint over qualitative non-functional properties are addressed by Soltani et al. [155], SPL conqueror [148], Ognjanovic et al. [116]. Our approach supports constraints over both functional and

non-functional properties.

All the approaches, except FCF and Ognjanovic et al. [116], provide tooling support. All approaches, except Benavides et al. [23], Bagheri et al. [17], C-YAWL [55], and C-EPC [137] ensure the satisfaction of integrity constraints (i.e., requires and excludes relations) and automatically generate configuration.

6.8 Conclusions

Several techniques have been proposed to manage variability in configurable process models. In this article, we proposed a novel technique which 1) separates the variability aspect of configurable process models and formulates it in feature models; 2) applies an annotation based mapping mechanism for linking the process elements (activities and sub-processes) to the features; 3) provides a configuration process and algorithms to derive a configured process model from a configurable process model based on the requirements of the target application. To this end, a decision making technique (Stratified Analytical Hierarchy Process (S-AHP)) is utilized and a model transformation technique is devised to transfer feature model into PDDL planning problem. The experiment results demonstrate the practicality of the approach for large size process models and criteria based comparison reveals the improvement of the approach with respect to the existing approaches.

Chapter 7

Goal Model and Process Model Validation

This chapter consists of "Validation of user intentions in process orchestration and choreography" paper which was published in *Information Systems Journal*.[1] The paper introduces an approach for developing a customizable process model and mapping it to a family goal model. The possible inconsistencies patterns between a family goal model and the customizable process model are identified and description logic based technique is employed to check inconsistencies. Several experiments have been done to evaluate the performance efficiency of the approach.

Author's role - I was the main contributor in developing the validation technique between goal models and process models. I identified the inconsistency patterns (orchestration and choreography). The case study, design of experiments, implementing the experiments, analysis of the results, and developing parts of description logic algorithms has been done by the author. I wrote all the sections of the paper except sections 5, 6, and 7.

[1] The paper reprinted (with minor adjustment to formatting), with permission from Elsevier publisher.

Groner, G., **Asadi, M.**, Mobabbati, B., Gasevic, D., Boskovic, M. (2014) Validation of User Intentions in Process Orchestration and Choreography, Information Systems Journal, Vol. 43, No 7, pp. 83-99, in press.

7.1 Abstract

Goal models and business process models are complementary artifacts for capturing the requirements and their execution flow in software engineering. In this case, goal models serve as input for designing business process models. This requires mappings between both types of models in order to describe which user goals are implemented by which activities in a business process. Due to the large number of possible relationships among goals in the goal model and possible control flows of activities, developers struggle with the challenge of maintaining consistent configurations of both models and their mappings. Managing these mappings manually is error-prone. In our work, we propose an automated solution that relies on Description Logics and automated reasoners for validating mappings that describe the realization of goals by activities in business process models. The results are the identification of two inconsistency patterns – orchestration inconsistency and choreography inconsistency – and the development of the corresponding algorithms for detecting these inconsistencies.

7.2 Introduction

With the growing importance of process-aware information systems (PAISs), business process modeling has gained a significant research attention [43]. A business process model is an operational representation of activities, their ordering and routing in order to achieve goals; that is, delivering services to customers. However, as outlined in [127], the reasons and rationales how the execution of a process and the corresponding tasks satisfy the requirements of PAISs are often not captured within business process models.

Requirements engineering offers proven means for understanding user intentions. Goal-oriented modeling is a prominent formalism to describe requirements of a system in terms of goals (intentions) and relationships between goals. A goal describes a certain system functionality or property that should be achieved. Thus, it is not surprising that the recent research on PAISs has focused on the integration of business process modeling with goal-oriented modeling (cf. Section 7.8). Related research concentrates on

basic mapping and aligning principles between the intentional perspective, represented by goal models [79, 81, 141] or business models [8, 24] and the process model perspective. The main focus of these approaches is either on enriching a process model with goals and relationships between goals or on transformations from goal models to process models. However, less attention has been paid to the analysis and formalization of the influence of intentional relations, which reflect the user requirement perspective, on process model relations, which cover the process control flow and message exchange between processes.

In this paper, we tackle the challenge of formalizing the realization of intentions by activities in business processes. These realizations are described by mappings. Based on the formalization, we specify orchestration and choreography inconsistencies that might occur due to contradicting relationships between activities and their corresponding (mapped) intentions. In that sense, an automated validation of the mappings needs to assure the fulfillment of user goals in each execution configuration/path of business processes.

To address this challenge, our first contribution (Section 7.4) is the definition of types of inconsistencies between goal and process models. The definition of inconsistencies is based on correspondences between intentional relationships of the goal model and relationships in the process model, which are given by control flow relations (orchestration) in terms of workflow patterns [138] and by interaction relations (choreography) between processes. Next, based on these correspondences, we propose a formal modeling (Section 7.5) and validation (Section 7.6) approach in Description Logics. The focus of the modeling is to capture the relationships among activities of the business process model and between user intentions of the goal model. Sound and complete Description Logics based reasoning is used to ensure realization equivalence between user intentions and their corresponding (mapped) activities; that is, that there are no contradictions between relations of intentions compared to their mapped activities. Finally, we compare our approach to already available approaches for validation of correspondences between goal and process models (Section 7.8) and outline the conclusions (Section 7.9).

7.3 Methodology Overview

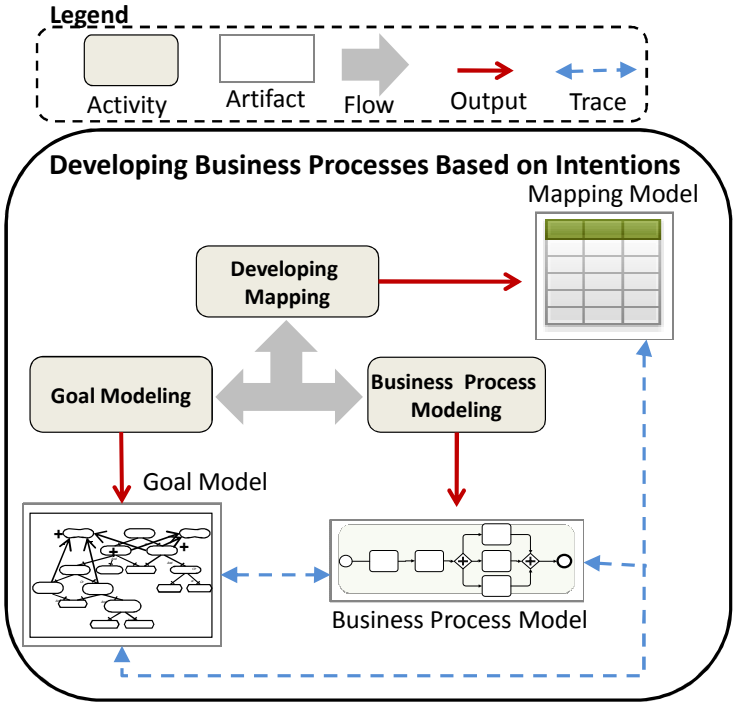
This section provides an overview of the proposed methodology and generated artifacts. This methodology consists of two subprocesses: (i) the *development of process models based on intentions* and (ii) the *validation*, as illustrated in Figure 7.1.

In the former subprocess (see Figure 7.1(a)), requirement engineers develop a goal model representing stakeholders' intentions and their refinement into tasks. This procedure starts with identifying the actors (e.g., Back Store and Front Store) of the system-to-be and their high level goals (e.g., Process Order) and soft goals (e.g., Minimize Risk). Actors dependencies are identified and modeled in the goal model. High level goals of actors are decomposed by following the framework proposed by Liaskos et al. [91] where intentional variability concerns are recognized for each high level goal. Then, these goals are refined according to the variability concerns. After refining goals, requirements engineers analyze the impacts of goals on other goals and model these impacts using contribution links. Afterwards, a process model is produced by defining activities and subprocesses that realize tasks of the goal model. Activities are organized in swim-lanes. Each lane refers to an internal actor in the goal model. According to the relations in the goal model, process orchestration and choreography relations are determined and represented. Simultaneously, mappings between tasks in the goal model and activities in the process model are devised. These models offer two different viewpoints (requirements on one side and process orchestration and choreography on the other side), covering artifacts and relationships among artefacts for different purposes.

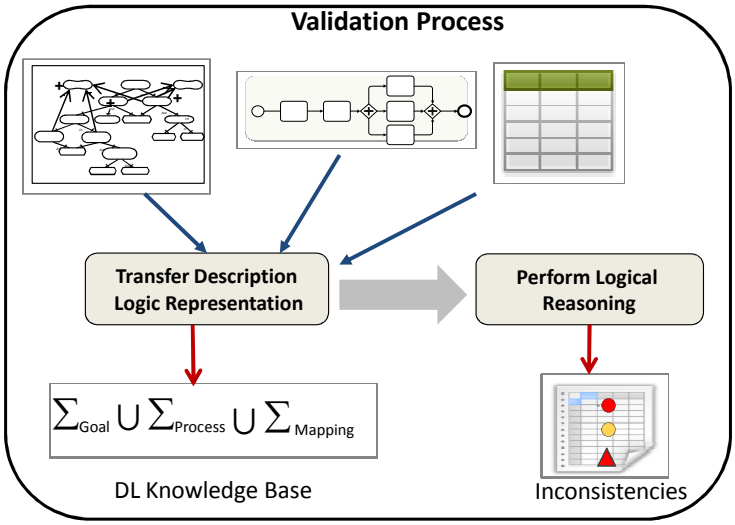
The second subprocess (see Figure 7.1(b)) identifies possible inconsistencies that may happen between relationships in the goal model and process models. As a first step, the goal model, the process model and the mapping model are transformed into a knowledge base in Description Logics. The knowledge base covers the relationships of elements from both models. Afterwards, the second step leverages reasoning services, which rely on the well defined Tarski-style semantics [160] of Description Logics, to recognize (possible) inconsistencies between goal models and process models. To this end, a subset of Description Logics constructs ($\mathcal{ALC}$ expressiveness) is used in our framework.

7.4 Realization Inconsistencies

Tasks in the goal model are those intentional elements that need to be performed in order to achieve the specified requirements. The basis for aligning business process models with goal models is to map *tasks* of the goal model to *activities* (atomic activities or subprocesses) in the business process model in order to express the implementation of a goal by an activity. We cover the following mapping rules between elements in the goal models and business process models, which follow basic mapping and realization



(a) Steps for developing the goal model, process models and mapping model



(b) Steps for detecting inconsistencies

Figure 7.1: Proposed methodology for developing process models based on user intentions and validating the process models and goal models relations.

principles in the literature [79, 75, 96]:

- *Actor mapping:* The actors within a goal model are mapped to parties that execute activities in the business process, i.e., to swim-lanes and pools. In our case, each pool contains one process.
  - Internal actors (actors that belong to the system-to-be) are mapped to the lanes of a pool. If a pool consists of multiple lanes the activities of a process (in this pool) might be distributed over the lanes of the pool.
  - External actors (actors that interact with the system-to-be) are mapped to pools.
- *Intentional element mapping:* Among the intentional elements, tasks are implemented by activities in a process. This is expressed by mappings.
  - Tasks in the goal model are mapped to the activities (either atomic or composite, i.e., sub-processes).

These mapping principles follow the modeling intuition of both models: (i) Actors in a goal model represent active entities that perform actions to achieve their goals. These goals are within the actor's scope. Likewise in the business process model, lanes group activities of a process and pools group processes that refer to different roles. (ii) Tasks in the goal model are achieved by executing activities (atomic or composite activities) in a process.

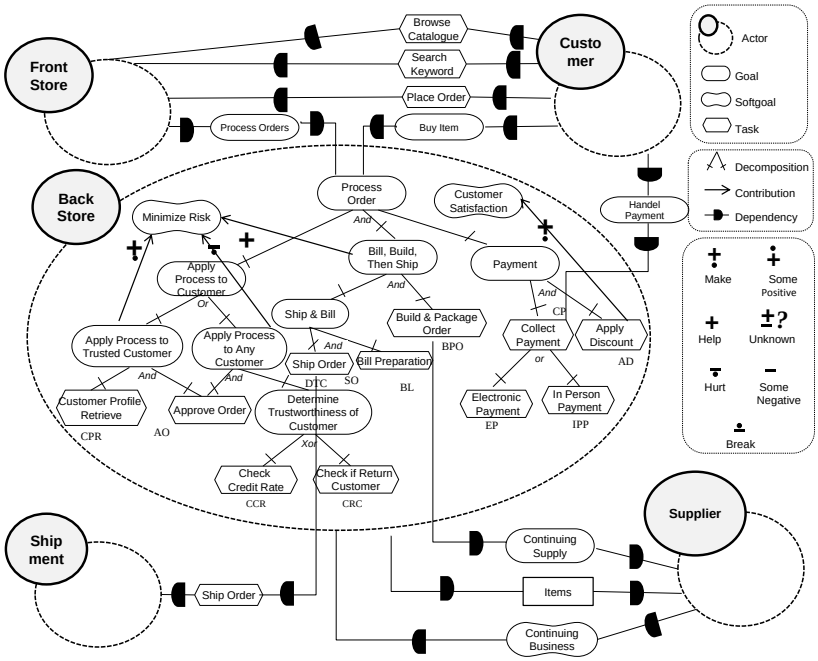
In Figure 7.2, mappings are represented by annotations of activities with the name of intentional elements. For instance, the activity Money Order is mapped to the intentional element In Person Payment (IPP).

When mappings are established, the process orchestration (ordering of activities) must be consistent with intentional relations between goals, and a process choreography (relations that are given by message exchange) must exist between pools and lanes in case there are dependencies between the corresponding actors.

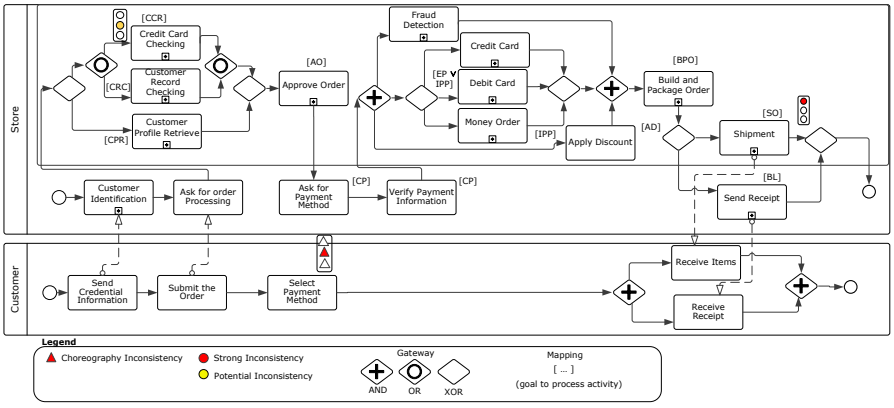
7.4.1 Orchestration Realization Inconsistencies

The process orchestration is described by control flow relations between activities, called workflow patterns[2]. Definition 7.1 specifies workflow patterns as a collection of all control

[2] http://www.workflowpatterns.com



(a) Goal Model of the E-Store



(b) Business Process Model of the E-Store

Figure 7.2: Goal Realizations in Business Process Models

flow relations an activity depends on.

Definition 7.1 (Workflow Patterns) *Let $PM = (\mathcal{A}, \mathcal{G}, \mathcal{E}, \mathcal{F}, \mathcal{E}_A)$ be a materialized business process model. We denote with WF_A the workflow relations of an activity $A \in \mathcal{A}$, specified on activities $A_1, \ldots, A_n \in \mathcal{A}$.*

The *workflow patterns* of a process describe activities and their execution ordering. The execution ordering is specified by a combination of the introduced modeling constructs: activities, gateways and edges (flow edges) between activities and gateways. The relations can be grouped into (i) basic control flow patterns, (ii) advanced branching and synchronization patterns, (iii) structural patterns, (iv) multiple instance patterns, (v) state-based patterns and (vi) cancellation patterns.

Basic control flow patterns cover the elementary aspects of a process like sequences, parallel and exclusive branching. Advanced branching and synchronization patterns offer more complex branching and merging constructs like multi-choice, discriminator and n-out-of-m joins. Structural patterns contain restrictions regarding the structure like loops with multiple entry and exit points and implicit termination. Multiple instance patterns represent processes where certain activities can have multiple instances (within one process instance). State-based patterns allow the specification that the process execution is determined by the state of the process instance at runtime. The cancellation pattern is used to describe possibilities and situations where the process execution can be terminated in certain circumstances.

According to the mapping principles,activities implement tasks of the goal model. These tasks might depend on other intentional elements in the goal model. This is described by intentional relations (Definition 7.2).

Definition 7.2 (Intentional Relations) *Let $GM = (\mathcal{A}, \mathcal{G}, \mathcal{P}, \mathcal{D}, \mathcal{C})$ be a goal model. IR_G denote the intentional relations of the intentional element $G \in \mathcal{G}$ on other intentional elements $G_1, \ldots, G_m \in \mathcal{G}$. Intentional relations are represented by decompositions $\mathcal{D}$, contributions $\mathcal{C}$ and dependencies $\mathcal{P}$ between intentional elements in the scope of different actors.*

According to the mapping rules, tasks in the scope of each actor are mapped to activities inside pools and lanes. Therefore, the execution relations between activities (i.e., workflow patterns) in business process models must be consistent with intentional

relations between intentional elements in goal models. Mappings "carry" intentional relations to the business process model. Thus, there might be inconsistencies between the relations given by the workflow pattern of an activity and the corresponding intentional relations of its mapped intentional element. We distinguish between two types of orchestration inconsistencies: 1) *strong inconsistency* and 2) *potential inconsistency* (cf. Definitions 7.3 and 7.4).

Definition 7.3 (Strong Inconsistency) *Assume a workflow pattern WF_A for an activity $A \in \mathcal{A}$ is specified over activities $A_1, \ldots, A_n \in \mathcal{A}$ and an intentional relation IR_G for $G \in \mathcal{G}$, defined over intentional elements $G_1, \ldots, G_m \in \mathcal{G}$. Activities $A, A_1, \ldots, A_n$ are realizations of intentional elements $G, G_1, \ldots, G_m$. A strong inconsistency between WF_A and IR_G occurs if there is no execution combination of activities that leads to the fulfillment of the intentional relation IR_G.*

A strong inconsistency says that for no execution that is expressed by workflow pattern/relation WF_A there is a goal fulfillment/achievement of IR_G possible. A weaker notion is the potential inconsistency. There can exist an allowed execution WF_A in which the corresponding intentional relation IR_G is not fulfilled.

Definition 7.4 (Potential Inconsistency) *Assume a workflow pattern WF_A for an activity $A \in \mathcal{A}$ is specified over activities $A_1, \ldots, A_n \in \mathcal{A}$ and an intentional relation IR_G for an intention G is defined over intentional elements $G_1, \ldots, G_m \in \mathcal{G}$. Activities $A, A_1, \ldots, A_n$ are realizations of intentional elements $G, G_1, \ldots, G_m$. We define a potential inconsistency between WF_A and IR_G if some execution combinations of activities lead to the fulfillment of the intentional relation IR_G and some execution combinations of activities do not lead to the fulfillment of IR_G.*

Both kinds of inconsistencies are recognized based on a set of existing mappings. Adding additional mappings might lead to further inconsistencies.

As our approach is applied for process engineering and modeling at design time, i.e., we consider process models rather than executions and execution observations, various patterns of the multiple instance, state-based and cancellation patterns are not affected by in our approach, since they deal with rather run-time related aspects that can not become inconsistent at design time. Table 7.1 summarizes the influence of intentional relations on workflow patterns, but it also provides a comprehensive overview how both

kinds of relations can be mapped to each other. The last column in Table 7.1 depicts the influence of dependencies in the goal model. In this case, we require that the corresponding intentions are in the scope of different actors.

The example in Figure 7.2 contains a strong and a potential inconsistency. The strong inconsistency is due to the activities Send Receipt and Shipment that are mapped to tasks Bill (BL) and Ship Order (SO), whereby these tasks are `AND`-siblings. Thus, each satisfaction of the target element Ship & Bill requires that both tasks Bill (BL) and Ship Order (SO) are fulfilled simultaneously, while each process execution allows either the execution of Send Bill or Shipment.

A potential inconsistency is caused by the mapping of activities Credit Card Checking and Customer Record Checking to the exclusive sibling tasks Check Credit Rate (CCR) and Check if Return Customer (CRC) in the goal model. There are executions where both activities are executed, but this would contradict to the exclusiveness of the tasks Check Credit Rate (CCR) and Check if Return Customer (CRC) to fulfill their target goal DTC.

Table 7.1: Correspondences and Mapping Influence between Workflow Patterns and Intentional Relations

Workflow Patterns	Intentional Relations					
	AND	IOR	XOR	⟊	⟙	Dep.
Sequence	✓	✓	↯	✓	↯	✓
AND-AND parallel split - synchronization	✓	✓	↯	✓	↯	✓
AND-IOR parallel split - multi merge	✓	✓	↯	✓	↯	✓
AND-DISC parallel split - discriminator	✓	✓	↯	✓	↯	✓
AND-XOR parallel split - simple merge	✓	✓	↯	✓	↯	✓
XOR-XOR exclusive - simple merge	↯	✓	✓	↯	✓	↯
IOR-IOR multi choice - synchronizing merge	±	✓	±	±	±	±
IOR-XOR multi choice - simple merge	±	✓	±	±	±	±
IOR-DISC multi choice - discriminator	±	✓	±	±	±	±
Arbitrary cycles	–	–	–	–	–	–
Implicit termination	–	–	–	–	–	–
Multiple instances (MI) pattern	–	–	–	–	–	–
Deferred choice	↯	±	↯	±	✓	↯
Interleaved parallel routing	✓	±	↯	±	↯	↯
Milestone	–	–	–	–	–	–
Cancellation pattern	–	–	–	–	–	–

As shown in Table 7.1, we use four symbols to indicate whether we can identify an inconsistency or not: (i) '✓' indicates no inconsistency between intentional relations *IR* and workflow relations *WP*, (ii) strong inconsistencies are denoted by '↯', (iii) the symbol '±' refers to potential inconsistencies, and (iv) if our approach cannot lead to inconsistencies at the modeling level based on a mapping between tasks and activities, the corresponding cells are marked with '−'. Inconsistencies are caused by contradicting relationships of mapped elements. Thus, adding additional mappings might cause further inconsistencies.

7.4.2 Choreography Realization Inconsistencies

Choreography describes the message exchange between processes. Recurrent business process choreography scenarios are formulated using service interaction patterns [19]. These patterns encompass a variety of interaction scenarios ranging from simple message exchanges to multiple participants and multiple message exchanges.

Based on the number of involved parties and maximum message exchange between two parties, the interaction patterns are classified to:

- single-transmission bilateral interaction patterns: send, receive, send/receive
- single-transmission multilateral interaction patterns: racing incoming messages, one-to-many send, one-from-many receive, on-to-many send/receive
- multi-transmission interaction patterns: multi- responses, contingent requests, atomic multicast notification

According to the terminology and modeling assumptions of the BPMN language, choreography is between processes of different pools. Thus, the following subset of the interaction patterns is supported in BPMN [37]: Single-transmission bilateral interaction patterns cover the message exchange between two participants, including (i.a) *send*, (i.b) *receive* and (i.c) *send and receive*. The single transmission multilateral interaction pattern describe message exchange between one participant on one side and multiple participants on the other side. In the realm of the BPMN language, we consider the pattern (ii.a) *racing incoming messages*. Finally, multi-transmission interaction pattern allow allow multiple rounds of message exchange between two actors. The BPMN language supports the (iii.a) *multi-responses pattern*.

When intentional elements are mapped to activities, there might be inconsistencies between the actor dependencies in the goal model and the message exchange between pools (i.e., participants in an interaction).

In the goal model, dependency links are utilized to represent the interactions between actors. According to Mahfouz et al. [96], actors' dependencies can be classified based on the types of dependum and the logical and physical nature of the dependency. The dependum element can be either a goal, a task, or a resource [189]. A dependency can be either physical — a physical occurrence that indicates the satisfaction of a dependency (e.g., shipping items into a warehouse) or informational — needed information are provided for a depender by a dependee. We follow the argumentation of [96] that physical activities, which participants perform, are not necessarily manifested in the choreography description in a direct way since the satisfaction of the dependency can only be assessed if the depender has observed a physical occurrence of indicators. Therefore, we focus on the informational dependencies.

Interactions between actors are realized in the corresponding business processes in two ways. Firstly, if actors are mapped to pools, dependencies are realized through message exchange between pools, or more precisely between the processes in the different pools. Secondly, in the case the actors are mapped to lanes, dependencies between intentions (of different actors) are realized through control flow relations [79, 75]. Therefore, there must exist a service interaction pattern between activities that are mapped to the intentional elements of a depender and activities that are mapped to the intentional elements of a dependee. Accordingly, there might be a choreography inconsistency in the business process model with respect to actor dependencies. A choreography inconsistency is defined as the following:

Definition 7.5 (Choreography Inconsistency) *Let $GM = (\mathcal{A}, \mathcal{G}, \mathcal{P}, \mathcal{D}, \mathcal{C})$ be a goal model and $G_1 \in \mathcal{G}$, $G_2 \in \mathcal{G}$ be intentional elements, $Act_1, Act_2 \in \mathcal{A}$ actors and G_1 is in the scope of actor Act_1 ($G_1 \in Scope(Act_1)$) and goal G_2 is in the scope of actor Act_2 ($G_2 \in Scope(Act_2)$).*

Let $Ch = (\mathcal{P}, \mathcal{M})$ be a process choreography. Assume there are mappings between intentional elements G_1 and $A_1 \in activities(P_1)$ and between G_2 and $A_2 \in activities(P_2)$ ($P_1, P_2 \in \mathcal{P}$). Furthermore, there is a dependency relation $(G_1, G_x, G_2) \in \mathcal{P}$ (G_1 is the depender, G_2 is the dependee and G_x is the dependum).[3]

[3] *From a logical point of view, a dependency relation depicts a relationship between actors (depender*

There is a choreography inconsistency if there is no service interaction patterns between activities A_1 and A_2 (i.e., $(A_1, A_2) \notin \mathcal{M}$ and $(A_2, A_1) \notin \mathcal{M}$).

Figure 7.2 shows an example of a choreography inconsistency. We map Back Store, Front Store, and Customer to the corresponding swim-lanes (BackStore, FrontStore) in the process Store and to the process Customer. The actor Customer depends on the actor Back Store for achieving the goal Handle Payment. More precisely, the intentional element Collect Payment (CP) (scope of actor Back Store) depends on Select Payment Method (scope of actor Customer).[4] The task Collect Payment (CP) (scope of the actor Back Store) is mapped to the activity Ask for Payment Method in the Store process (FrontStore lane). As we can see, there is no interaction (i.e., message flow pattern) between the corresponding activities in the processes for realizing this dependency. Therefore, there exists a choreography inconsistency since we expect a message exchange between the activity Ask for Payment Method and the corresponding activity Select Payment Method (Customer process).

7.5 Knowledge Base for Realization Validation

We use Description Logics (DL) [16] to model workflow patterns, intentional relations and mappings. Based on this representation, DL reasoning services are used to validate realizations of intentional elements by workflow patterns.

7.5.1 Representation of Models and Realizations

The key part of our modeling formalism contains the relations of both models, combined with mappings between them. The goal model describes *intentional relations* between goals and the business process model specify *control flow relations* on activities, which refer to basic *workflow patterns* [138, 169]. According to the inconsistency detection problem, as specified in Section 7.4, our approach is based on a comparison of these relations. In the remainder of this section, we consider how such a comparison is realized by concept comparison (i.e., subsumption checking) in Description Logics.

---

and dependee) or between intentions of their scope, but there is no relationship imposed to the dependum.

[4] Please remember that in Figure 7.2(a) the intentional elements of the actor Customer are omitted, thus, the intentional element Select Payment Method is not depicted in Figure 7.2(a).

Representation of Intentional Relations

The *intentional relations* of a goal model GM are described in a DL knowledge base Σ_{GM}. Algorithm 1 depicts the representation of intentional relations of a goal model $GM = (\mathcal{A}, \mathcal{G}, \mathcal{P}, \mathcal{D}, \mathcal{C})$. Only tasks are implemented by activities. Thus, the algorithm introduces for each task G ($G \in \mathcal{G}_t$) a (complex) concept Rel_G to capture its intentional relations, which are either decompositions or contributions. Complex concepts describe relationships of atomic concepts and roles (properties) in DL. Intentional elements G are represented as atomic concepts.

Algorithm 1 Intentional Relations Σ_{GM}

1: Input: Goal model $GM = (\mathcal{A}, \mathcal{G}, \mathcal{P}, \mathcal{D}, \mathcal{C})$
2: **for all** $A \in \mathcal{A}$ **do**
3: **if** $(G, IOR, \{G_1, \ldots, G_n\}) \in \mathcal{D}_A$ ($\mathcal{D}_A \subseteq \mathcal{D}$) **then**
4: $Rel_{G_i} \equiv \bigsqcup_{j=1,\ldots,n} \exists requires.G_j$ $(for\ i = 1, \ldots, n)$
5: **end if**
6: **if** $(G, AND, \{G_1, \ldots, G_n\}) \in \mathcal{D}_A$ ($\mathcal{D}_A \subseteq \mathcal{D}$) **then**
7: $Rel_{G_i} \equiv \bigsqcap_{j=1,\ldots,n} \exists requires.G_j$ $(for\ i = 1, \ldots, n)$
8: **end if**
9: **if** $(G, XOR, \{G_1, \ldots, G_n\}) \in \mathcal{D}_A$ ($\mathcal{D}_A \subseteq \mathcal{D}$) **then**
10: $Rel_{G_i} \equiv \bigotimes_{j=1,\ldots,n} \exists requires.G_j$
11: **end if**
12: **end for**
13: **for all** $(G', \dagger, G) \in \mathcal{C}$ **do**
14: $Rel_G := Rel_G \sqcap \exists requires.G'$
15: **end for**
16: **for all** $(G', \dot{-}, G) \in \mathcal{C}$ **do**
17: $Rel_G := Rel_G \sqcap \neg\exists requires.G'$
18: **end for**
19: **for all** $(G_{depender}, G_{dependum}, G_{dependee}) \in \mathcal{P}$ **do**
20: $Rel_{G_{depender}} := Rel_{G_{depender}} \sqcap \exists requires.G_{dependum}$
21: $Dep_{G_{depender}} := Dep_{G_{depender}} \sqcap \exists requires.G_{dependum}$
22: **end for**

Lines 4–6 treat IOR-decompositions of an intention into subgoals G_i $i \in (1, \ldots, n)$. Thus, intentions G_i are disjunctively related to each other. This is represented in DL by a concept union over all intentions G_j that are members of the IOR-decomposition. For each intention G_i that is part of the IOR-decomposition, we introduce a concept Rel_{G_i} to describe the relations of each intention. In this vein, a conjunctive decomposition (lines 10–12) is described by a concept intersection in DL. As in the previous case,

we introduce relation concepts Rel_{G_i} to capture conjunction for all members of the decomposition.

The representation of exclusive decompositions is straightforward (lines 7–9)[5]. An intention can only be the source of one decomposition, i.e., either IOR, XOR or AND.

Sufficient positive contributions (lines 14–16) specify that the fulfillment of G requires the fulfillment of G'. Thus, we add the expression $\exists requires.G'$ to the definition of the relation concept Rel_G. The meaning in Description Logics is that an instance of goal G' must have a role / property *requires* to another goal instance. Sufficient negative contributions (lines 17–19) use concept negation in order to represent that the fulfillment of G can not be achieved if G' is fulfilled. A task might be involved in multiple (positive and negative) contributions simultaneously. Contributions that are not sufficient are not included in this representation since their effect cannot be completely determined at design time when mappings are assigned to the models.

Lines 19–22 capture dependencies between intentions, where intention $G_{depender}$ depends on the intention $G_{dependee}$ on an intention $G_{dependum}$. Logically, this is an implication that the fulfillment of $G_{depender}$ requires the fulfillment of the intention $G_{dependum}$, which is reflected in the DL concept expression $Rel_{G_{depender}}$. Following the discussion in Section 7.4, this relation is only added into the knowledge base if there is an explicit dependency relation between the intentional elements $G_{depender}$ and $G_{dependee}$ and both are in the scope of different internal actors. In line 21 the dependency between intentions $G_{depender}$ and $G_{dependee}$ is represented by an additional concept expression $Dep_{G_{depender}}$ to handle the other case when at least one actor is an external actors (i.e., does not belong to the system-to-be). The dependency relation might cause both orchestration and choreography inconsistencies, depending on whether at least one actor is an external actor or not. This depends on the mapping, i.e., whether an actor is mapped to a pool (process) or to a lane (part of a process). Thus, we will distinguish these two cases in the inconsistency detection later on. All other intentional relations can only cause inconsistencies in the process orchestration.

[5]Please note that $\otimes$ is not a standard operator in DL. For a more concise representation, we use $\bigotimes_{G' \in \{G_1,\ldots,G_n\}} \exists requires.G'$ as an abbreviation for $\bigsqcup_{G' \in \{G_1,\ldots,G_n\}} \exists requires.G' \sqcap \neg(\bigsqcup_{G'',G''' \in \{G_1,\ldots,G_n\}} (\exists requires.G'' \sqcap \exists requires.G'''))$.

$$Rel_{ApproveOrder} \equiv \exists\, requires.ApproveOrder \sqcap \exists requires.CustomerProfileRetrieve \quad (7.1)$$

$$Rel_{ElectronicPayment} \equiv \exists requires.ElectronicPayment \sqcup \exists requires.InPersonPayment \quad (7.2)$$

Axiom 7.1 describes an AND-relation of the sibling intentional elements Approve Order and Customer Profile Retrieve. The intentional relation of *Customer Profile Retrieve* is defined equally. The AND-relation is expressed by the concept intersection ($\sqcap$) of the concept expressions
$\exists\ requires.\ ApproveOrder$ and
$\exists\ requires.\ CustomerProfileRetrieve$.

An inclusive OR-relation between Electronic Payment or In Person Payment is exemplified in Axiom 7.2, in which the fulfillment relationship of both tasks is reflected, while both tasks are inclusive siblings within an OR decomposition. The relation of intention In Person Payment is defined equally.

Workflow Relations

We represent business process models in terms of workflow (control flow) relations between activities. Algorithm 2 describes how the corresponding knowledge base Σ_{PM} is built.

There might be an overlapping of activity relations. For instance, the activity Money Order in Figure 7.2(b) is part of an exclusive branching fragment (internal fragment in Figure 7.2(b)) and also within a parallel branching fragment, i.e., the activity Money Order is conjunctively and exclusively related to other activities. Accordingly, we build orchestration relations of activities ($Orch_A$) as a conjunction (intersection in DL) of activities from the different control flow patterns. Initially, each orchestration relation concept $Orch_A$ is defined as equivalent to the universal concept $\top$ (line 3).

In lines 2–6, the sequential control flow relations (in both directions) of an activity A are covered by restricting the concept $Orch_A$. Since gateways do not realize intentions, they are transparent in the representation (cf. Definition **??**). Afterwards, relations within fragments $\mathcal{F}$ are considered, whereby only those fragments that start and end with a gateway are relevant. Each branch $B \in \mathcal{B}$ of a fragment $F \in \mathcal{F}$ is a set of

Algorithm 2 Workflow Patterns Σ_{PM}

Input: Mat. process model $PM = (\mathcal{V}, \mathcal{E}, \mathcal{F}, \mathcal{E}_A)$
for all $A \in \mathcal{A}$ $(\mathcal{A} \subseteq \mathcal{V})$ **do**
  $Orch_A \equiv \top$
end for
for all $E \in \mathcal{E}_A$ **do**
  if $(A_1, A_2) = E$ **then**
    $Orch_{A_1} := Orch_{A_1} \sqcap \exists requires.A_2$
    $Orch_{A_2} := Orch_{A_2} \sqcap \exists requires.A_1$
  end if
end for
for all $F \in \mathcal{F}$ **do**
  if $F = (AND, AND, \mathcal{B}) \vee F = (AND, IOR, \mathcal{B}) \vee F = (AND, XOR, \mathcal{B}) \vee F = (AND, DISC, \mathcal{B})$ **then**
    $Orch_{A_i} := Orch_{A_i} \sqcap \bigsqcap_{B_j \in \mathcal{B}} \exists\ requires.(\bigsqcap_{A_k \in B_J} A_k)$
  end if
  if $F = (IOR, IOR, \mathcal{B}) \vee F = (IOR, DISC, \mathcal{B}) \vee F = (IOR, XOR, \mathcal{B})$ **then**
    $Orch_{A_i} := Orch_{A_i} \sqcap \bigsqcup_{B_j \in \mathcal{B}} \exists\ requires.(\bigsqcap_{A_k \in B_J} A_k)$
  end if
  if $F = (XOR, XOR, \mathcal{B})$ **then**
    $Orch_{A_i} := Orch_{A_i} \bigotimes_{B_j \in \mathcal{B}} \exists\ requires.(\bigsqcap_{A_k \in B_J} A_k)$
  end if
end for

atomic activities. In lines 8–10, the algorithm restricts the concept definitions $Orch_{A_i}$, in which A_i are those activities that appear in parallel branches. We use an intersection between activity sets of sibling branches, indicating the conjunctive relationship between sibling activities in parallel branches. From a logical point of view, we treat different closing gateways (multiple merge, synchronization and discriminator) equally. Branching relations do not impose restrictions on activities within the same branch in a fragment (in contrast to the sequence pattern). Thus, we describe all activities of the same branch by a concept intersection, independent of the kind of branching.

Multi choices are treated in lines 15–17, including synchronizing merge, simple merge and discriminator. Logically, activities of sibling branches are connected by a concept union. In case of exclusive branching (lines 23–25), the concept definitions $Orch_{A_i}$ contain a further restriction that allows only the execution of one branch.

Axiom 7.3 depicts a part of the control flow relation of activity Credit Card Checking (cf. Figure 7.2(b)). The activity is part of a choice fragment, i.e., either activity Credit Card Checking or Customer Records Checking can be executed, or even both. This relation is represented by a concept union in the first line of the axiom. The second line of the axiom covers sequential relations of the activity Credit Card Checking to its successor Approve Order.

Axiom 7.4 describes the relation of activity Fraud Detection, as member of a parallel branch, in which activity Apply Discount and the internal fragment with activities Credit Card, Debit Card and Money Order. The relation concept also covers predecessor (Verify Payment Information) and successor (Build And Package Order) activities.

$$
\begin{aligned}
Orch_{CreditCardChecking} \equiv\ & (\exists\ requires.CreditCardChecking \\
& \sqcup \exists requires.CustomerRecordsChecking) \\
& \sqcap\ \exists requires.ApproveOrder \qquad (7.3) \\
Orch_{FraudDetection} \equiv\ & (\exists\ requires.FraudDetection \\
& \quad \sqcap \exists\ requires.ApplyDiscount \\
& \quad \sqcap (\exists requires.CreditCard \\
& \quad\quad \otimes \exists requires.DebitCard \\
& \quad\quad \otimes \exists requires.MoneyOrder)) \\
& \sqcap \exists\ requires.VerifyPaymentInformation \\
& \sqcap \exists\ requires.BuildAndPackageOrder \qquad (7.4)
\end{aligned}
$$

Process Choreography

The workflow relations of a single business process model PM are already represented in the knowledge base Σ_{PM}. Finally, a representation of the process choreography between a set of processes is needed.

Algorithm 3 depicts the axioms of a process choreography Ch in the knowledge base Σ_{Ch}.

Algorithm 3 Choreography Knowledge Base Σ_{Ch}

```
1: Input: Process Choreography Ch = (P, M)
2: for all Proc ∈ P do
3:    for all A ∈ activities(Proc) do
4:       Chor_A ≡ ⊤
5:    end for
6: end for
7: for all (A_1, A_2) ∈ M do
8:    Chor_{A_1} := Chor_{A_1} ⊓ ∃requires.A_2
9:    Chor_{A_2} := Chor_{A_2} ⊓ ∃requires.A_1
10: end for
```

For each activity, a concept $Chor_A$ is initialized (line 4). If there is a message exchange between activities A_1 and A_2, expressed by $(A_1, A_2) \in \mathcal{M}$, we extend the concept definitions $Chor_{A_1}$ and $Chor_{A_2}$ by referring to the other activity. According to Section 2.3, a message exchange is only allowed between activities of different processes. In the concept definitions of the choreography relations, the same DL role *requires* is used as for the workflow relations that describe the process orchestration.

Mapping between Intentions and Activities

Besides intentional relations and workflow patterns, we have to represent the realization of tasks by the corresponding activities in terms of mappings in the knowledge base Σ_M. A mapping is described as a concept equivalence in the knowledge base. If there is a mapping $m(G, A)$ from a task G to an activity A, we represent the mapping by an axiom $A \equiv G$. In both models, we use the same role *requires* in order to allow for a comparison of relations of both models.

7.6 Detection of Realization Inconsistencies

Mappings describe the implementation of a task (intentional element) by an activity. As a consequence, it is expected that the relations of a task and its corresponding activity are not contradicting. The detection of orchestration and choreography realization inconsistencies can be checked independently from each other, by comparing the intentional relations of the goal model with the process orchestration of a particular business process model and with the choreography of a set of interacting business processes.

7.6.1 Process Orchestration Inconsistencies

For a given mapping $m(G, A)$, we compare the corresponding workflow patterns WF of activity A and the intentional relations IR of intentional element G in order to test whether these relations might cause any inconsistencies. We distinguish between the two introduced kinds of inconsistencies and no inconsistency, which is also called realization equivalence, according to the overview of inconsistencies of Table 7.1 (Section 7.4). Relations of G and A are represented by concepts Rel_G and $Orch_A$ in the knowledge base.

In the knowledge base, this is reflected by the combination of the concepts Rel_G and $Orch_A$ as follows: (i) A strong inconsistency means that there can not be any execution combination of activities that fulfills the intentional relations of the corresponding intentional elements. In the DL sense, the intersection of both concepts $Rel_G \sqcap Orch_A$ is unsatisfiable, i.e., the intersection $Rel_G \sqcap Orch_A$ can not have a common individual. (ii) A potential inconsistency indicates that there might be an execution of activities, in which the corresponding intentional relation is not fulfilled. In this case, the intersection $Rel_G \sqcap Orch_A$ is satisfiable, i.e., there can exist common individuals of both concepts. (iii) The intentional relations of G and the workflow patterns of activity A are called realization equivalent if all execution combinations that involve activity A lead to a fulfillment of the intentional relations of G. This is true if the subsumption $Orch_A \sqsubseteq Rel_G$ holds. Logically, this means that $Orch_A$ implies Rel_G.

In order to check these three different cases, we introduce the following validation concepts. The concept $Valid^{\checkmark}$ is defined as $\neg Orch_A \sqcup Rel_G$ to encode the subsumption test $Orch_A \sqsubseteq Rel_G$. Thus, $Orch_A$ is subsumed by Rel_G if $Valid^{\checkmark} \equiv \neg Orch_A \sqcup Rel_G$ is equivalent to the universal concept $\top$. This indicates the realization equivalence. A concept $Valid^{\pm}$ is defined as the intersection $Orch_A \sqcap Rel_G$ to test whether there is

a potential or a strong inconsistency, i.e., if $Valid^{\pm}$ is satisfiable there is a potential inconsistency, otherwise a strong inconsistency. Formally, the knowledge base Σ_O is obtained as described in Definition 7.6.

Definition 7.6 (Orchestration Knowledge Base Σ_O) *The knowledge base $\Sigma_O := \Sigma_{GM} \cup \Sigma_{PM} \cup \Sigma_M$ is extended as follows: For each mapping $m(G, A)$ from an intentional element G to an activity A, which is represented in Σ_M by an axiom $G \equiv A$, we insert the following axioms into Σ_O:*

- $Valid^{\checkmark}_{A,G} \equiv \neg Orch_A \sqcup Rel_G$
- $Valid^{\pm}_{A,G} \equiv Orch_A \sqcap Rel_G$

Given the orchestration knowledge base, we get the validation result *en passant*. Classifying the validation concepts $Valid^{\checkmark}_{A,G}$ and $Valid^{\pm}_{A,G}$ of the knowledge base Σ_O leads to the following results:

1. If $Valid^{\checkmark}_{A,G}$ is classified equal to the universal concept $\top$, we can guarantee the realization equivalence of the workflow patterns over activity A and the intentional relations over intentional element (task) G.

2. In the other case, there is either a strong inconsistency or a potential inconsistency. This is indicated by the classification of the concept $Valid^{\pm}_{A,G}$. If $Valid^{\pm}_{A,G}$ is classified as a subconcept of the bottom concept $\bot$ (i.e., $Valid^{\pm}_{A,G} \sqsubseteq \bot$), there is a strong inconsistency, otherwise (i.e., $Valid^{\pm}_{A,G} \not\sqsubseteq \bot$) there is a potential inconsistency.

7.6.2 Process Choreography Inconsistencies

In case a goal model GM is mapped to multiple processes and there are dependencies between actors in the goal model, we have to check whether there is a choreography inconsistency. Like for the orchestration inconsistencies, the mappings between intentions and activities are represented in the mapping knowledge base Σ_M. Intentional relations are covered in the goal model knowledge base Σ_{GM}. Additionally, choreography between processes of mapped activities need to be considered. We obtain our knowledge base Σ_C as described in Definition 7.7.

Definition 7.7 (Choreography Knowledge Base Σ_C) *The knowledge base $\Sigma_C := \Sigma_{GM} \cup \Sigma_{Ch} \cup \Sigma_M$ is extended as follows: For each mapping $m(G, A)$ from an intentional*

element G to an activity A, which is represented in Σ_M by an axiom $G \equiv A$, we insert the following axiom into Σ_C:

- $Valid_{G,A}^{Chor} \equiv \neg Rel_G \sqcup Chor_A$

From a logical point of view, this is also an implication as for the potential inconsistency detection. The intentional relations Rel_G imply the choreography relation $Chor_A$. We expect that a dependency between intentions (in the scope of different external actors) is also covered in the choreography $Chor_A$ of the corresponding mapped activities. The implication is not satisfied, which is tested in the validation later on, if there is a dependency relation but not the corresponding message exchange (choreography). Otherwise, the implication holds. In the DL representation, this means that the concept $Valid_{G,A}^{Chor}$ is equal to the universal concept $\top$. Therefore, we detect a choreography inconsistency if a concept $Valid_{G,A}^{Chor}$ is not equal to the universal concept $\top$ after a concept classification by a reasoner.

7.7 Proof-of Concept and Discussion

The first part of this section conducts a performance evaluation of the inconsistency detection algorithms, demonstrating the tractability of the approach for models of realistic size. Discussions on qualitative analysis and on the general methodology and applicability are presented afterwards.

7.7.1 Performance Evaluation

We conduct an evaluation by providing a proof-of concept in which workflow patterns from BPMN processes, message exchange between processes and intentional relations from GRL goal models are represented in an OWL DL (OWL2 DL)[6] knowledge base. We analyze the reasoning performance with different sizes of goal and process models and with varying numbers of inconsistencies in order to test how the validation approach is applicable for real sized process models.

[6]The Web Ontology Language (OWL): http://www.w3.org/TR/owl2-overview/

Experimental Setting and Data Set

The goal of our evaluation setting is to systematically observe how the presented validation algorithms scale with the increase of the models and the increase of orchestration and choreography inconsistencies. To achieve this, we apply the simulation modeling technique by following guidelines similar to those proposed in [76]. We selected the simulation technique, as it is commonly used in the context of model validation and verification [61, 140].

We build the models directly in the Description Logics representation as follows:

1. We randomly generated goal models with three different sizes: (I) 200, (II) 400 and (III) 600 intentional elements. More than 50% are tasks, i.e., intentional elements that can be mapped to activities. The ratio / distribution of AND, IOR and XOR decompositions is nearly equally. Each goal model consists of 4 actors (2 internal and 2 external actors).

2. For each goal model, 2 corresponding process models are derived, each distributed over two lanes. For each task, a corresponding activity is obtained, ending up with 2 interacting process models with a total number of (I) 100, (II) 200 and (III) 300 activities (of both models).

   - AND, IOR and XOR decompositions are "transformed" to AND, IOR and XOR workflow (control flow) patterns.
   - Positive contributions and dependencies between internal actors lead to sequential patterns.

   Due to this generation principle, the "correct" mappings between tasks and activities are already given by the experimental process model design, instead of manually establishing them. By this design, we already know which activity implements which task.

3. Finally, we modify mappings between tasks and activities such that different orchestration and choreography inconsistencies occur. The ratio of inconsistencies varies (for both kinds of inconsistencies) between 10%, 30% and 50% (out of all possible inconsistencies). This principle is based on the mutation testing technique, originally proposed in [41] as a common fault-based technique where simple faults are predicted.

Table 7.2: Characteristics of the Generated Models

No	Goal Model [Intentions]	Process Model [Activities]	Realizations [Mappings]	Choreography Inc. [%]	Orchestration Inc. [%]
(I)	200	100	100	[10, 30, 50]	[10, 30, 50]
(II)	400	200	200	[10, 30, 50]	[10, 30, 50]
(III)	600	300	200	[10, 30, 50]	[10, 30, 50]

Due to the artificial changing of mappings (third step), we inject orchestration and choreography inconsistencies into (originally correct) mappings between goal and process models. Thus, we exactly know the number of possible and existing inconsistencies in each concrete case, which paves the way for a meaningful analysis of the performance of our validation algorithm by varying the number of orchestration and choreography inconsistencies.

As already mentioned, the generated process models consist of 100, 200 and 300 activities on average. This size of process models has been observed substantially through existing (reference) process models in the literature [101] such as the SAP reference process model.

The distributions of inconsistencies were chosen not only to preserve diversity in the number of orchestration and choreography inconsistencies, but also to have realistic view of the number of possible inconsistencies that might happen in real case-studies. We assume that it is rare to have a model with 80% inconsistencies.

Table 7.2 shows the different settings we apply. There are three different sizes (I)–(III) and for each size there are 9 different distributions of inconsistencies (10, 30, 50 % of orchestration and choreography inconsistencies), ending up with 27 different settings. For each setting, 50 samples were generated.

Evaluation Results

After reasoning on the knowledge bases Σ_O and Σ_C, our tool produces a list of realization equivalent mappings ($Valid^{\checkmark} \equiv \top$) and a list of strong inconsistencies $Valid^{\pm} \sqsubseteq \bot$, the remaining are known as potential inconsistencies (i.e., $Valid^{\pm} \not\sqsubseteq \bot$). Likewise, a list of choreography inconsistencies ($Valid^{Chor} \not\sqsubseteq \bot$) is obtained. The ontology creation is

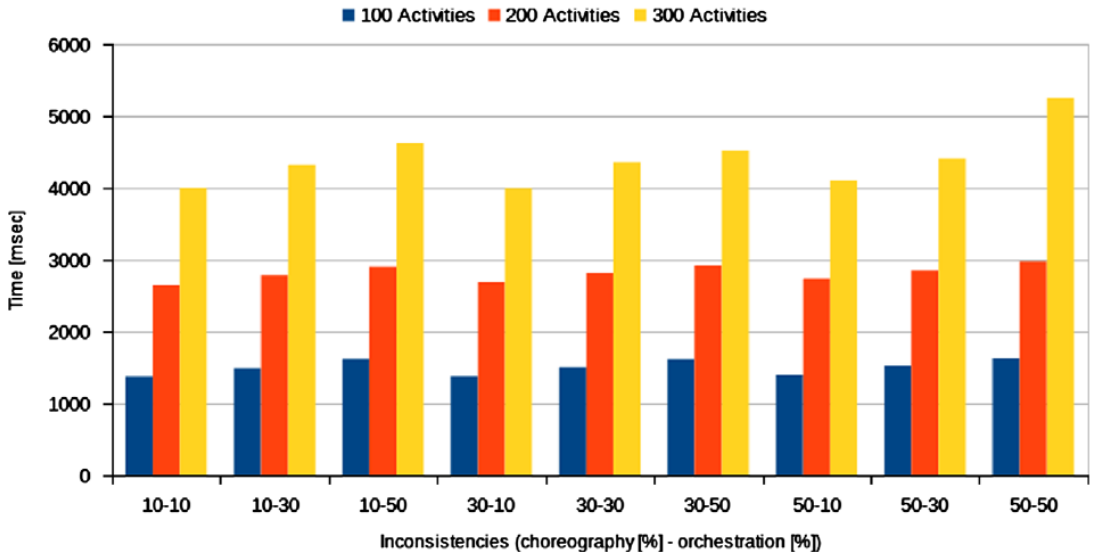


Figure 7.3: The results of the experiment

implemented with the OWL-API[7]. For reasoning, we used the Pellet reasoner[8]. Our test system is a Notebook with an Intel Core 2 Duo 8700 CPU (2.5 GHz, 4 MB cache and 2GB DDR2 RAM). The DL expressiveness of the knowledge bases Σ_O and Σ_C is $\mathcal{ALC}$.

The overall result for the different settings according to Table 7.2 is depicted in Figure 7.3. For each of these 27 different settings, three different model sizes, three different numbers of orchestration inconsistencies and also three different numbers of choreography inconsistencies are applied. The time for inconsistency detection, i.e., the time the reasoner needs to classify the validation concepts, is the average time for each setting (each setting consists of 50 generated models).

The observations in this experiment show that the execution time is reasonable for realistic-size models and for a varying number of inconsistencies. As shown in Figure 7.3, the *size of the models* affect the execution time. The mean values of the execution time is: 1493.04 msec (100 activities), 2807.74 msec (200 activities) and 4394.27 (300 activities). This is a rather expected result since in the increase in the model size also increase proportionally the number of mappings and therefore also the number of validation

[7]OWL-API site: http://owlapi.sourceforge.net/

[8]Pellet reasoner site: http://clarkparsia.com/pellet/

concepts (three for each mapping) that have to be classified. We can observe that the execution time of our algorithm for the largest experimented model with 300 activities in less than 5.3 seconds.

Regarding the number of inconsistencies, the experiment demonstrate that the increase of *orchestration inconsistencies* increases the execution time of the algorithm. Independent of the model size, an increase of the percentage of orchestration inconsistencies increases the validation time (with a fixed model size and a fixed number of choreography inconsistencies) on average by $7\% - 9\%$. For instance, from setting $10 - 30$ to setting $10 - 50$ the mean time (M) increases from 1486 to 1613 msec. There is no difference between strong and potential inconsistencies.

The increase of *choreography inconsistencies* has less influence on the execution time, only a slight increase is shown. A possible explanation is the lower number of relationships that are covered by the concept expressions ($Chor_A$ and $Valid^{Chor}$) since in contrast to orchestrations (strong and potential inconsistencies) only dependencies are covered by these concept expressions and no other intentional relations. The increase is on average $3\% - 5\%$.

Threats to Validity

An experimental evaluation is always subject to different threats that can affect the validity of the results. In the following, we investigate possible threats to the validity of our results. Three main factors influence the validity of our experiments: (i) the size of the models, (ii) the distribution of intentional (and therefore also control flow) relations and (iii) the modeling approach. With respect to the size of the process models, our investigation over existing publications in process management confirmed that the sizes of generated models are aligned with the size of existing models in the literature. The same holds for the assumed distribution of intentional relations (AND, IOR and XOR), which is assumed as approximately equal. Regarding the proposed modeling and formalization approach of process models and goal models, we may have slightly different execution times for different formal representations. Thus, our results can not be generalized to other formalisms, which could be used as alternative for implementation of the proposed validation approach. However, the experimental results show that employing Description Logic is a tractable means to model and validate goal-oriented process models.

7.7.2 Validation Exemplified

We demonstrate the validation for an excerpt of Figure 7.2. Consider the strong inconsistency for the activities Send Receipt and Shipment.[9] They are siblings in exclusive branches of the same fragment. In Σ_{PM}, there are the definitions of this relation for both activities. An excerpt of the relation definition of activity Send Receipt is shown in Axiom 7.5.

$$\begin{aligned} Orch_{SendReceipt} \equiv\ & (\exists\ requires.SendReceipt \\ & \sqcup \exists\ requires.Shipment) \\ & \sqcap \neg(\exists\ requires.SendReceipt \\ & \sqcap \exists\ requires.Shipping) \end{aligned} \tag{7.5}$$

$$\begin{aligned} Rel_{BL} \equiv\ & \exists\ requires.ShipOrder \\ & \sqcap \exists\ requires.BL \end{aligned} \tag{7.6}$$

Since Send Receipt is mapped to the intentional element (task) Bill Preparation (BL), Send Receipt is defined as equivalent to the concept Bill Preparation (BL) and Shipment is equivalent to Ship Order (mapping knowledge base Σ_M). From the goal model, there is a relation definition of $Bill$ in Σ_{GM} as depicted in Axiom 7.6.

The validation concept $Valid^{\checkmark} \equiv \neg Orch_{SendReceipt} \sqcup Rel_{BL}$ is classified by the reasoner as different from the universal concept $\top$, i.e., we know that the realization equivalence does not hold between Send Receipt and Bill Preparation. The other validation concept $Valid^{\pm} \equiv$
$Orch_{SendReceipt} \sqcap Rel_{BL}$ is classified equivalent to the bottom concept $\bot$, i.e., indicating a strong inconsistency. The same holds for Shipment.

7.7.3 Lessons Learned

The course of our research, as presented in the previous part of this paper, has raised the following issues for further discussions.

[9] Please note that we represent here the exclusive relation in the standard Description Logics notation to ease the understanding of the inconsistency that is caused by the existence of a positive and its negated statement (after mapping Send Receipt to Bill Preparation (BL)).

Inconsistency detection and validation The purpose of the validation is to detect inconsistencies between mapped elements with respect to their relationships, i.e., intentional relations and relations of activities. This is based on a comparison how these elements are logically related to each other, e.g., whether their relations coincide or contradict to each other. However, an analysis whether or to which degree the execution of an activity satisfies a particular goal from a qualitative perspective leads to further interesting research questions, e.g., like the consideration of soft goal satisfaction in reasoning algorithms.

Resolution of detected inconsistencies The motivation of the presented work is to ensure correct mappings between two different kinds of models. We do not restrict how a detected inconsistency can be resolved. In general, there are three non-disjoint possible directions: (i) changing the mappings, (ii) redesigning the process model or (iii) redesigning the goal model.

However, another application context could be the dedicated redesign of process models, in which inconsistent process models (with respect to goal models) are redesigned until they meet the requirements that are imposed by the goal models.

From requirements to orchestration As mentioned in Section 7.4, we restrict mappings to tasks in the goal model since they operationalize stakeholders' goals, and therefore, they can be directly realized by activities in a process. The proposed modeling framework is based on these assumptions. Discussions whether hard and especially soft goals could be mapped to activities and whether this is meaningful are outside of the scope of this paper.

From requirements to choreography As discussed in Section 7.4, the inconsistency detection does not check the directions of dependencies and interaction patterns, since the dependency relations in the goal model do not provide enough information to decide the type of service interaction patterns between the corresponding processes, we can not make a conclusion about the types and directions of interaction patterns based on dependency relations. Hence, we did not make a claim in this regard. However, it is an interesting point for further research investigations and case studies to investigate whether and how directions of dependencies in goal models influence the directions of message exchange between process models.

Overall application context Another aspect, to be discussed, is the usefulness and applicability in real applications. The key benefit of the modeling setting with goal and process models in combination with mappings offers two different "views" on a system / application (requirements and process models), while ensuring the correct alignment between modeling constructs and their logics in both view. This allows a (partly) independent development and maintenance of a system from two viewpoints. Alternative approaches consider the derivation of a process model from an existing requirement model (cf. Section 7.8).

7.7.4 Rationale for Description Logics

Description Logics is an expressive language for knowledge representation. The semantics of Description Logics provides an unambiguous interpretation of expressions in the knowledge base that is a prerequisite for automated reasoning. The contribution of Description Logics reasoning to the validation is threefold:

- We use reasoning to recognize orchestration and choreography inconsistencies.
- Due to the classification of concepts and the subsumption checking between concepts, the reasoner directly pinpoints the source of an inconsistency, i.e., which element (intention and activity) is part of an inconsistency. This is crucial in large models, where various mappings exist.
- We use classifications of concept expressions by the reasoner in order to check whether a concept expression is always satisfied, is satisfiable or is unsatisfiable. This is exploited to distinguish between strong and potential inconsistency.

The *strengths* of Description Logics are quite efficient reasoning procedure in practical settings and there is a well established infrastructure for modeling and reasoning tool support. Description Logics is a decidable subset of first-order logic, and the theoretical exponential reasoning complexity is tractable in practical settings. In contrast to propositional logic, Description Logics is more expressive and allows the integration of further background knowledge like annotations of elements.

Finally, there are some *weaknesses* of Description Logics and the proposed modeling approach. Certain aspects of the models cannot be represented in standard Description Logic models, e.g., the influence of qualitative aspects, as captured by soft goals in

which there is no clear-cut criteria to measure the achievement, need further modeling approaches.

Our representation relies on a structural representation of processes and how activities are related to each other. Temporal relations are covered by predecessor and successor relations of activities, but without (rather powerful) temporal (logic) representation formalisms. Due to transformations and dedicated modeling decisions, like the introduced materialized process model representation, we can disregard temporal constructs. However, such temporal modeling constructs are widely used in process modeling and the incorporation of them is another interesting aspect. The standard Description Logics language does not support temporal constructs. However, there are some initial approaches that address such issues in Description Logics [95].

7.8 Related Work

The first group of related work considers relations between goal models and business process models, but not validation as it is done in our work. Transforming goal models into business process model has been one of the major research areas in business process management. Lapouchnian et al. [86] use goal models for the configuration of business processes. Goal models are annotated with control flow information and afterwards transformed into BPEL processes. Similarly, Decreus and Poels [38] annotate goal-oriented models in the so-called B-SCP framework with control flow information and transform them into BPMN skeletons. Furthermore, Frankova et al. [50] transform SI*/Secure Tropos models into skeletons of process models in BPMN, from which they generate executable processes in BPEL. On the other hand, Santos et al. [141] derive goal models from existing process models. Such goal models are then used to control variability and configuration of processes. Finally, Koliadis et al. [79] annotate activities of process models with effects, whereby these effects serve for a comparison of a process with user goals, i.e., effects of activities are compared with goal fulfillment. The basic principle is to reflect changes from an $i^\star$ model to a BPMN model and vice versa.

The second group of related research is more related to our work, where some kind of verification between requirements and business process models is investigated. In this line of related research, Kazhamiakin et al. [75] propose a methodology that provides a set of high-level mapping rules to produce process models in BPEL from Tropos models. In order to enable requirements driven verification of process models, they employ the

formal Tropos language in combination with temporal constraints. While their work and their tools support several types of formal analyses for consistency checks in the requirements models as well as in the process model, they do not focus on validation of the inconsistencies across models, i.e., those cases where both models are consistent but the mapping align elements with contradicting relationships, as it is done in our work.

Soffer and Wand [153, 154] propose a generic theory-based process modeling framework based on Bunge's ontology for a goal-driven analysis of process models to check goal reachability in process models. They define a goal as stable state, which indicates the termination of the process. Their framework defines a set of assumptions and parameters (based on goal relations and workflow relations), which are used to check if a set of workflow patterns ensure that processes can always reach their goals. Hence, using these parameters, it is possible to identify sets of valid and invalid design decision with respect to business goals. They also suggest appropriate redesign actions to eliminate invalid designs. Our approach follows the similar objective, but we use DL based reasoning to identify inconsistencies automatically.

Pistore et al. [124] introduce an approach to design and verify web services. The requirements are also specified using goal models or more precisely Formal Tropos models. BPEL4WS is used for business processes. Their approach enables verification of whether linear time temporal constraints specified in goal models hold in business process models. Mahfouz et al. [97] complement the approach of Pistore et al. and verify if linear time first order temporal constraints, specified in goal models, hold in message exchange (choreography). Our approach verifies that intentional relationships in goal models are consistent with execution and message exchange in process models. Similarly to our approach, Markovic et al. [99] introduce an ontology based specification of goal models and their relationships to process models. However, potential inconsistencies and validation of goals in process models are not discussed.

Liaskos et al. [92] also introduce an approach for goal based customization of workflows. A family of processes is extended with the notation for partial temporal ordering of goals. A particular member of the family is specified by constraints with the means of linear temporal logic operators. However, this approach does not take into consideration business process logic embedded in the realization of business processes, which is the focus of our validation. La Rosa et al. [84] propose a questionnaire based customization of configurable business processes represented in C-YAWL [55]. They use a Petri-net

based reasoner [166] to preserve the correctness during process configuration. Variability patterns of La Rosa et al.'s work are narrower than patterns introduced in this work.

7.9 Conclusions

In this paper, we have presented a novel approach for handling inconsistencies in mappings between goal models and business process models. We distinguish between two kinds of inconsistencies. (i) Inconsistencies of the process orchestration are caused by contradicting control flow relationships between activities in the business process model and between relationships of user intentions in the goal model. (ii) Choreography inconsistencies occur if dependencies among actors in the goal model are not reflected by message exchange between the corresponding activities in the business process models. By automatically identifying these inconsistencies, we are able to detect executable processes that did not meet user requirements or lead to undesired executions. Additionally, we allow for goal models and business process models to evolve independently.

Our contribution extends the body of knowledge in the field by considering these mapping as first-class citizens along with goal models and business process models. We plan to extend this approach in combination with our existing work on configuration of business process families [59] to provide a complete solution for software product lines that use goal models, feature models and process models as main artifacts.

Chapter 8

Goal Model and Feature Model Validation

This chapter consists of "Goal-oriented modeling and verification of feature-oriented product lines" paper which is published in *The Journal of Software Systems and Modeling (SOSYM)*[1]. The paper introduces the notion of product line variability in goal models and proposes notation to represent the variability in the goal models. Then, a set of possible inconsistency patterns are identified. Finally, a description logic based approach is developed to ensure alignment of the feature model variability with family goal model variability.

Author's role - I was the main contributor in developing validation technique between goal models and feature models. I identified the inconsistency patterns. The case study, design of experiments, implementing the experiments, analysis of the results, and developing parts of description logic algorithms has been done by the author. I wrote all the sections of the paper except sections 5 and 6.

[1] Paper originally published in Software Systems and Modeling, Issue 1, 2014. DOI 10.1007/s10270-014-0402-8.

Asadi, M., Groner, G., Mobabbati, B., Gasevic, D. (2014) Validation of User Intentions in Feature-oriented Software Families. Journal of Software and Systems Modeling (SOSYM), 20 pages, in press.

8.1 Abstract

Goal models and business process models are complementary artifacts for capturing the requirements and their execution flow in software engineering. In this case, goal models serve as input for designing business process models. This requires mappings between both types of models in order to describe which user goals are implemented by which activities in a business process. Due to the large number of possible relationships among goals in the goal model and possible control flows of activities, developers struggle with the challenge of maintaining consistent configurations of both models and their mappings. Managing these mappings manually is error-prone. In our work, we propose an automated solution that relies on Description Logics and automated reasoners for validating mappings that describe the realization of goals by activities in business process models. The results are the identification of two inconsistency patterns – orchestration inconsistency and choreography inconsistency – and the development of the corresponding algorithms for detecting these inconsistencies.

8.2 Introduction

One of the most prominent paradigm for reuse in software engineering is Software Product Lines Engineering (SPLE). An SPL (also known as a product family) is a set of software systems that share most of their *features*. SPLE consists of the *domain engineering* and *application engineering* life-cycles [125]. In the *domain engineering* life-cycle, an SPL is developed as a whole. Variability and commonality among SPL members are represented by feature models. In the *application engineering* life-cycle, a final application can be derived through the feature model configuration, i.e., the process of selecting and removing features from the feature model based on stakeholders' requirements [73].

Features represent both technical elements (internal features) and non-technical elements (external features) of a software system [22]. That is, not only does a feature model encompass the visible characteristics of product lines, but it also includes many design and implementation features along with the variability relations between those

features. Moreover, there are complex mapping relations between features and stakeholders' objectives, such that a goal can be realized by several features and a feature can be used for the realization of several goals. Considering the technical aspects of feature models, the large size of feature models, and the complex mapping relations between features and stakeholders intentions, it is not easy for a stakeholder to understand the functional and non-functional aspects of features and to select features based on their objectives. Additionally, the features in the feature model are of different interest for stakeholders involved in the project [32, 2]. For instance, final stakeholders are interested to the user visible features while designer and programmer require to see detail and technical features.

These challenges urge techniques which can handle the complexity of feature models and facilitate the selection of features during the configuration process. Clarke and Proenca [32] and Acher et al. [2] emphasized the importance of modularity in managing complexity by using views to show a feature model up to certain levels of detail to stakeholders. To this end, we aim at providing a more stakeholder oriented view of feature models where goal-oriented requirements engineering (GORE) [111] is employed to generate a stakeholder view of feature models.

GORE makes extensive use of goal models, i.e., models capturing user intentions for a system-to-be and facilitates the exploration of design alternatives, described in high-level non-technical terms. Typically, goal models are also used by the SPLE community in both life-cycles of SPLE. In domain engineering, they are applied for a top down development of SPLs [190, 191, 86]. In application engineering, they ensure the selection of features that are based on the objectives of a target application stakeholder [11].

In essence, goal models and feature models provide different variability perspectives. Goal models represent intentional variability, which is different in objectives of stakeholders and the way stakeholders may use a system-to-be to reach their objectives [190]. On the other hand, feature models are commonly used to illustrate variability between various systems, which is called product line variability [102]. When applying goal models in the context of SPLE, we need to capture and represent the stakeholders' objectives of several products. Hence, not only should the goal models be able to represent the objectives of the stakeholders of various products, but they also should be able to distinguish between objectives of different products. In other words, the goal model should illustrate product line variability in the intentional space which refers to differences in

intentional spaces of product line members. Standard goal model languages like i* and GRL can represent intentional variability, but lack mechanisms for representing differences between intentional spaces of various systems (i.e., product line variability in the intentional space). Hence, we introduced the notion of *family goal model* by extending standard goal modeling techniques (see Section 8.3.1).

The family goal model and the feature model are connected by mappings, which provide bidirectional relationships and traceability links between high-level business objectives of stakeholders, described by goal models and implementation units encapsulated within features in feature models. In this paper, we refer to the combination of the family goal model, the feature model, and the mapping model as a *family requirements model*. Due to the different perspectives of family goal models and feature models, the variability semantics might be different in both models. Also, these models might be developed by different stakeholders. Moreover, due to technical constraints, relations in feature models may be changed by software designers during the development of a product line. All these factors may lead to inconsistency between family goal model relations and feature model relations, which limit the product configuration within the application engineering life-cycle since several feature selections could lead to non-satisfaction of stakeholders' intentions.

To remedy these problems, we present a validation approach that can detect inconsistencies in variability between goal and feature models already in the *domain engineering* life-cycle, i.e., based on mappings between goals and features, independent of a particular feature selection. We establish a family requirements model and we provide an approach for the validation of the family requirements model. The approach does not only ensure the alignment of feature selections to user intentions but also prevent the high cost of changing design and implementation models in last phases of domain engineering to align them with variability in intentions of stakeholders.

In particular, this paper makes the following contributions: (1) a goal model profile extension (i.e., family goal model) to represent intentions of product lines and the difference of intentions for various products; (2) a representation and formal definition of family goal models, feature models, and their mappings in Description Logic (DL); (3) a logic-based reasoning through standard reasoning mechanisms for the discovering of inconsistencies in a family requirements model; (4) support for maintainability and traceability between goal and feature models; (5) formal and empirical evaluation of the

proposed approach.

The paper is structured as follows: Section 8.3 explains the role of goal models in the software product line domain and introduces the notion of family goal models. The family requirements model and a set of inconsistency patterns, which may happen in the family requirements model, are explained in Section 8.4. After providing a representation of the family requirements model in Description Logic in Section 8.5, the validation procedure for identifying inconsistencies in family requirements model is explained in Section 8.6. After describing the evaluation of the inconsistency checking algorithms in Section 8.7 and discussing related work in Section 8.8, concluding remarks and directions of the future work are highlighted in Section 8.9.

8.3 Goal Models in the SPLE Life-Cycle

Goal and feature models are used for different purposes. In order to adopt goal models in the development life-cycles of software product lines, we continue with a foundational analysis of the different modeling constructs in both types of models.

8.3.1 Family Goal Models and Feature Models

However, model elements and the variability among these elements are differently captured in both models. To outline this, in this section, we present distinctions between goal models and feature models and compare variability types in both models.

We refer to goal models when employed in product lines as *family goal models*, which represent the intentional space of a domain for which the product line is developed. Several works proposed a set of transformation rules to produce feature models from family goal models [186, 10]. However, such approaches do not take into account the details and differences between these two models.

A family goal model is an artifact representing stakeholders' objectives and strategies. It describes the intentional space of stakeholders of a domain[2]. Feature models have a different purpose. They describe the configuration space of a product line (i.e., a set of products). Accordingly, they represent characteristics of software products that belong to a product line. Features that are assigned to goals can be seen as the realization

[2]The stakeholders includes final users, managers, designers, clients, etc. However, in the rest of the paper, we only concentrate the final users and representation of their intentions in goal models.

of requirements (solutions) in the software products. One or more features may be developed for realizing one or more tasks in goal models. Different terms are defined for goals: achieve, maintain, avoid, and cease, while features can only be selected or deselected [191].

The relationships have also different meanings in both models. Intentional relations in family goal models show decompositions of stakeholders' goals into subgoals, and further to low-level subgoals and tasks [90]. An intentional decomposition (except for soft goals) in a goal model is an entailment, i.e., an AND-decomposition of source intentional elements is one possible combination (among others) of source intentional elements that implies the target intentional element. On the other hand, an AND-decomposition in a feature model is a *PartOf* relation that implies a parent feature contains its child features.

With respect to the variability in the context of software product lines, we also distinguish between *product line variability* and *behavioral variability* [102]. Product line variability refers to differences between products in a product line, which may exist among their requirements, design models and implementation models [181]. On the other hand, behavioral variability represents the various behavior that a single system may be used by its user [92]. For example, workflow patterns in process modeling languages such as BPMN, BPEL, and activity diagrams provide mechanisms for representing variability in behavior of a single system.

In family goal models, OR and XOR relations represent variability in stakeholders' goals (i.e., intentional variability), and the ways that stakeholders' goals can be achieved (i.e., behavioral variability). When developing a reference design and implementation models from the family goal model, these variability relations can lead to either *intentional/behavioral variability* or *product line variability* in the reference models. In the family goal model, OR-decompositions (and similarly XOR-decompositions) represent *intentional/behavioral variability*, if all products having a target intentional element involved in the OR relation (XOR relations), contain all source intentional elements of the OR-decomposition (XOR-decomposition) in their goal models. For example, as shown the Figure 8.1, a final user in an on-line shop may variably choose to perform Check if Return Customer or Check Credit tasks in order to satisfy the hard goal Trustworthiness of the Customer Determined, even though both tasks are invariably available for them in all products.

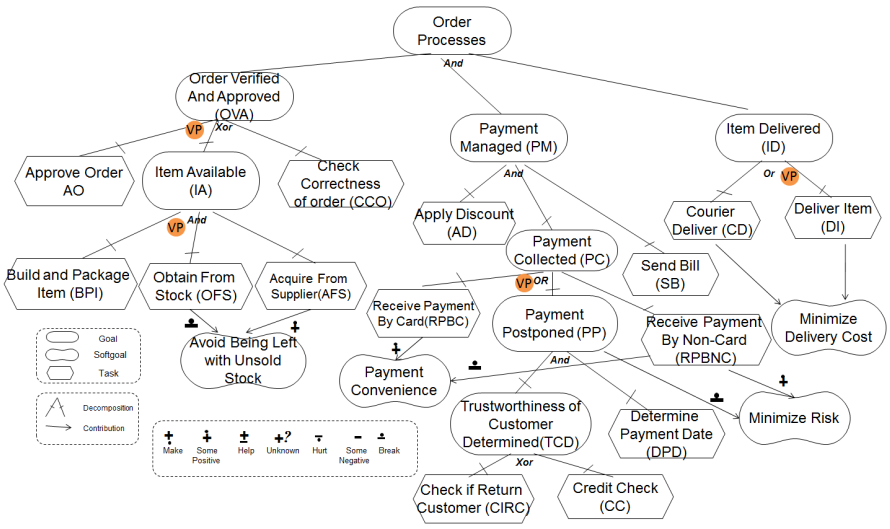


Figure 8.1: Goal Model of the e-store Case Study.

On the other hand, in the family goal model, OR-decompositions (and similarly XOR-decompositions) represent product line variability if source intentional elements involved in OR-decompositions (XOR-decompositions) vary between different products that contain the target intentional element. For example, in Figure 8.1, some products provide Receive Payment by Card or Payment Postponed to achieve Payment Collected, while other products offer Payment Postponed and Receive Payment By Non-Card to achieve Payment Collected goal.

Feature models aim at modeling differences between products of a product line [22, 74, 145]. Therefore, feature models only represent product line variability and variability related to a single system is represented in feature models as commonalities. For example, since XOR relation between Check if Return Customer and Check Credit in the family goal model is a behavioral variability, the features mapped to those tasks are considered as mandatory features. On the other hand, since the OR intentional relation between Deliver Item and Courier Deliver is a product line variability, there is a variability relation between their corresponding features (see Figure 8.5). The variability relations in feature models are resolved when a product is derived from a product line, i.e., when a new configuration is created.

We should note that a family goal model represents stakeholder's requirements and

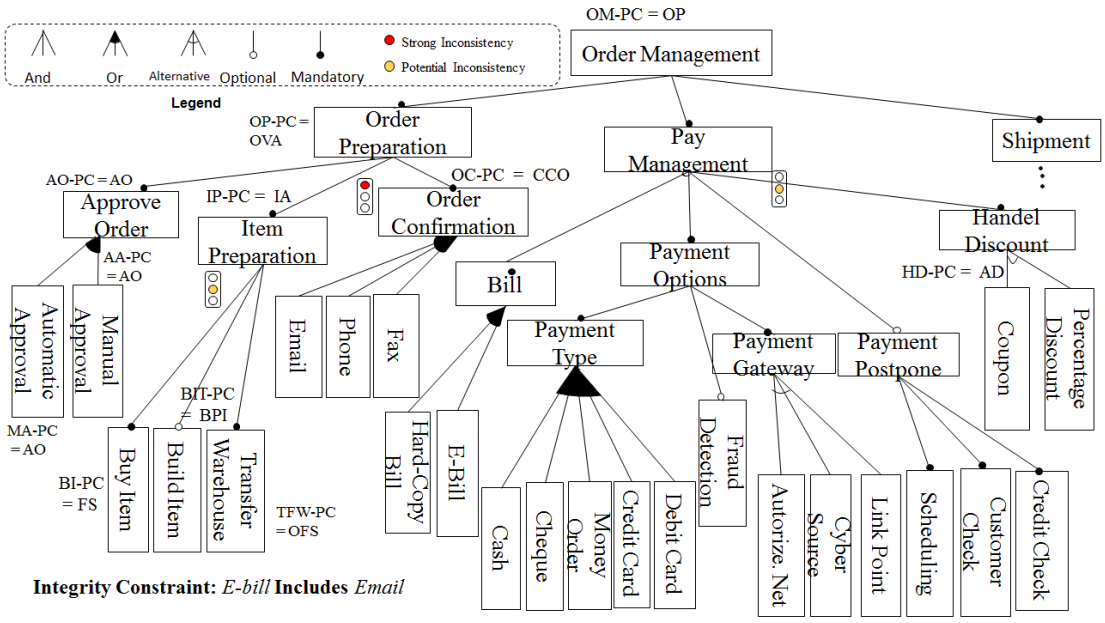


Figure 8.2: Feature Model of the online shopping Case Study

their variability. On the other hand, feature models not only show variabilities in requirements but also encapsulate product line variabilities in designing and implementation models [74]. Therefore, a feature model may contain several variability relations that do not exist in the goal model. For example, as shown in figure 8.2 there is alternative relation between Autorize. Net, Cyber Source, and Link Point features for implementing Payment Gateway feature, which shows variability in the level of a product line implementation. However, this variability relation does not exist in the family goal model, because Receive Payment By Card task is not concerned with different ways of implementation.

Having investigated the notion of variability in the family goal models and feature models, we found out that the semantics of variability in these two models are different. Variability relations in goal models can be either product line variability or intentional/behavioral variability while feature models only represent the product line variability.

The existing goal model notations do not discriminate between product line variability and intentional/behavioral variability. Thus, we extend the standard goal model notation, in order to distinguish the types of variability in the family goal model.

For OR-decompositions (XOR-decompositions), the product line variability and intentional/behavioral variability are distinguished using the *VP* notation. OR-decompositions, showing product line variability, are labeled as VP. For example, the VP annotation on the Item Delivered goal in Figure 8.1 shows that different products having the goal Item Delivered vary in ways they provide fulfillment of this goal for their final users (i.e., Delivered By Courier or Delivered By Company). Therefore, during the development time, at least one of source intentional elements might be selected for the target product in goal model.

In standard goal modeling [186, 173, 68], AND-decomposing a target intentional element (goals or tasks) into source intentional elements implies that the satisfaction of the target intentional element is dependent on the satisfaction of all of its sources. However, in family goal models there may be some source of intentional elements in AND-decompositions whose fulfillment are necessary for the target intentional element of a particular product, but their non-fulfillment does not make the target intentional element unsatisfiable in other products. To enable family goal models to present and describe these situations, we add the notion of optional goals, resembling optional features in feature model. Hence, intentional elements in an AND-decomposition can be optional if their non-fulfillment does not lead to non-fulfillment of the target intentional element. For example, Figure 8.1 shows that non-fulfillment of task Apply Discount does not necessarily lead to the non-satisfaction of the goal Payment Managed.

In family goal models, we use soft goal as means for resolving product line variability in the intentional space. Therefore, contribution links can propagate the desired satisfaction level of soft goals into goals and tasks and help in the selection of a proper variant of product lines based intentional variability. For example, the Minimize cost delivery soft goal can be used as criteria to resolve product line variability in the Item Delivered goal. We formally define family goal models as follows:

Definition 8.1 (Family Goal Model) *A family goal model* $FGM = \langle \mathcal{G}, \mathcal{C}, \mathcal{D}^F \rangle$ *extends a goal model* $GM = \langle \mathcal{G}, \mathcal{C}, \mathcal{D} \rangle$ *as follows: The decomposition relation* $\mathcal{D}$ *is extended by decompositions that cover product line variability* $\mathcal{D}^F \subseteq \mathcal{G} \times \{IOR, XOR, AND, AND\text{-}O, IOR\text{-}VP, XOR\text{-}VP\} \times \mathcal{P}(\mathcal{G})$.

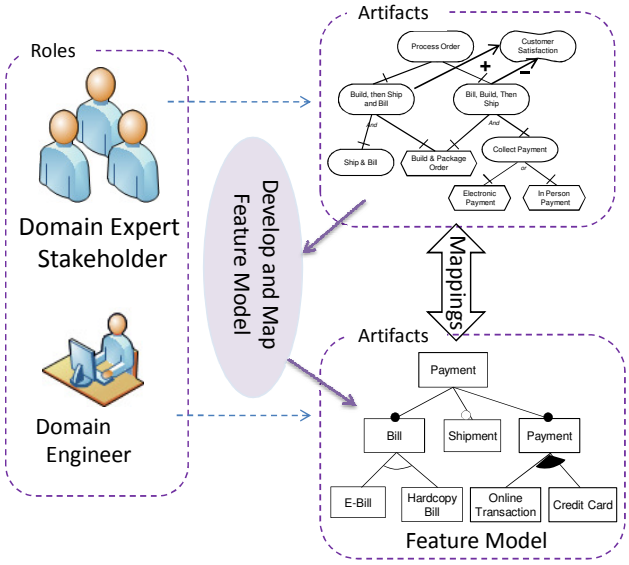
8.3.2 Development Life-Cycles

In our approach, we use goal models in both life-cycles of SPLE. Figure 8.3 shows a family goal model and a feature model in combination with mappings between them in the domain engineering life-cycle, as well as the application goal model and configuration in the application engineering life-cycle.

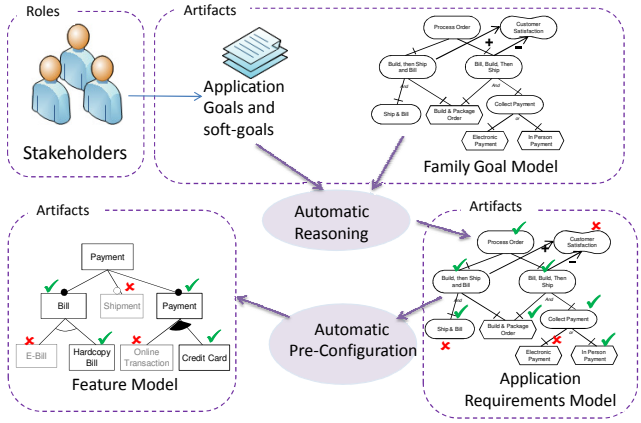
In the *domain engineering* life-cycle (see Figure 8.3(a)), one of the most important issues are the elicitation, representation and management of different stakeholders' functional and non-functional requirements. This is reflected by the goal model. Hence, in domain requirements engineering phase, first the family goal model is created by domain engineers. High level goals (e.g., Order Processed) and soft-goals (e.g., Minimize Risk) of the stakeholders are discovered and then the high level goals are decomposed into lower level goals and finally tasks. Goal decomposition is done by following the framework proposed by Liaskos et al. [91] where intentional variability concerns are recognized for each goal. Then, the goals are refined according to the variability concerns. After refining goals, requirements engineers analyze the impacts of each sub-goal on the soft-goals and model the impacts using contribution links. After developing the family goal model,using the proposed extensions (i.e. VP and optional), the family goal model is analyzed with respect to product line variability and the goals are annotated with the proper annotations.

The feature model describes elements of a system, their functionality and how different elements depend on each other. Finally, relationships between stakeholders' goals and SPL features, which realize these goals, are represented by mappings between goals and features, describing a realization of goals by system features (cf. [11]).

The feature model is generated from the family goal model by converting product line variability into variability relations in the feature model. In the family goal model, goals are finally refined to tasks, which show the requirements for designing a product line. These tasks are the sources for designing features in the feature model. Hence, features are developed based on the tasks in the family goal model and mapped to the corresponding tasks. Mappings represent the realization of intentional elements by features. One feature can operationalize several intentional elements, and one intentional element can be realized by several features. We propose a mapping model which is based on template-based approach proposed by Czarnecki et al. [33]. Although mapping between features and tasks may require domain engineers effort, this mapping needs



(a) Domain Engineering



(b) Application Engineering

Figure 8.3: Contribution to the SPLE Life-Cycle

to be created only once. Our approach only requires that atomic tasks in the family goal are mapped to the features, which provide realization for those features. Also, mapping features in the feature model to development artifacts is a common practice in the software product line engineering [66], which is required for further reusability in application engineering life-cycle. After designing features for realizing tasks in the family goal model, the feature model is generated by developing variability relations between features based on the variability relations in the family goal model.

In the *application engineering* life-cycle (see Figure 8.3(b)), application engineer communicates and understands the stakeholders' needs and requirements by identifying their objectives. The family goal model is used as a reference model for communicating with the customers and capturing their goals. Afterwards, a particular application goal model is obtained based on an individual stakeholder's goals and business objectives and by executing the backward reasoning algorithm [54]. Accordingly, based on the mappings between goals and features, a pre-configuration process is executed and features which are not based on the current stakeholders' objectives are filtered out from the feature model, by preserving the feature model constraints. The details of the pre-configuration process is out of scope of this paper. The detailed process of pre-configuration is outside of the scope of this paper. Interested readers can find more details about the pre-configuration process in our previous work [11].

8.4 Goal-Oriented Requirements Engineering for SPLs

The comparison of goal and feature models and their variability resulted in an extension of goal models, the so-called *family goal models*, which present different notions of variability. In the following, we specify the *realization* of goals from a family goal model by features of a feature model in terms of mappings. Based on these mappings, we specify *realization inconsistencies* that might happen due to contradicting relationships of goals and their mapped features.

8.4.1 Relating Intentional Elements to Features

In the domain engineering life-cycle, goal models capture intentional variability [190, 91] and describe the intentions behind existing features in the software product line. Hence, using the goal model, we can ensure that existing features and variability relations in

feature models are aligned with intentional variability in the goal models. We can also trace back differences in products to differences in the intentions of the stakeholders.

In our model, for representing an explicit mapping (i.e., a mapping indicated by domain engineers), a mapping relation for each mapped task is developed. For example, the Receive Payment By Card task is mapped to the Debit Card Payment, Credit Card Payment and Payment Gateway features, hence, a mapping relation $\Phi_{RPBC}(ReceivePayment\ ByCard, \{DebitCardPayment, CreditCardPayment, PaymentGateway\})$ is created. If a feature is mapped to more than one goal or/and task, then the corresponding feature appears in the mapping relations of all those goals or/and tasks. After explicit mapping between tasks in a family goal model and features in a feature model, we can drive implicit mappings between intermediate tasks and goals and features through existing relations in goal models and feature models. For example, we can infer that the goal Payment Managed in the family goal model is implicitly mapped to the feature Payment Management (see Figure 8.1).

Definition 8.2 specifies mappings between a family goal model and a feature model, as well as the resulting *family requirements model*, which is the combination of both models including the mappings between them.

Definition 8.2 (Mapping and Family Requirements Model) *Let* $FGM = \langle \mathcal{G}, \mathcal{C}, \mathcal{D}^F \rangle$ *be a family goal model where* $\mathcal{G} = (\mathcal{G}_g \cup \mathcal{G}_t \cup \mathcal{G}_s)$ *and* $FM = \langle \mathcal{F}, \mathcal{F}_M, \mathcal{F}_O, \mathcal{F}_{IOR}, \mathcal{F}_{XOR}, \mathcal{F}_{incl}, \mathcal{F}_{excl} \rangle$ *a feature model,* $\Phi_i(G_i, \mathcal{F}_i)$ *is a mapping relation between a intentional element* $G_i \in (\mathcal{G}_g \cup \mathcal{G}_t)$ *and a set of features* $\mathcal{F}_i \subset \mathcal{F}$*, and* Φ *denotes the set of all mappings between FGM and FM. A family requirements model* Π *is defined as a triple of FGM, FM and* Φ*:* $\Pi = \langle FGM, FM, \Phi \rangle$*.*

8.4.2 Realization Inconsistency

In a family requirements model $\Pi = \langle FGM, FM, \Phi \rangle$, mappings Φ describe the realization of goals by features, while goals depend on intentional relations *IR* and features depend on feature relations *FR*. Relations in a family goal model capture the relations among objectives of stakeholders, while feature relations originate from intentional relations in the family goal model. This means that variability and commonality in feature models should be aligned with intentional relations defined in family goal models. Therefore, there is an inconsistency in a family requirements model if intentional relations do not coincide with the feature relations. Hence, we aim at developing a logical based technique

that detects whether the intentional/behavioral relationships in a family goal model are correctly implemented with the commonility and variability relationships in feature models.

An inconsistency can only happen if source and target elements of both relations (i.e., *IR* and *FR*) are mapped to each other. We make the following assumptions: (i) only hard goals and tasks are mapped to features, since soft goals usually describe non-functional requirements; (ii) these mappings are collaboratively developed by domain experts and domain engineers. (iii) only contributions Make ($\dot{+}$) and Break ($\dot{-}$) are considered in the inconsistency detection of a family requirements model since only these contributions are sufficient for goal fulfillment; (iv) unmapped elements (i.e. elements which are not mapped explicitly or their implicit mapping can not be derived from existing relations in the family goal model and the feature model) do not contribute to inconsistencies; and (v) a mapping of a feature means implicitly also a mapping of the parent feature to the parent goal of the mapped goal (cf. mapping principles in [11]).

Assume *FR* is a feature relation, with target feature F and source features $F_1, \ldots, F_n$, i.e., F depends on $F_1, \ldots, F_n$. Likewise, *IR* is an intentional relation with target element $G \in \mathcal{G}$ and source intentional elements $G_1, \ldots, G_m \in \mathcal{G}$. The fulfillment of G depends on the fulfillment of $G_1, \ldots, G_m$. We distinguish between potential and strong inconsistency.

Definition 8.3 (Potential Inconsistency) *A permissible satisfaction of the intentional element G, which depends on the satisfaction of $G_1, \ldots, G_m$, might lead to an incorrect configuration of feature F, while F depends on $F_1, \ldots, F_n$.*

Definition 8.4 (Strong Inconsistency) *All permissible satisfactions of G, which depend on the satisfaction of $G_1, \ldots, G_m$ with respect to IR, lead to an incorrect configuration of feature F (F depends on $F_1, \ldots, F_n$).*

Figure 8.2 depicts an example of a strong inconsistency between a feature relation and an intentional relation. The feature Oder Preparation has three mandatory children (Approve Order, Item preparation and Order Confirmation). The corresponding goal Order Verified and Approved (OVA) has exclusive subgoals, which are mapped to the children of Order Preparation. Thus, no goal fulfillment of OVA will lead to a valid feature configuration.

Another example illustrates a potential inconsistency. Feature Buy Item is mapped to task Acquire From Supplier (AFS), feature Build Item is mapped to task Build and Package

Table 8.1: Correspondence between Intentional Relations and Feature Relations

Features Relations	Intentional Relations							
	AND	Optional	IOR	IOR-VP	XOR	XOR-VP	$\dot{+}$	$\dot{-}$
Parent-Mandatory Child	✓	±	✓	±	✓	↯	✓	↯
Parent-Optional Child	✓	✓	✓	✓	✓	✓	✓	✓
OR Feature Group	✓	✓	✓	✓	✓	✓	✓	±
Alternative Feature Group	↯	✓	↯	✓	↯	✓	↯	✓
Include Relation	✓	±	✓	±	✓	↯	✓	↯
Exclude Relation	↯	✓	↯	✓	↯	✓	↯	✓

Legend: no inconsistency (✓), strong (↯) and potential (±) inconsistency. $\dot{+}$ and $\dot{-}$ are sufficient contribution links. IOR and XOR show intentional variability, which are converted to behavioral variability and should remain for run-time. IOR-VP and XOR-VP show intentional variability, which is transformed to product line variability and should be resolved during the configuration of products.

Item (BPI), and finally, feature Transfer from Warehouse is mapped to the task Obtain from Stock (OFS). By mapping goals to features, the implicit mapping between parent goals (e.g., Item Available) and parent features (e.g., Item Preparation) is established. Let us assume that an application engineer wants goal Item Available be satisfied in a target product. This can be achieved by selecting Build and Package Item (BPI) to be fulfilled, but not Acquire From Supplier (AFS) and Obtain from Stock (OFS). However, features Transfer from Warehouse and Obtain from Stock are mandatory features and removing them from the feature model violates the feature model relations. Thus, this can lead to a potential inconsistency.

Table 8.1 shows combinations of intentional relations (*IR*) and feature relations (*FR*). The comparison between contributions ($\dot{+}$ and $\dot{-}$) to feature groups means that the goals are mapped to the siblings of the feature group.

Among intentional relations, the relations IOR-VP, XOR-VP, and AND-Optional depict product line variability (difference in intentional elements of different products) and the other intentional relations illustrate intentional/behavioral variability (variability in the intentional elements of the stakeholders of a product) in the intentional space. Accordingly, the former relations should be aligned with variability relation in feature models (i.e., Optional, Alternative, and OR) and the latter should be aligned with commonality (i.e., Mandatory relation) in the feature model.

8.5 Knowledge Base of the Family Requirements Model

In order to recognize inconsistencies in family requirements models, we need a formal representation formalism that captures intentional relations, feature relations and mappings between goals and features. Furthermore, we need (automatic) means to (automatically compare these relationships and pinpoint the source of an inconsistency. Following this line of argumentation, we propose a formal knowledge representation that offers reasoning (or inference) services to facilitate model validation. We use Description Logic (DL) as it is expressive enough to represent all kinds of elements (i.e., goals and features) and relationships and mappings between them. Besides this, DL offer quite efficient, sound and complete reasoning services.

DL modeling has been used to align between different models and to use reasoning for model verification. For instance, in [57], we proposed an approach for inconsistency detection in design artifact of services (represented in business process model) with respect to requirements that are represented in goal models. In this work, our DL model has to cover three different kinds of variability that are given by the family goal model and by the feature model.

8.5.1 Representation of Models and Mappings

The key part of our modeling formalism is to represent the different relations of both models, combined with mappings between features and goals.

Intentional Relations.

A family goal model $FGM = \langle \mathcal{G}, \mathcal{C}, \mathcal{D}^F \rangle$ contains the *intentional relations (IR)* on goals $G \in \mathcal{G}$. The corresponding DL knowledge base Σ_{FGM} is built according to Algorithm 4. For each goal G, we represent its relations by the concept Rel_G. Relationships between goals are expressed in DL by the role *require*.

Lines 4–6 capture an IOR-decomposition, in which a goal G is satisfied only if at least one of its subgoals G_i is satisfied. This is represented in DL by a concept union over G_i. We use the role *relates* to express relationships between goals. The representation of exclusive decompositions is straightforward (lines 7–9).[3] Conjunctive decompositions,

[3]Please note that $\otimes$ is not a standard operator in DL. For a more concise representation, we use $\bigotimes_{G' \in \{G_1,\ldots,G_n\}} \exists requires.G'$ as an abbreviation for $\bigsqcup_{G' \in \{G_1,\ldots,G_n\}} \exists requires.G' \sqcap \neg(\bigsqcup_{G'',G''' \in \{G_1,\ldots,G_n\}} (\exists requires.G'' \sqcap \exists requires.G'''))$.

Algorithm 4 Representation of the Intentional Relations Σ_{FGM}

1: Input: Family Goal Model $FGM = \langle \mathcal{G}, \mathcal{C}, \mathcal{D}^F \rangle$
2: Output: Knowledge base Σ_{FGM}
3: **for all** $(G, \gamma, \{G_1, \ldots, G_n\}) \in \mathcal{D}$ **do**
4: **if** $\gamma = IOR\text{-}VP$ **then**
5: $Rel_G := Rel_G \sqcap (\bigsqcup_{i=1,\ldots,n} \exists requires.G_i)$
6: **end if**
7: **if** $\gamma = XOR\text{-}VP$ **then**
8: $Rel_G := Rel_G \sqcap (\bigotimes_{G' \in \{G_1,\ldots,G_n\}} \exists requires.G')$
9: **end if**
10: **if** $\gamma \in \{AND, IOR, XOR\}$ **then**
11: $Rel_G := Rel_G \sqcap (\bigsqcap_{(i=1,\ldots,n) \wedge (\neg(G \circ G_i))} \exists requires.G_i)$
12: **end if**
13: **end for**
14: **for all** $(G', \dagger, G) \in \mathcal{C}$ **do**
15: $Rel_G := Rel_G \sqcap \exists requires.G'$
16: **end for**
17: **for all** $(G', \dot{-}, G) \in \mathcal{C}$ **do**
18: $Rel_G := Rel_G \sqcap \neg \exists requires.G'$
19: **end for**

as well as OR decompositions, which represent behavioral variability, are described by concept intersections (lines 10–12), optional goals are neglected, as the fulfillment of an optional goal depend on an individual requirement selection and cannot be determined in the domain engineering life-cycle.

Sufficient positive contributions (lines 14–16) specify that the fulfillment of G requires the fulfillment of G'. Thus, we add the expression $\exists requires.G'$ to the definition of concept Rel_G. Sufficient negative contributions (lines 17–19) use concept negation in order to represent the exclusiveness of goals G and G'.

Axioms 8.1 and 8.2 exemplify the DL representation for an excerpt of the goal model of Figure 8.1. An AND-decomposition of the goal OP (Order Processed) into subgoals OVA (Order Verified and Approved), PM (Payment Managed) and ID (Item Delivered) is described in Axiom 8.1. An IOR-VP-decomposition of the goal ID (Item Delivered) is given in Axiom 8.2.

$$Rel_{OP} \equiv \exists\, requires.OVA \sqcap \exists\, requires.PM \sqcap \exists requires.ID \tag{8.1}$$

$$Rel_{ID} \equiv \exists\, requires.CD \sqcup \exists\, requires.DI \tag{8.2}$$

Algorithm 5 Representation of the Feature Model Knowledge Base Σ_{FM}

1: Input: Feature Model $\langle \mathcal{F}, \mathcal{F}_M, \mathcal{F}_O, \mathcal{F}_{IOR}, \mathcal{F}_{XOR}, \mathcal{F}_{incl}, \mathcal{F}_{excl} \rangle$
2: Output: Knowledge base Σ_{FM}
3: **for all** $F \in \mathcal{F}$ **do**
4: $\quad Rel_F \equiv \top$
5: **end for**
6: **for all** $(F, \{F_1, \ldots, F_n\}) \in \mathcal{F}_M$ **do**
7: $\quad Rel_F := Rel_F \sqcap \bigsqcap_{i=1,\ldots,n} \exists requires.F_i$
8: **end for**
9: **for all** $(F, IOR, \{F_1, \ldots, F_n\}) \in \mathcal{F}_{IOR}$ **do**
10: $\quad Rel_F := Rel_F \sqcap \bigsqcup_{i=1,\ldots,n} \exists requires.F_i$
11: **end for**
12: **for all** $(F, XOR, \{F_1, \ldots, F_n\}) \in \mathcal{F}_{XOR}$ **do**
13: $\quad Rel_F := Rel_F \sqcap (\bigotimes_{F' \in \{F_1,\ldots,F_n\}} \exists requires.F')$
14: **end for**
15: **for all** $(F, F') \in \mathcal{F}_{incl}$ **do**
16: $\quad Rel_F := Rel_F \sqcap \exists requires.F'$
17: **end for**
18: **for all** $(F, F') \in \mathcal{F}_{excl}$ **do**
19: $\quad Rel_F := Rel_F \sqcap \neg\exists requires.F'$
20: **end for**

Feature Model Relations.

Similar to goal models, the feature model relations *FR* of a feature model *FM* are represented in a DL knowledge base Σ_{FM} (Algorithm 5). The DL representation is based on the general modeling principles of Wang et al. [177]. However, due to the different validation purpose, we adapt some modeling principles according to our particular need. We use only one role requires to describe the relations of a feature that requires other features, while Wang et al. use different roles. It is easier and more intuitive to compare concept expressions that use the same role. We use concept definitions (equivalence axioms) in order to allow for a subsumption checking between the different concepts that represent intentional and feature relations (cf. Section 8.6).

Initially, for each feature F, Rel_F is equal to the universal concept (line 4 in Algorithm 5), to capture the case that a feature does not depend on any other feature. All mandatory child features of a feature F are represented by a concept intersection (lines 6–8). An inclusive OR decomposition of a feature F into features $F_1, \ldots, F_n$ is

represented by a concept union over the mapping concepts of each feature F_i (lines 9–11). Likewise, XOR decomposition are described by concept unions, but with a further restriction that the selection of only one feature is allowed (line 12–14). The includes integrity constraint specifies that the selection of a feature F also requires the selection of a feature F'. In DL, we define Rel_F dependent of the feature F' (lines 15–17). The excludes integrity constraint is defined similarly (lines 18–20).

Axiom 8.3 defines features Order Preparation, Pay Management and Shipment as mandatory children of Order Management. An exclusive grouping of the features Coupon and Percentage Discount is depicted by Axiom 8.4. The second part of the axiom excludes the selection of multiple child features. Axiom 8.4 describes an integrity constraint, i.e., feature E-bill includes feature E-mail.

$$\begin{aligned} Rel_{OrderManagement} &\equiv \exists\, requires.OrderPreparation \sqcap \exists\, requires.PayManagement \\ &\quad \sqcap \exists\, requires.Shipment \qquad (8.3) \\ Rel_{HandelDiscount} &\equiv \exists\, requires.Coupon \otimes \exists\, requires.PercentageDiscount \\ Rel_{E\text{-}bill} &\equiv \exists\, requires.E\text{-}mail \qquad (8.4) \end{aligned}$$

Mapping Representation.

Besides intentional relations IR and feature relations FR, we have to represent the realization of goals by the corresponding features in terms of mappings in the knowledge base Σ_Φ. A mapping is described as a concept equivalence in the knowledge base. If there is a mapping $\phi_i(G_i, \mathcal{F}_i)$ ($\phi_i \in \Phi$) from a goal G_i to the set of features $\mathcal{F}_i$, we represent the mapping by an axiom $G \equiv \mathcal{P}(\mathcal{F}_i)$ where $\mathcal{P}$ shows propositional formula over features in $\mathcal{F}_i$. Axiom 8.5 shows the mapping relation between the task Receive Payment By Card and the features Debit Card Payment, Credit Card Payment and Payment Gateway.

$$\begin{aligned} Rel_{ReceivePaymentByCard} &\equiv (Rel_{DebitCardPayment} \sqcup Rel_{CreditCardPayment}) \\ &\quad \sqcap Rel_{PaymentGateway} \qquad (8.5) \end{aligned}$$

8.6 Verification of Family Requirements Models

The verification aims at detecting inconsistencies between intentional relations in the family goal model and variability/commonality relations in the feature model. Hence, the verification compares the intentional and feature relations of mapped elements. For each mapping, we check whether there is a strong or potential inconsistency, or even no inconsistency, according to the correspondences of Table 8.1.

8.6.1 Verification Procedure

The knowledge base contains DL concepts Rel_G and Rel_F that describe intentional relations IR of G and feature relations FR of F. From a logical point of view, concepts Rel_G and Rel_F represent formulas, and we compare them in order to analyze the influence of Rel_G on Rel_F. (i) A potential inconsistency is identified if the satisfaction of IR does not necessarily imply the satisfaction of FR (expressed by Rel_F). Thus, the concept Rel_G is not subsumed by Rel_F, i.e., Rel_G does not imply Rel_F. (ii) A strong inconsistency is recognized by contradicting relations of goal G and feature F. Thus, the intersection of Rel_G and Rel_F is unsatisfiable, i.e., the intersection is subsumed by the empty concept $\bot$ in DL.

Accordingly, we check either whether $Rel_G \sqcap Rel_F \sqsubseteq \bot$ (strong inconsistency) or if the implication $Rel_G \Rightarrow Rel_F$ holds, i.e., $\neg Rel_G \vee Rel_F$ is a tautology (no potential inconsistency). In DL, this is represented by a concept union: $\neg Rel_G \sqcup Rel_F$. For this purpose, we extend the knowledge base by verification concepts $Valid_{G \wedge F}$ (strong inconsistency) and $Valid_{G \Rightarrow F}$ (potential inconsistency) for each mapped elements (G, F) ($\phi(G, \mathcal{F})$ with $F \in \mathcal{F}$) (cf. Definition 8.5).

Definition 8.5 (Knowledge Base Σ for the Verification) *The knowledge base* $\Sigma := \Sigma_{FM} \cup \Sigma_{FGM} \cup \Sigma_{\Phi}$ *is extended as follows: For each mapped elements* (G, F) *with* $(\phi(G, \mathcal{F})\ F \in \mathcal{F})$ *in* Σ*, we insert the following axioms:*

(1) $Valid_{G \Rightarrow F} \equiv \neg Rel_G \sqcup Rel_F$ *(2)* $Valid_{G \wedge F} \equiv Rel_G \sqcap Rel_F$

Given the final knowledge base, we get the verification result *en passant*. We classify the verification concepts $Valid_{G \Rightarrow F}$ and $Valid_{G \wedge F}$ of the knowledge base Σ, leading to the following observations:

1. A verification concept $Valid_{G \Rightarrow F}$ indicates a potential inconsistency or no inconsistency. If $Valid_{G \Rightarrow F}$ is classified equal to the universal concept $\top$ we can guarantee that the fulfillment of intentional relations IR of G ensure the fulfillment of feature relations FR of F.

2. Otherwise, $Valid_{G \Rightarrow F} \not\equiv \top$ holds, and we know that there is at least a potential inconsistency, but which kind of inconsistency is unknown. We identify a strong inconsistency if the other verification concepts $Valid_{G \wedge F}$ is classified equal to the empty concept $\bot$, otherwise it is a potential inconsistency.

As described in Section 8.3, if there is a mapping between a goals G and a feature F the fulfillment of G determines whether F will be removed or not.

8.6.2 Correctness of the Verification

For a family requirements model $\Pi = \langle FGM, FM, \Phi \rangle$, the verification recognizes inconsistencies between goal G and feature F, based on their relations IR and FR.

Relationship Coverage in the Knowledge Base

Relationships of both models are represented in a common DL knowledge base Σ. Intentional relations IR of a goal G are represented by a single concept Rel_G. Likewise, feature relations FR of a feature F are covered by a concept Rel_F. Mappings between goals and features are represented by equivalence axioms in the knowledge base. For each mapping, the corresponding concepts Rel_G and Rel_F are compared in order to determine the influence of intentional relations IR on feature relations FR.

Based on this representation, we compare whether Rel_G is subsumed by Rel_F or the intersection of them is a satisfiable concept. With respect to Table 8.1, the correspondences between intentional and feature relations are reduced to subsumption checking and satisfiability in DL.

Verification Principles

As a last step, we have to show that the detection of strong and potential inconsistency is correctly achieved in the verification, i.e., the classification of the verification concepts $Valid_{G \Rightarrow F}$ and $Valid_{G \wedge F}$ holds if there is an inconsistency between G and F. Lemma 8.6.2 summarizes these statements.

Let ($\phi(G, \mathcal{F})$ with $F \in \mathcal{F}$) be a mapping in a family requirements model Π the following holds: (i) the concept $Valid_{G \Rightarrow F}$ is classified equal to the universal concept $\top$, iff all dependent features of feature F appear in a feature configuration, whenever feature F, which is mapped to G, appears; and (ii) the concept $Valid_{G \wedge F}$ is classified equal to the empty concept $\bot$, iff there is no configuration possible where F appears when G is fulfilled.

We sketch only the proof for the first statement, but the proof of the second statement is based on the same argumentation.

'$\Rightarrow$' For a mapping ($\phi(G, \mathcal{F})$ with $F \in \mathcal{F}$), the concept $Valid_{G \Rightarrow F}$ is equal to the universal concept $\top$. We demonstrate that all dependent features of feature F will appear in a configuration that contains F. Let $F_1, \ldots, F_n$ be the dependent features of feature F. From the classification of $Valid_{G \Rightarrow F}$, we know that the subsumption $Rel_G \sqsubseteq Rel_F$ holds, while both concept definitions contain the relationships of goal G and feature F and the same roles are used in both concept definitions. The subsumption $Rel_G \sqsubseteq Rel_F$ can only hold, if each dependent features F_i ($i = 1, \ldots, n$) of F is mapped to a goal G_j (i.e., $F_i \equiv G_j$) and the goal is a dependent goal of G, i.e., G cannot be fulfilled if G_j is not fulfilled. Therefore, F_i will be in each configuration that contains F.
'$\Leftarrow$' We demonstrate the other direction by contradiction. Assume F_i is a dependent feature of F that has to appear in each configuration if F appears, but $Valid_{G \Rightarrow F}$ is not equal to the universal concept $\top$. Either (i) the structures in the concept definitions of Rel_G and Rel_F are different, i.e., different types of relationships, or (ii) feature F_i is not mapped to goal G_j, where G_j is a dependent goal of G that appears in Rel_G, and F_i is a dependent feature of F that appears in Rel_F. In both cases, we can not guarantee that F_i occurs in a configuration if F occurs, because no dependent goal of G guarantees the selection of F_i. This is a contradiction to our assumption.

8.6.3 Verification Exemplified

We show how DL reasoning detects a potential and strong inconsistency between family goal intentional relations and variability and commonality relations in feature models, based on the examples in Figure 8.2.

Feature Item Preparation(IP) has two mandatory child features Buy Item (BI) and Transfer From Warehouse(TFW) (Axiom 8.6). Feature IP is mapped to goal Item Available(IA), and BI and TFW are mapped to goals Acquire From Supplier(AFS) and Obtain

From Stock(OFS), respectively. Goals AFS, OFS and BPI are subgoals within an OR-VP decomposition of goal IA (Axiom 8.7). Mappings make the concepts BI and AFS, as well as TFW and OFS equivalent.(BPI does not contribute to the inconsistency.)

$$Rel_{IP} \equiv \exists requires.BI \sqcap \exists requires.TFW \tag{8.6}$$

$$Rel_{IA} \equiv \exists requires.AFS \sqcup \exists requires.OFS \sqcup \exists requires.BPI \tag{8.7}$$

In this case, the verification concept $Valid_{IA \Rightarrow IP}$ is not equal to the universal concept $\top$, since even if the mapped concepts are equal, Rel_{IA} is not subsumed by Rel_{IP}.

Strong inconsistencies between relations in family goal models and feature model are recognized if the verification concept $Valid_{G \wedge F}$ is equal to the empty concept $\bot$. Consider the strong inconsistency from Figure 8.2. Feature Order Preparation(OP) is mapped to the goal Order Verified and Approved(OVA). Mandatory child features Approve Order(AO), Item Preparation(IP) and Order Confirmation(OC) are mapped to goals Approve Order(AO), Item Available(IA) and Check Correctness of Order(CCO), respectively. These three goals are in an XOR-VP decomposition, i.e., the exclude each other, while the corresponding features can only appear together. The feature relations are described by Axiom 8.8 as a concept intersection in DL, while the intentional relations is represented as a concept union that excludes the appearance of more than one goal (Axiom 8.9).

$$Rel_{OP} \equiv \exists requires.AO \sqcap \exists requires.IP \sqcap \exists requires.OC \tag{8.8}$$

$$Rel_{OVA} \equiv \exists requires.AO \otimes \exists requires.IA \otimes \exists requires.CCO \tag{8.9}$$

As the XOR-expression ($\otimes$) contains the expression $\neg(\exists requires.AO \sqcap \exists requires.IA \sqcap \exists requires.CCO)$, the intersection of Rel_{OP} and Rel_{OVA}, which is the verification concept $Valid_{OVA \wedge OP}$, is equal to the unsatisfiable concept $\bot$, indicating a strong inconsistency.

8.7 Evaluation

In this section, we highlight two examples of inconsistencies, which happen in online shopping case study, and analyze the performance of the inconsistency detection algorithm.

8.7.1 Case Study Analysis

We investigate some parts of the online shopping case study available in the SPLOT repository[4]. The family goal model and feature model were developed based on the existing models of the SPLOT repository. We only concentrate on two common scenarios where inconsistency can happen in a family requirements model. These inconsistencies show where variability relations in online feature models are not aligned with intentional relations in the family goal model.

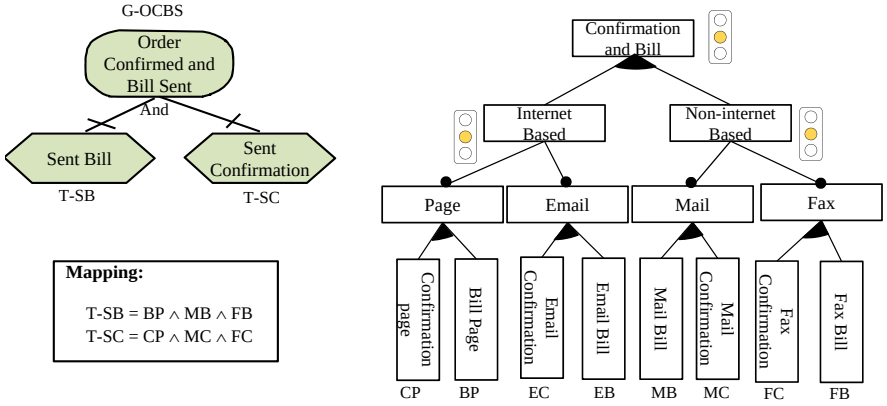
As shown in figure 8.4(a), there is a goal Order Confirmation and Bill Sent, which is AND-decomposed into Sent Bill and Sent Confirmation. Features corresponding to these tasks and their variability relation along with mapping relations are also shown in the figure. These mapped models have some potential inconsistencies. These inconsistencies are caused by the allowable selections of features that lead to the unsatisfaction of the Order Confirmed and Bill Sent goal. For example, Figure 8.4(b) shows two instances of the feature model configurations that do not lead to the satisfaction of Order Confirmed and Bill Sent. These inconsistencies are since 1) the variability relations in the feature model are not aligned with variability relations in the family goal model; and 2) the mapping relations should be logical OR instead of logical AND between features that are mapped to tasks in the family goal model.

Figure8.4(c) shows the revised version of the feature model and mapping relations that ensure consistency between stakeholders intentions (i.e., goals), feature model, and mapping relations.

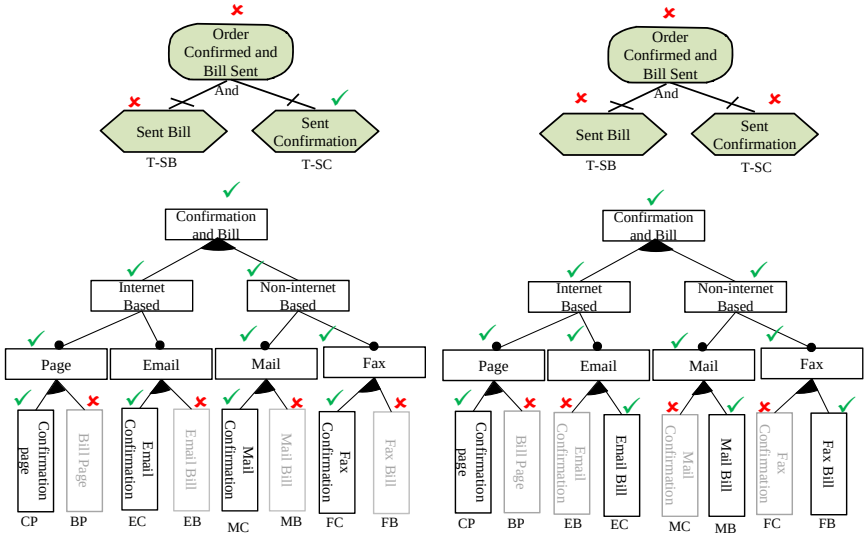
In the second example from this case study, we will show how our description logics-based reasoning approach helps detect inconsistencies. Therefore, throughout the discussion about this example, we will also show description logic axioms that are used for inconsistency detection. In particular, this example, shown in Figure 8.5, represents goal Item Shipped (G-IS), which is OR-decomposed to Courier Deliver(T-CD) and Deliver Item (T-DI) tasks. The *VP* notation over the OR-decomposition indicates a product line variability. After applying Algorithm 4, axiom 8.10 is generated.

$$Rel_{G-IS} \quad := Rel_{G-IS} \sqcap (\exists\ requires.T-CD \sqcup\ \exists\ requires.T-DI) \tag{8.10}$$

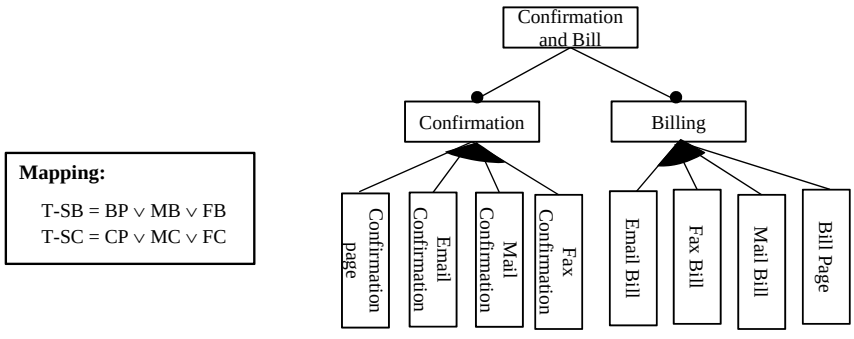
[4]Software Product Lines Online Tools- http://www.splot-research.org/



(a) Order Confirmed and Bill Sent Goal and corresponding features and mapping relations



(b) Sample Configurations of the feature models which lead to unsatisfaction of Order Confirmed and Bill Sent Goal



(c) consistent feature model and mapping relations

Figure 8.4: A part of goal model and its corresponding feature models

The feature model that corresponds to the Item Shipped goal, is shown in Figure 8.5. Feature Shipment (Sh) is OR-decomposed to the Shipping Gateway (SG) and Store Delivery (SD) features, where the former is further OR-decomposed into the FedEX (FE), UPS, Canada Post (CP), and USPS features. Additionally, the optional feature Shipping Cost Calculation (SCC) is developed which is required by the Shipping Gateway (SG) feature and excluded by the Store Delivery (SD) feature. The transformation of the feature model to Description Logic using Algorithm 5 is shown by Axioms 8.11 to 8.14.

$$Rel_{Shipment} := Rel_{Sh} \sqcap (\exists\ requires.SG \sqcup \exists\ requires.SD) \tag{8.11}$$

$$Rel_{SG} := Rel_{SG} \sqcap (\exists\ requires.FE \sqcup \exists\ requires.UPS \sqcup \exists\ requires.CP \sqcup \exists requires.USPS) \tag{8.12}$$

$$Rel_{SG} := Rel_{SG} \sqcap \exists\ requires.SCC \tag{8.13}$$

$$Rel_{SD} := Rel_{SD} \sqcap \neg\exists\ requires.SCC \tag{8.14}$$

The mapping model in figure 8.5 shows that the Courier Deliver task is mapped to the Shipping Gateway and Shipping Cost Calculation features and the Deliver Item task is mapped to the Store Delivery and Shipping Cost Calculation features. The transformation of the mapping relations into Description Logic generates Axioms 8.15 to 8.18:

$$Valid_{T-CD \Rightarrow \langle SG,SCC \rangle} \equiv \neg Rel_{T-CD} \sqcup (Rel_{SG} \sqcap Rel_{SCC}) \tag{8.15}$$

$$Valid_{T-CD \sqcap \langle SG,SCC \rangle} \equiv Rel_{T-CD} \sqcap (Rel_{SG} \sqcap Rel_{SCC}) \tag{8.16}$$

$$Valid_{T-DI \Rightarrow \langle SD,SCC \rangle} \equiv \neg Rel_{T-DI} \sqcup (Rel_{SD} \sqcap Rel_{SCC}) \tag{8.17}$$

$$Valid_{T-DI \sqcap \langle SD,SCC \rangle} \equiv Rel_{T-DI} \sqcap (Rel_{SD} \sqcap Rel_{SCC}) \tag{8.18}$$

After performing reasoning over the generated axioms, the results shows that both $Valid_{T-CD \Rightarrow \langle SG,SCC \rangle}$ and $Valid_{T-CD \sqcap \langle SG,SCC \rangle}$ equal to universal concept $\top$, which shows that there is no inconsistency for the mapping and relations in the family goal model and the feature model. However, results of the reasoning reveal that $Valid_{T-DI \sqcap \langle SD,SCC \rangle}$ and $Valid_{T-DI \Rightarrow \langle SD,SCC \rangle}$ are equal to the bottom concept $\bot$ (i.e., $Valid_{T-DI \sqcap \langle SD,SCC \rangle} \equiv$

$Valid_{T-DI \Rightarrow \langle SD,SCC \rangle} \equiv \bot$). This shows that there is a strong inconsistency between task Deliver Item and features Store Delivery and Shipping Cost Calculation. The inconsistency is because the Store Delivery feature excludes the Shipping Cost Calculation feature, while they both are mapped (using an AND relation) to Deliver Item. By changing the

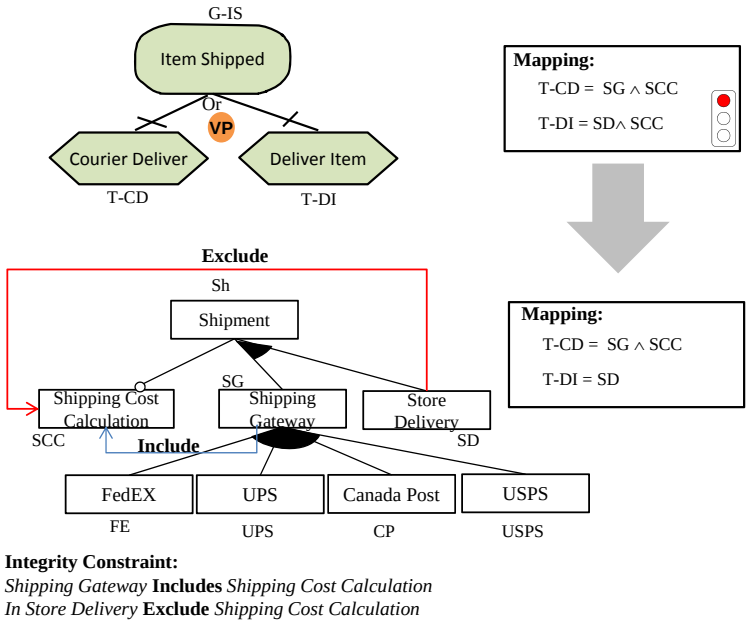


Figure 8.5: Item Shipped goal and its corresponding feature model

mapping (i.e. removing the mapping between Shipping Cost Calculation (SCC) feature and Deliver Item (T-DI) task), the inconsistency in the family requirement model can be resolved.

8.7.2 Performance Evaluation

The main objective of our study in this section is to analyze (1) the performance of the verification algorithm and (2) the factors of influence on the performance.

Scope of the Evaluation

In order to analyze the performance of the verification algorithms and the factors of influence on this verification approach, we summarize our investigations by the following four research questions:

- RQ1: How does the execution time of the inconsistency detection algorithms scale-up as the size of family requirements model increases?

- RQ2: Does the increase or decrease of product line variability in the family goal model have significant impact on the running time of inconsistency detection algorithms?
- RQ3: Does the percentage of the inconsistency types have major impact on the running time of algorithms?
- RQ4: Does the different distributions of intentional relations in the family goal model have significant impact on the running time of inconsistency detection algorithms?

Besides this, the study also serves as an empirical confirmation of the correctness of our verification approach.

Experimental Setting

In order to investigate the above research questions, we applied the simulation modeling technique by following guidelines similar to those proposed in [76]. We selected the simulation technique as it is commonly employed in the context of software product line verification and configuration [22, 61, 182].

Hence, we developed a generator, by utilizing the FAMA framework, to randomly developed family goal models, feature models, and mapping based on the given parameters. By setting the parameters, the generator produces the random models, which satisfy the requested characteristics.

To control the number and kinds of inconsistencies in the family requirements models, we first randomly produced family goal models and then generated feature models from goal models by transforming product line variability in the goal models into variability relations in feature models and behavioral variability into mandatory relations. This way, we can generate random inconsistencies in the family requirements model, by considering inconsistency patterns illustrated in Table 8.1 and check if the proposed algorithms can find these inconsistencies.

The initial feature models, which are derived from the goal models are expanded with additional features. We based the expansion strategy for feature models on the FORM method case-study [74] as between one and five implementation features were added to feature model for each capability feature. Thus,we add between 0 to 5 features to each atomic feature in the feature models to reflect the technical features that are added

Table 8.2: Specification of eight Models in the first experiment with 100 goals and tasks. The other sixteen models have similar specification except number of goals and tasks.

Model Size	PV [%]	Pot. Inc. [%]	Str. Inc. [%]
100	25	25	25
100	25	25	50
100	25	50	25
100	25	50	50
100	50	25	50
100	50	50	25
100	50	50	50

to realize the conceptual feature. The numbers of potential and strong inconsistencies varies in the family requirements model.

We investigated questions RQ1 – RQ4 in two different settings. In the first setting, the distribution of intentional relations is fixed and the other potential factors of influence to answer questions RQ1 – RQ3 vary. In the second setting, we analyze the influence of different distributions (question RQ4). Furthermore, in both settings, we check whether the algorithms work correctly.

Experimental setting 1. In this setting, the distribution of intentional relations in the family goal model is fixed. The setting compares different sizes of goal models (RQ1), different distributions of product line variability (RQ2), and different number of inconsistencies (RQ3). The distributions of intentional relations are considered as follows: 50% AND-decomposition , 25% OR-decomposition, and 25% XOR-decomposition. For AND-decompositions, 50% are considered optional and the other 50% are mandatory intentional elements. This distribution is selected in order to avoid any influence of different distributions of intentional relations on the running time for this experiment. We investigate such different distributions of intentional relations in the experimental setting 2. There are contribution links from 20% of the goals and tasks (as source intentional elements) to the soft-goals (as destination intentional elements).

With the fixed distribution of intentional relations, we generated goal models with 100, 200, and 300 goals and tasks and 10 soft-goals. The number of goals and tasks was selected based on investigations of the size of practical goal models in the literature [93, 109]. From, existing intentional relations in the generated goal models, either 25% or

50% of the relations are set as product line variability by annotating them with *VP*. For changing the behavioral variability and product line variability percentages, we do not change the structure of family goal model. We should note that in our approach generated OR and XOR relations in family goal model are fixed (5% OR-decomposition, and 25% XOR-decomposition.). The change in the product line variability is only done by annotating the generated family goal models, hence, we prevent any change in the structure of family goal models. With respect to inconsistencies, we considered 25% and 50% of the total number of possible inconsistencies. Hence, we generated 24 family requirements models covering the wide range of goal models sizes, product line variability and inconsistency distributions. Table 8.2 illustrates eight generated models with 100 number of goals and tasks. The other 16 models have similar characteristics except different number of goals and tasks (i.e., 200 and 300). We use 10 different models for each kind of generated family requirements model to reduce the impact of the randomness of generated models on the running time.

Experimental setting 2 This setting concentrates on question RQ4, which aims at investigating the effect of the distribution of intentional relations on the running time of the inconsistency detection. During the experiments, we generated goal models with 300 goals and tasks, 10 soft-goals. We consider 50% distribution of product line variability in all the models. For every intentional relation AND, OR, and XOR, three distributions 25%, 50%, and 75% are devised leading to nine different kinds of models, as outlined in the columns 1-3 in Table 8.3. This enables us to cover wide ranges of structural variability in family goal models. The average number of inconsistencies in all models were 50% of the total number of possible potential and strong inconsistencies.

System information and implementation details. The knowledge base creation is implemented with OWL-API. For reasoning, we used the Pellet reasoner (Pellet reasoner site: http://clarkparsia.com/pellet/). Our test system is a Notebook with an Intel Core 2 Duo T7300 CPU (2.0 GHz, 800 MHZ FSB, 4 MB L2 cache and 2GB DDR2 RAM). We used 256MB RAM for the Java VM of the Eclipse environment.

Given generated family requirements model, our tool creates an knowledge base as described in Section 8.5. The DL expressivity is $\mathcal{ALC}$. After reasoning on the family requirements model, the tool produces a list of potential and strong inconsistent mappings.

Table 8.3: Specification of Models in the second experiment. After deciding on one of intentional relation distribution, the other relations have the equal distributions in the models

AND [%]	OR [%]	XOR [%]	Avg. Time [msec]
25	37	38	5510
50	25	25	5680
75	13	12	5120
37	25	38	5710
25	50	25	5130
12	75	13	4950
37	38	25	5620
25	25	50	5840
13	12	75	5970

Experimental Results and Analysis

The results of the analyses of performance and influence factors are illustrated in Figures 8.6 and 8.7. The percentage of potential inconsistencies (third column in Table 8.2) and strong inconsistencies (fourth column in Table 8.2) refer to the corresponding percentage of the total number of possible potential and strong inconsistencies that can occur in each individual model of each type. Because the models are randomly generated, the number of possible inconsistencies (potential and strong inconsistencies) is different in each model.

According to the results shown in Figures 8.6 and 8.7, we reflect and discuss each of our question.

- RQ1: As answer to this research question, we investigate the main effect of the model size on the running time and the results indicates that the mean value of the execution time was significantly higher in the goal model with 200 elements ($M = 2372.91, SD = 524.38$) than in the goal model with 100 elements ($M = 796.66, SD = 51.69$) and also the mean value of the execution time was significantly higher in the goal model with 300 elements ($M = 4608.65, SD = 1209.49$) than in the goal model with 200 elements ($M = 2372.91, SD = 524.38$). According to the result, we can conclude that the size of models has impact on the running time. However, as we can observe the execution time of our algorithm for the largest

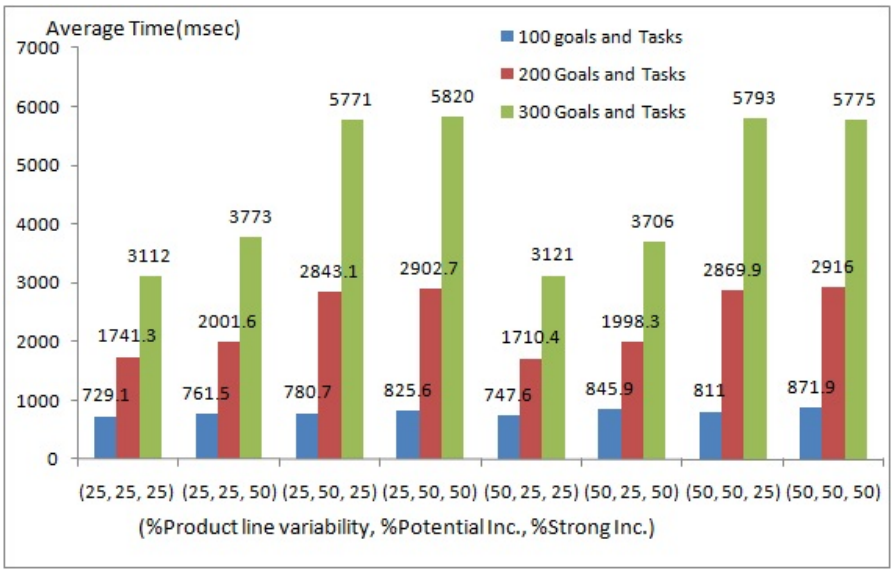


Figure 8.6: The average running time for models in the first experiment: the Y-axis shows the running time and the X-axis shows different distribution of product line variability, potential inconsistencies and strong inconsistencies.

experimented model in less than 6 seconds on a comptuer with rather modest hardware capabilities. As we mentioned in experimental setting, a common size of goal models is less than 300 goals and tasks.

- RQ2: Based of our analysis, the main effect of product line variability indicates that the mean value of the execution time was not significantly greater for the 50% product line variability ($M = 2597.02, SD = 1734.38$) than for the 25% product line variability ($M = 2588.42, SD = 1755.15$). Therefore, different distributions of product line and software variabilities do not impact the running time of the verification algorithm. As we mentioned in Section 8.7.2, in the generated models, the software and product line variability distributions are computed based on the total number of generated XOR and OR relations. For example, the 75% product line variability is computed by randomly annotating 75% of OR and XOR relations with VP. Interestingly, the results show that having different distributions of product line variability in the family goal models does not affect the running time, significantly.

- RQ3: The increase of the inconsistencies increase the execution time of the algorithm. This can be observed for each model size. The results shows the mean value of the execution time was significantly higher in the 50% strong inconsistency ($M = 2683.07, SD = 1750.81$) than in the 25% strong inconsistency ($M = 2502.42, SD = 1734.03$). Also, the mean value of the execution time was significantly higher in the 50% potential inconsistency ($M = 3164.91, SD = 2046.67$) than in the 25% potential inconsistency ($M = 2020.58, SD = 1113.35$). According to the results, the influence of the number of potential inconsistencies is larger than the influence of the strong inconsistencies. This result can be attributed to the two main reasons: First, as we expected, each strong inconsistency is also a potential inconsistency and the representation of potential inconsistency is a more complex concept expression due to the negation in the verification concept. Second, according to Table 8.1, the total number of possible potential inconsistencies is larger than the total number of possible strong inconsistencies, consequently, 25% and 50% potential inconsistencies is higher than 25% and 50% strong inconsistencies.

- RQ4: According to the results for this research question, distribution change of intentional relations has different impact over running time, while the higher number of OR relations decreases running time, the higher number of XOR relations increases the running time. Interestingly, if the half of the relations are AND relations, the running time is the higher. Also, OR relations require less verification time, while XOR relations are the most expensive one. This result is expected based on the logical representation in our knowledge base, as XOR descriptions are quite complex concept expressions using negation. However, our purpose is to check whether our approach can cope with different kinds of distributions and this seems to be confirmed by the experimental evaluation.

As already stated, the aim of this evaluation was to demonstrate that the modeling and reasoning approach is tractable in practical software product lines. The size of 300 goals is a realistic size of the family requirements model to capture real software product lines. The evaluation shows that even if up to 50% of all possible inconsistencies occur the execution time is feasible (6 seconds) for the largest experimented model.

Finally, the evaluation confirmed the that the verification algorithms are correct as all inconsistencies are recognized by the reasoner in the DL knowledge base.

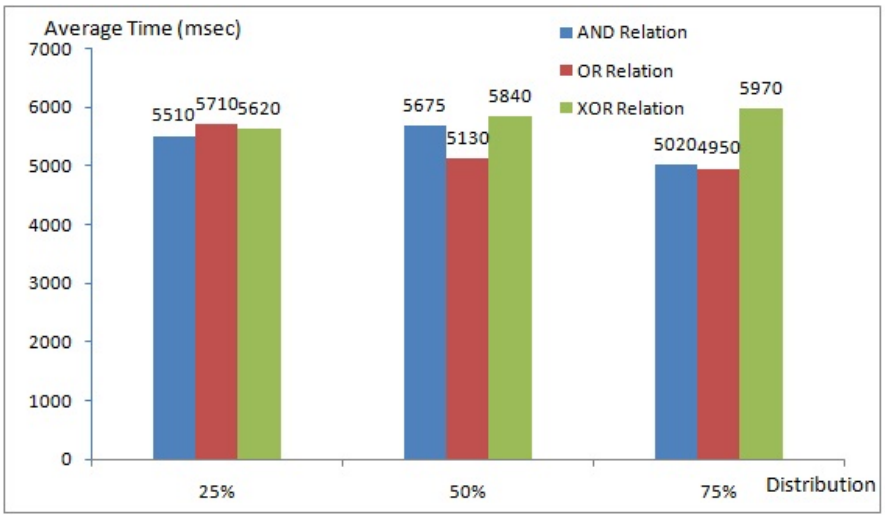


Figure 8.7: The average running time for models in the second experiment: The Y-axis shows the running time and the X-axis shows different distribution of intentional relations.

Threats to Validity

Threats to internal validity refer to the confounding variables that might have impact on the running time of the verification algorithm and were neglected during the experiment. We prevented these kinds of threats by designing two different sets of experiments and by controlling confounding variables. For example, in the first experiment setup, we considered a fixed distribution of intentional relations and focused on the impact of other independent variables (i.e., size of model, product line variability, and number of potential and strong inconsistencies).

External validity investigates if the results of the experiments are generalized. With respect to the size of family requirements models, our investigation over existing publications in SPL and requirements engineering communities confirmed that the sizes of generated models are aligned with the size of existing models in the literature.

In order to trace the precentages of inconsistency injected in family requirements model, we generated feature models from family goal models and added inconsistencies based on Table 8.1. This method leads to structural similarity between family goal models and feature models. Hence, one may concern that the samples are not proper

representatives of existing models in the real world. Family goal model shows the intentions of the stakeholders and their refinements into sub-goals and tasks. On other hand, feature models shows hierarchical representations the features of a product line which implements the tasks in family goal model. Therefore, there are similarities between features model structure and family goal model structure. Additionally, many researchers in software product line requirements engineering domain [191, 86, 149, 25] proposed techniques to transfer goal models into feature models which supports our claim for similarity between family goal models and feature models.

With respect to distribution of intentional relations and variability relations, we reduced this threat by covering wide variety of distributions including 25, 50, and 75 percent. Additionally, we automatically generated 100 samples for each combination to include the structure diversity in generated models.

Another threat to external validity of our approach is that the execution time is influenced by the proposed modeling and formalization approach of family requirements model. Hence, for various representation, we may have slight different execution times for each distribution of independent variables. Thus, our results can not be generalized to other approaches such as SAT solvers, which could be used as alternative for implementation of the proposed verification approach. However, the experimental results shows that employing description logic can cope with different kinds of distributions.

8.8 Related Work

In this section we disscuss the related works and compare them with our work using a set of criteria found in the litrature.

8.8.1 Goal Modeling Techniques In Software Product Line Engineering

Goal models, seen as a complementing technique for other requirements engineering approaches, are started to be applied in the context of SPLE. Yu et al. [190, 191] developed an approach to (i) generating design models from goal models and (ii) using the goal models for configuration of the generated design models. The authors of [190] introduce a set of annotations for generating feature models from goal models, which they extend in [191] to design models such as statecharts and components.

Similarly, Lapouchian et al. [86] introduce an approach to automatically generating

business process models and their configuration with extended goal models. They introduce variation points for specifying design time and run-time variabilities. In order to generate business processes automatically from goal models, they propose ordering annotations between goals and tasks that are involved in a AND-decomposition relations.

Silva et al. [149] employ aspectual i* to support variability in software product lines. Mandatory features are considered internal elements of i*, while optional, alternative and OR features are treated as aspectual elements in their approach. Composition rules are defined for all variabilities in the models. Introducing aspects in i* models and describing composition rules for all variabilities increased the complexity of models which leads to scalability problems. Finally, this work is based on the assumption that each optinal and alternative feature is mapped into one aspect.

Goal and feature model relations are further investigated by Antonio et al. [10]. They propose an approach to deriving feature models from i* goal models. A feature is defined as a relevant characteristics of the system, while a system characteristic allows for achieving a certain goal. They integrated cardinalities into i* models to represent (optional and alternative cases) variability in the models. Furthermore, a means-end relation may be transferred into OR-relation or alternative relation according to the cardinality of means involved in the relation. In addition to the extension on i* models, they introduced a set of heuristics to produce a feature model from i* models.

Borba and Silva [25] extend the i* language to represent variability and commonality. They introduce mandatory means-end, optional means-end, cardinality means-end, alternative means-end, and means-end group cardinality. They also provide a set of heuristic rules along with high level processes for transforming feature models to goal models. This approach performs product configuration using the notion of soft-goals and contribution links between tasks and soft-goals.

Santos et al. [142] introduce an aspect-oriented requirements modeling language called PL-AOVGraph to represent variability in software product lines. The language is an extention of AOV-graph goal models that include cardinalityMin, cardinalityMax, groupFeature, cardinalityGroupMin, cardinalityGroupMax, and isFeature to support variability. In their approach, they provide a bidirectional transformation between feature models and PL-AOVGraphs, where produce one of models when the other model is available.

Mussbacher et al. [110] apply the AoURN framework in the software product line

domain. They use GRL for modeling stakeholders' objectives and propose a URN profile for defining feature models. In their approach, they create goal models and feature models independently. Next, for atomic features in a feature model, their behavior and structure is generated. Afterwards, features are mapped into goals and their impacts are identified over the goals using contribution links.

Liaskos et al. [90] exploit the intuitiveness of goal models for configuration of customizable software. In their approach alternative system configurations are matched with alternative ways of satisfying high level goals. Hence, by perfroming reasoning over a goal model, they can customize a software system.

Jureta et al. [72] proposed an abstract requirements modeling language called Techne, which introduces optional, mandatory, and preference notions into the standard goal modeling languages. Techne describes inference, conflict, preference, is-mandatory, and is-optional relations and models the requirements and their relations in terms of graphs called r-net. In order to identify the preferred solutions, r-nets are encoded into propositional formula and logical reasoning is employed.

Ernst et al. [46] extended the Techne language with some operations to perform paraconsistent reasoning over models represented in Techne models. The approach, called KOMBINE, describes a set of consistency criteria over requirements problems to find a solution in presence of conflict. It applies the PARACONSIST-MIN-GOAL-ACHIEVEMENT and PARACONSIST-GET-CANDIATAE-SOLUTION operations which identify the mandatory requirements and non-mandatory requirements, respectively.

Finally, Ali et al. [5] enriched Tropos goal models with contextual information and proposed contextual goal models. They defined contextual information in terms of propositional formula and considered alternative behaviors of the system to be variants derived based on contextual information. Their technique identifies inconsistencies of the contexts specified as preconditions for an alternative behavior and inconsistencies in the changes on the context caused by execution of tasks [6]. They applied SAT techniques to discover inconsistencies.

8.8.2 Criteria based Comparison With Related Works

In order to compare the goal oriented approaches in product lines, we developed a set of criteria by adapting the criteria proposed in Mussbacher et al. [110] and Varela et al. [175]. *Feature model* and *goal model* criteria investigates if the approach provides

Table 8.4: Comparisons of Goal-oriented Software Product Line Approaches (+ means criterion is met with the approach and – means criterion is not met with the approach)

Approaches	Criteria							
	Goal Model	Feature Model	B/P variability	Conflict Identification	Impact analysis	Stakeholders' trade-off	Configuration	G/F validation
Yu et al. [190]	+	+	–	+	–	+	+	–
Lapouchian et al. [86]	+	+	+	+	–	+	+	–
Silva et al. [149]	+	+	–	+	–	–	+	–
Antonio et al. [10]	+	+	–	+	–	–	+	–
Borba and Silva [25]	+	+	–	+	–	–	+	–
Santos et al. [142]	+	+	–	+	–	+	+	–
Mussbacher et al. [110]	+	+	–	+	+	+	+	–
Liaskos et al. [90]	+	–	–	+	–	+	+	–
Techne [72]	+	–	+	+	–	+	+	+
KOMBINE [46]	+	–	–	+	–	+	+	+
Contextual Goal Models [5]	+	–	–	+	–	+	+	+
Our approach	+	+	+	+	+	+	+	+

these two models. The *behavioral and Product line variability* criterion refers to discrimination of software and product line variability in goal models. The *Impact analysis, Conflict Identification* and *Stakeholders trade-offs* criteria explore if the impacts of features over intentions of stakeholders are addressed, if the conflict in the soft-goals are identified, and if reasoning over stakeholders' needs is preformed, respectively. Finally, the *Configuration* and *Goal Model and Feature Model Validation* criteria investigate if the approach configures a feature model according to the requirements of stakeholders and checks consistencies of variability relations between goal models and feature models.

Table 8.4 illustrates differences between our approach and the existing goal modeling approaches in the context of software product lines.

Our approach mainly differs from other approaches in providing a comprehensive validation approach that takes potential changes in feature models which can appear during the design of SPLs and validates whether variability relations in the feature model is aligned with the intentional relations in the family goal model. Additionally, we discriminate between product line variability and behavioral variability in goal models, which

is neglected in most of the works except in [86, 72]. However, it is crucial to consider the difference between these two types of variability when using goal models in the software product line domain. For conflict analysis we rely on the GRL techniques and through mapping we can identify the impact of the features over tasks and consequently by using propagation algorithms in the GRL we can see the impacts of the features on the higher level objectives.

Other related work consider the detection of inconsistency in feature configurations, but without goal models. Czarnecki et al. [36] introduce an approach based on SAT solvers that detects violations of OCL-based well-formedness constraints of design models in feature configurations. Thaker et al. [162] extend even further this research direction and detect the absence of references in feature models to undefined classes, methods, and variables. Janota et al. [70] and van der Storm [171] use propositional logic to validate the correctness of mappings between feature and component models. All of these approaches are used to detect inconsistencies between design models and feature models where inconsistencies are the result of the change of one model. In this work, we go a step further and validate inconsistencies between feature and goal models.

8.9 Conclusions and Future Work

As demonstrated in the paper, our contribution advances the state-of-the-art in goal-oriented requirements engineering for SPLs with a formal and correct validation approach for family requirements models. The proposed validation assures that intentional variability of an SPL, captured in goal models, does not violate constraints of technical variability captured in feature models. The proposed validation approach compares the influence of intentional relations given by a goal model with feature relations from the feature model.

Both goal models and feature models successfully have been used in software engineering research and practice to capture stakeholders' needs and intentions [5, 7, 54] and managing product line variability [74, 34, 100]. Their combinations advance requirements engineering and configuration in the software product line context. Applying goals in software product line not only facilitates identifying features in domain engineering life-cycle, but also ease the selections of features based on stakeholders' intentions and needs in application engineering life-cycle. Several related works as well as our framework and tooling support demonstrate the practical applicability of the goal oriented

requirements engineering in the context of software product line engineering. Therefore, we believe that the results of our work will be useful for the development of professional tools.

In our future work, we will also describe the validation procedure for the further steps of product line configuration. First, we can validate if existing relations in the reference architecture models, in our research reference process models, are aligned with the relation defined in the family goal models. Second, we are going to develop an approach to ensure validity of customized process models with respect to relations in reference process models and feature models. Finally, we aim at extending the validation technique to cover other properties such as safe composition (i.e., for all application goal models, we have at least one feature model configuration and vice versa).

Chapter 9

Customized Process Model Validation

This chapter consists of "Development and validation of customized process models" paper which is published in *the journal of System and Software*.[1] The paper introduces a technique for the customization of process models from a configured process model. Behavioral and configuration inconsistency patterns which may occur during the customization are identified and description logic is used to ensure validity of final customized process model. A case study and a set of experiments were developed to evaluate the approach.

Author's role - I was the main contributor in developing the customization and validation framework. I identified the customization operations and proposed a two-step customization approach. Also, I identified the behavioral and configuration inconsistency patterns. The case study, design of experiments, implementing the experiments, analysis of the results, and developing parts of description logic algorithms has been done by the author. I wrote all the sections of the paper except sections 6 and 7.

[1] The paper reprinted (with minor adjustment to formatting), with permission from Elsevier publisher.

Asadi, M., Mobabbati, B., Groner, G., Gasevic, D. (2014) Validation of Customization of Reference Process Models. Journal of Systems and Software (JSS), Vol. 96, No. 10, pp. 73-92.

9.1 Abstract

Goal models and business process models are complementary artifacts for capturing the requirements and their execution flow in software engineering. In this case, goal models serve as input for designing business process models. This requires mappings between both types of models in order to describe which user goals are implemented by which activities in a business process. Due to the large number of possible relationships among goals in the goal model and possible control flows of activities, developers struggle with the challenge of maintaining consistent configurations of both models and their mappings. Managing these mappings manually is error-prone. In our work, we propose an automated solution that relies on Description Logics and automated reasoners for validating mappings that describe the realization of goals by activities in business process models. The results are the identification of two inconsistency patterns – orchestration inconsistency and choreography inconsistency – and the development of the corresponding algorithms for detecting these inconsistencies.

9.2 Introduction

Nowadays many business domains such as automotive engineering [108] and health care [89] need to support flexible processes to cope with business process variability [84]. Hence, the development and customization of reference process models is a common practice in business process management, adopted by many enterprises [84, 65]. Reference process models capture proven practices and recurrent business operations in a specific domain [166, 132, 131]. Reference process models are designed in a generic way and intended to be configured and customized to meet requirements of individual stakeholders. However, reference process models often lack an explicit representation of configuration options and alternatives [166, 132]. The explicit representation of the variability is one of the key aspects of variability and configuration management. Research agrees that the variability should be modeled and represented in a uniform way in the development life-cycle [170, 39, 150, 26]. Therefore, many researchers have been

working on the concept of *configurable process models* in the context of process-oriented software development [55, 137] and model compliance [158].

A configurable reference process model represents a family of similar process models and describes multiple variants of a process model in an integrated way [83, 56][2].Several approaches and methods have been proposed to model variability in configurable reference process models [55, 132, 137], in which process modeling languages like EPC, YAWL, and BPMN are extended with some notations for supporting variability.

9.2.1 Open Challenges

Developing and customizing a reference process models encompasses several challenges. Four important challenges are:

Challenge 1 (Variability Complexity): In analogy with product line variability [181], the variability in reference process models [3] refers to how process variants of a reference process model may differ from each other [12, 84]. Hence, the variability represents differences among the customized process models, derived from a reference process model, in terms of their process elements. Different variability patterns [12] (e.g., alternative, OR, optional) may occurs between different process elements. For example, alternative variability pattern between a set of activities represent that each customized process model has only one of the alternative activities. Also, control-flow variability shows differences of control-flow patterns, defined over the same set of activities, in customized process models. Therefore, many complex variabilities may exist in the reference process model and efficient techniques are required to capture and model those variabilities.

Challenge 2 (Modeling complexity): Although reference process models deal with variability in process models, variability still is not considered as a first class object. Variant-specific information is maintained in the constructs of a reference process language [65] which overloads process models with selection options (i.e., variation points and variants) and dependencies (i.e., integrity constraints) in a single model in addition

[2]Some researchers use process lines [161] in analogy with product lines where product lines principles are applied for modeling processes. Both process lines and configurable process models concentrate on modeling variability in process models. Similar to template based approach [34] for implementing product lines, configurable process models can be considered a technique for implementing process lines.

[3]In the remaining of the paper, when we use the term *reference process model*, we refer to *configurable reference process model*.

to process logic (i.e. execution semantic). Incorporating all these concerns in a single model, especially for large size process models, increases the complexity of reference process models for the development and customization of these models.

Challenge 3 (Delta Requirements): Many of reference process modeling approaches derive a customized process for a reference process model by removing process elements from the reference process model via operations like hide and block [55]. According to the literature [128], it is unlikely that a reference process model covers all the requirements of the target application. Thus, the process engineers have to customize the configured process model to meet the remaining requirements of the stakeholders. These remaining requirements are known as *delta requirements*. The customization approaches should not only provide mechanisms for deriving a customized process model from a reference process model, but also should provide the developers with mechanisms for making changes to the customized process model to fit it to target application requirements.

Challenge 4 (Customization Validation): While customizing a reference process model for an individual customer, the generated process should adhere to the regulations defined in a reference process model. Several configuration and behavioral regulations might be enforced when a reference process model is constructed. However, derivation of a customized process model from a reference process model is error-prone process especially considering changes need to be done for delta requirements. Hence, the customization approach should guarantee the correctness and compliance of every customized process model with respect to the specified configuration and behavioral constraints and inform process engineers of possible inconsistencies.

9.2.2 Contribution and Outline

In this paper, we aim at tackling down the the aforementioned challenges by proposing a novel customization and validation approach, which utilizes the principles of software product lines and description logic formalism.

Software product lines community extensively investigated the problem of model configuration [125]. Product line approaches considered variability a first class object and developed dedicated languages such as feature models [73] and Orthogonal Variability Model (OVM) [125] for modeling variability and selection dependencies (i.e., integrity constraints). Among the existing variability modeling languages, feature models are

widely employed to represent variability in software artifacts including requirements, design, and implementation artifacts [105, 59, 21]. Some researchers [161] employed feature models for a unified representation of the variability of reference process models. This helps to manage complexity of process models (i.e., Challenge 2) and to employ the existing configuration approaches in SPLE [35, 106, 155] to configure a reference process model. We adopt a similar strategy for modeling process variability and have the following contributions:

- A feature-oriented customization approach which 1) addresses more complex variability types (Challenge 1) in reference process models by applying feature models to represent activity variability and implicit variability constructs in process model to represent control-flow variabilities; 2) defines a guideline to derive a final customized process model from a reference process model based on the stakeholders' requirements for a target application.

The proposed approach applies restriction strategy by configuring feature models and consequently deriving configured process model and applies extension strategy to further customize the process model based on delta requirements. Our approach improves reference process customization [55, 137, 65, 132] in several aspects: i) it handles more comprehensive variability types (Challenge 1), ii) it manages the complexity of modeling variability along with process logic by employing feature models (Challenge 2), and iii) it deals with delta requirements by providing an extension strategy (Challenge 3).

In addition to customization guidelines, our approach provides a description logic based technique that ensures the correctness of customized process models with respect to behavioral and configuration constraints formulated in reference process models. In this regard, our contributions are:

- A set of inconsistency patterns which may happen between customized process models and reference process models. These patterns are divided into two groups: *behavioral inconsistency* and *configuration inconsistency* patterns. The former group represents possible inconsistencies between a customized process model and a reference process model and the latter one reflects inconsistencies between a customized process model and a feature model.

- A formal framework that enables the validation of customizations of reference process models. The formalism uses Description Logic and reasoning technologies

to automatically detect the inconsistency patterns between customized process model and reference process model and feature models.

The framework is implemented and a set of experiments is designed to evaluate the running time of the algorithms and investigate the behavior of the algorithms for different sizes and percentages of variability and inconsistency distributions. The experiments results revealed that the running time of the validation framework is tractable in practical reference process models with various model sizes and inconsistency distributions. Also, our customization technique, unlike many correctness approaches [168, 64, 140, 134] which only concentrate on behavioral correctness of the customized process model, ensure both behavioral and configuration (i.e. variability and integrity constraints) constraints.

This paper is organized as follows. Section 9.3 describes a running example, which is used throughout the paper. Types of variabilities in reference process models and the proposed notations for their representations are discussed in Section 9.4. Section 9.5 explains the proposed customization process which is followed by a description of possible inconsistencies that can happen between a reference process model and a customized process model in Section 9.6. Section 9.7 and Section 9.8 present the representation and validation formalism. The evaluation is given in Section 9.9, followed by the related work (Section 9.10) and the conclusion (Section 9.11).

9.3 Running Example

The running example is a part of an industrial case study in the health care domain. In collaboration with our industrial partner, we developed a reference process model for their information system[4]. The system is the main working information system product of a company in the domain of optometry. The system provides the necessary information management services to support optometrists in their daily activities such as conducting different types of eye exams and managing patient information. The information system consists of components for managing information of customers (patients), different types of eye exams (pre-exam, history exam, and main room exam), prescribing, and insurance. Figure 9.1 shows a part of a reference process model developed for the exam component of the system.

[4]Due to privacy concerns, we filtered the company and system name.

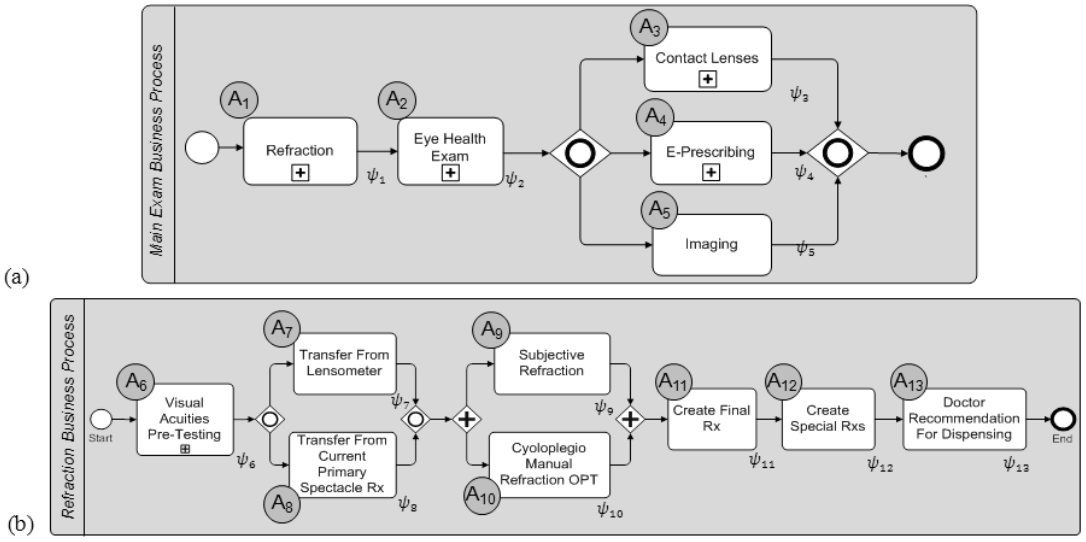


Figure 9.1: a) A part of the reference process model: Main exam. b) A part of the reference process model Refraction process model

The system gives the opportunity to the doctors to customize their software service according to the special needs of their practices.

9.4 Variability in Reference Process Models

In order to support the development of different customized process models from a reference process model, variability must be included in the reference process models. This section provides a detailed discussion of two different kinds of variability, which may happen in a reference process model: *activity variability* and *workflow variability*. These types of variabilities were identified by investigating the existing research on the notion of variability in process modeling [94, 4, 12]. In this paper, we limit our scope only to control flow in process models and do not consider data flow and roles in process models. Thus, we do not investigate possible data variability and role variability in the process models. The information about data flow and roles can be obtained from [135].

9.4.1 Activity Variability

A reference process model is a template for the family and contains a set of activities in the domain. There are a number of core activities in the reference process model that need to be included in every customized process, while the other activities may vary in different customized process models. Hence, these optional activities may be included/excluded for different customized process models according to preferences and requirements of a target application. Additionally, there might be complex variability relations between optional activities encapsulated in a sub-process of a reference process model such as exactly one of optional activities in a sub-process must be included in the customized process model. We refer to these types of variability in reference business processes as activity variability. For example, in the Main Exam sub-process of Figure 9.1.(a), among E-prescribing, Contact Lenses, and Imaging one or more of these activities may be included in the customized process models. Additionally, several types of selection dependencies can exist between different activities of a reference process model. The common classes of dependencies, which have been identified in the literature [4, 140] are:

- *Include*: A selection of one activity in the reference process model requires the selection of one or more activities (one-to-many include relationship). For example, in the Refraction sub-process, shown in Fig. 9.1.(b), the selection of the activity Create Final Rx requires the selection of the activity Transfer From Current Primary Spectacle RX. Also the selection of each of Insert Patient Insurance Information, Update Patient Insurance Information, and Delete Patient Insurance Information activities requires the selection the other two activities (e.g. selection of Insert Patient Insurance Information requires the selection of Update Patient Insurance Information and Delete Patient Insurance Information).

- *Exclude*: A selection of one activity in the reference process model excludes the selection of one or more activities (one-to-many exclude relationship). For instance, selecting the activity Visual Acuities testing excludes a need for the activity Visual Acuities pre-testing in a customized process.

Representing activity variability and configuration dependencies between activities in reference process models increases complexity of reference process models [65]. Hence, techniques and languages are required to separate the activity variability and selection

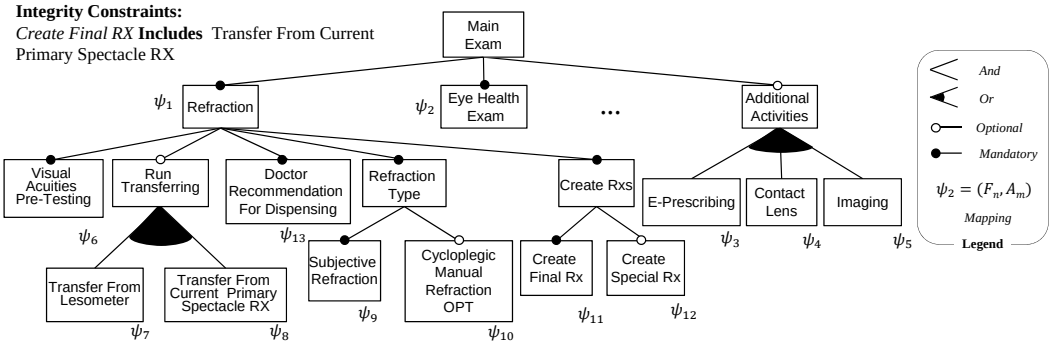


Figure 9.2: A part of the feature model for the Eye-care case-study

dependencies from the reference process model and facilitate producing customized process models. Inspired by software product line engineering, we adopt feature models, whereby features refer to activities in order to represent activity variability in the reference process model. Using feature models for representing activity variability, we can map a feature to an atomic activity or a composite activity.

A feature model is represented visually as a tree-like structure in which nodes represent features and edges express the relationships between them. Fig.9.2 shows a simplified part of the feature model of the investigated information system developed in collaboration with our industrial partner. In our case-study, the most recurring mapping situation was when a feature is mapped to a sub-process, containing several activities. For example, visual acuities pre-testing was a sub-process, which is mapped to the leaf feature visual-acuities pre-testing.

The relations in feature models are divided into hierarchical relations and cross-tree relations, also called integrity constraints. In our context, we employ hierarchical relations for representing different types of variability that may happen in the reference process models. These relations comprise of:

- *Mandatory Relation*: The selection of a parent feature (e.g., Main exam) requires the selection of its *mandatory* child features (e.g., Eye Health Exam).

- *Optional Relation*: If its parent feature (e.g., Main Exam) is selected, the *optional* feature (e.g., Additional Activity) may or may not be selected.

- *OR-Group*: If a parent feature (e.g., Run Transferring) is selected at least one of the children (e.g., Transfer From Lesometer and Transfer From Current Primary Spectacle RX) must be selected.
- *Alternative-Group*: If a parent is selected exactly one of the childern must be selected.

On the other hand, dependencies such as *include* and *exclude* are supported through integrity constraints in the feature models. For example, in Fig. 9.2, the selection of Create Final Rx requires the selection of Transfer From Current Primary Spectacle RX. Feature models [73, 34] only support simple *include* and *exclude* integrity constraints where one feature include or exclude a list of features. Batory et al. [21] discuss the need for more complex integrity constraints for include and exclude relations where a feature may include or exclude a boolean combination of features. For example, a feature F_1 *includes* F_2 *or* (F_4 *and* (F_6 *or* F_7)). They proposed the use of propositional formula for representing these complex integrity constraints. We adopt the similar approach with respect to integrity constraints to cover the complex include and exclude dependencies in process models [4, 140].

We formally define a feature model with respect to feature relations as follow:

Definition 9.1 (Feature Model) *A feature model* $FM = (\mathcal{F}, \mathcal{F}_M, \mathcal{F}_O, \mathcal{F}_{IOR}, \mathcal{F}_{XOR}, \mathcal{F}_{incl}, \mathcal{F}_{excl})$ *consists of features* $\mathcal{F}$ *and feature relations in terms of parent-child and integrity constraints.* $\mathcal{F}_M \subseteq \mathcal{F} \times \mathcal{P}(\mathcal{F})$ *and* $\mathcal{F}_O \subseteq \mathcal{F} \times \mathcal{P}(\mathcal{F})$ *are sets of parent features and the sets of all their mandatory and optional child features, respectively.* $\mathcal{F}_{IOR}$ *and* $\mathcal{F}_{XOR} \subseteq \mathcal{F} \times \mathcal{P}(\mathcal{F})$ *are sets of parent and their corresponding child features, respectively.* $\mathcal{F}_{incl}$ *and* $\mathcal{F}_{excl} \subseteq \mathcal{F} \times \Phi(\mathcal{F})$ *are sets of* includes *and* excludes *relationships (integrity constraints) where* $\Phi(\mathcal{F})$ *is a propositional formula over features.*

For example, based on the above definition, $\mathcal{F}_{IOR}$ for the feature model in figure 9.2 is: $\mathcal{F}_{IOR}$ = { (Run Transferring, { Transfer From Lesometer, Transfer From Current Primary Spectacle RX }), (Additional Activities, {E-prescribing, Contact Lens, Imaging}) }.

9.4.2 Workflow Variability

Reference process models formalize comprehensive business logic for orchestration and choreography of activities, which is described using control flow patterns. In addition

to diversity in business activities, customized process model may vary in their business process logic (i.e., workflow patterns). For example, in the refraction sub-process of eye-care domain, many applications require to run at least one of Transfer From Lensometer or Transfer From Current Primary Spectacle Rx (OR-workflow relation), but few of applications need to execute exactly one of them in their pre-testing sub-process (XOR Workflow relation) (See Fig 9.1.(a)). Hence, in the reference process model, a typical workflow pattern (OR-relation in our running example) is created in business process models and is later customized into special cases according to the requirements of stakeholders.

Further investigation of sequential patterns reveals another type of variability in reference process models, which is referred to as order variability [12]. This type of variability exists when a set of activities can be executed in a sequential order, but the order of their executions differ in various customized processes. For example, based on different contexts during an Examination process, pre-test exam is executed either before or after History exam.

To highlight the order variability type in reference process models, we divide sequence patterns into two sub-categories: *strict sequence* patterns and *weak sequence* patterns. These categories were adapted from behavioral profile for the context of reference process models [180].

Definition 9.2 (Strict Order Relation) *Let R be a reference process model defined by a structured process model $(\mathcal{V}, \mathcal{E}, \mathcal{F}, \mathcal{E}_A)$ with activities $\mathcal{A} \subseteq \mathcal{V}$. There is a strict order relation between two activities a_i and a_j, if for all customized process models derived from the reference process model R where a_i and a_j activities exist, activity a_i must be executed before activity a_j.*

The strict order relation illustrates commonality (with respect to the execution order) of customized process models that belong to a process family. The strict order relation in the reference process model is modeled using the sequence workflow pattern.

Definition 9.3 (Weak Order Relation) *Let R be a reference process model defined by a structured process model $(\mathcal{V}, \mathcal{E}, \mathcal{F}, \mathcal{E}_A)$ with activities $\mathcal{A} \subseteq \mathcal{V}$. There is a weak order relation between two activities a_i and a_j, for customized process models which both activities a_i and a_j exist, if in different customized process models, activity a_i is executed before or after or concurrently with activity a_j.*

The weak order relation in reference process models represents diversity in customized process models (i.e., order variability). Similar to strict order, we use the sequence pattern for showing the most common execution order of activities in weak order relation, but the activities are annotated to show needs for the ordering change of activities. During the customization process, we change the order of their execution based on stakeholders' requirements.

9.5 Reference Process Model Configuration and Customization

In the customization process, we assume that a reference business process model and a feature model were developed and the mappings between them were established (Fig. 9.3). The reference business processes describe the orchestration and choreography of activities in a family and the feature model encapsulates configuration knowledge and enables the derivation of different customized processes of a reference process model [106, 105, 131, 152]. Mappings link features of the feature model (Fig. 9.2) to the corresponding activities of the reference process model (Fig. 9.1). We consider many-to-many mappings by naming convention $\boldsymbol{\phi_i}(F_i, \mathcal{A}_i)$ where $\mathcal{A}_i$ is a subset of activities in a process model. If an activity is mapped into more than one features,the activity present in the corresponding subsets $\mathcal{A}_i$ of all those features. We define a mapping model as follow:

Definition 9.4 (Mapping Model) *Let $R = (\mathcal{V}, \mathcal{E}, \mathcal{F}, \mathcal{E}_A)$ be a reference process model defined by a structured process model and $FM = (\mathcal{F}, \mathcal{F}_M, \mathcal{F}_O, \mathcal{F}_{IOR}, \mathcal{F}_{XOR}, \mathcal{F}_{incl}, \mathcal{F}_{excl})$ be a feature model. A mapping model between feature model and reference model is $\Phi \subseteq FM \times R = \{\Phi_i(F_i, \mathcal{A}_i)$: $F_i \in \mathcal{F}$ and $\mathcal{A}_i \subseteq \mathcal{A} \subseteq \mathcal{V}\}$.*

The detail of developing reference business process, feature models, and their mapping is out of scope of this paper. (Interested readers can see [105, 106] for further information.)

As shown in Fig. 9.3, a customized process model is derived through *configuration* and *customization* steps [132, 131, 35]. As shown in the figure, both steps utilize change operations in order to adjust the reference process model to the requirements of a target process. Weber et al. [178] identified a set of change operations, which can be employed on a process model. In order to customize a reference process model according to the target requirements, our approach supports the following change operations:

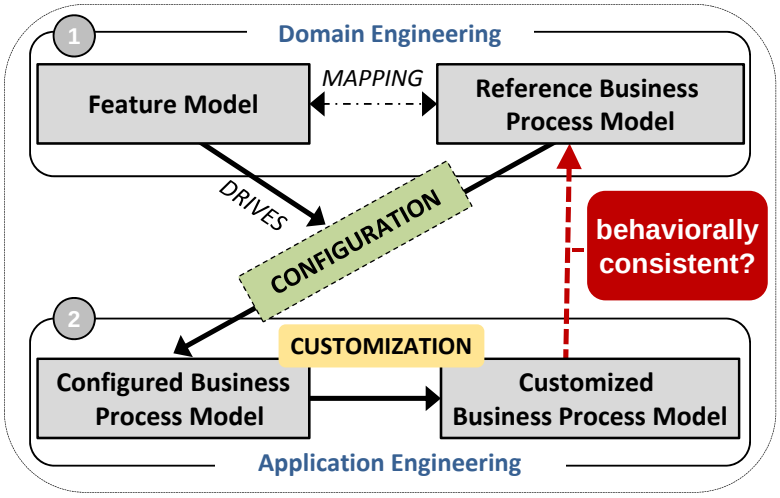


Figure 9.3: Customization in Application Engineering life-cycle

- INSERT: This operation adds a process element (i.e., an activity, gateway, or sub-process) into a process model.
- DELETE: This operation removes a process element (i.e., an activity, gateway, or sub-process) from a process model.
- MOVE: This operation changes the execution order of activities in a process model.
- MODIFY: This operation changes the execution patterns of a process model. It changes the workflow patterns or converts a pattern into another pattern.
- REPLACE: This operation replaces a process element (i.e., activity or sub-process) with another process element (i.e., activity or sub-process) in a process model.

According to [65] and [114], the above change operations are most commonly used for deriving a customized process model form a reference process model. These operations also provide the capabilities required to resolve different types of variability mentioned in Section 9.4. Hence, based on the rationale model in [114], variability points can be considered as decisions (i.e. rationale) which trigger different change operations in a process model. Having defined the change operations, in the reminder of this section, we define the proposed configuration and customization steps (Fig. 9.3).

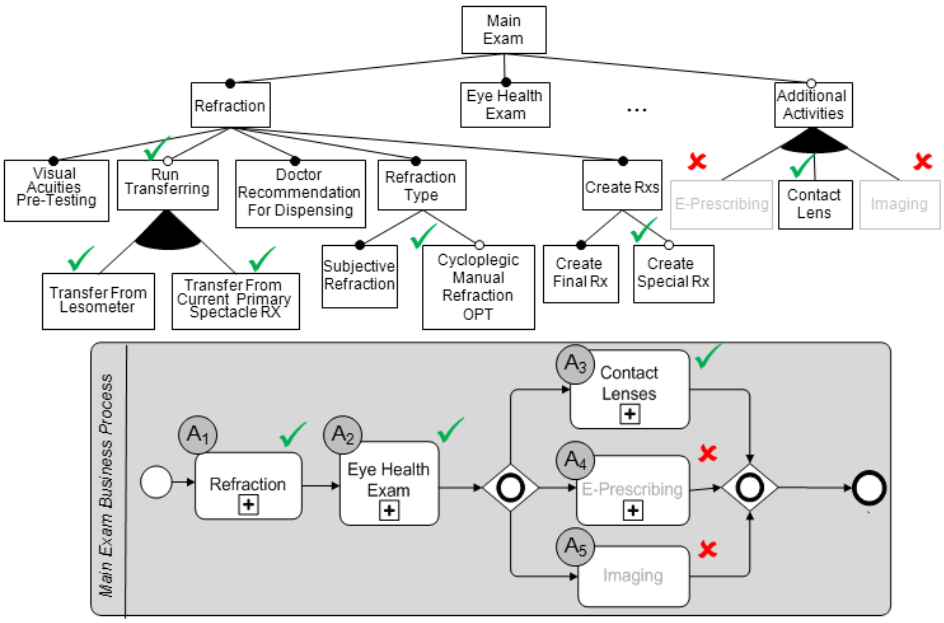


Figure 9.4: A configured feature model generated after configuration step and corresponding configured main exam process model.

Configuration step: *Configuration* is driven and performed by a stepwise specialization of the feature model. Specialization is known as a process where some configuration choices are eliminated in each stage of a specialization [35]. For instance, stakeholders select and deselect features of the feature model based on their preferences and business objectives. Figure 9.4 illustrates a configured feature model for the health care domain. The features E-prescribing and Imaging are deselected and all features related to Refraction and Eye Health Exam are selected. Several researchers in software product line engineering have developed algorithms for automated configuration of feature models according to stakeholders requirements [155, 35].

After configuring the feature model, due to the established mappings between features and activities of the reference process model, the DELETE operation is executed over activities mapped to the deselected features. As a result of executing the DELETE operation on reference process models, a configured process model is automatically generated. For example, in our case-study, if the feature `Imaging` is deselected then its corresponding activity is removed consequently from the reference process model (cf. Fig. 9.4).

Customization step: It is rare that the configured process model satisfies all the requirements of the target application (service consumer) [128]. Hence, the customization step further adapts and extends the configured process model according to the remaining requirements (i.e., so-called *delta requirements* [128]). The customization step includes adding, removing, and refining some activities and gateways in the configured process model. During the customization step, process engineers iteratively apply the above-mentioned change operations to the configured process model to derive a customized process model. For example, in Fig. 9.5, Transfer From Lensometer is replaced by two refined activities Transfer From Auto Lensometer and Transfer From Manual Lensometer by performing the REPLACE operation. The AND-gateway is changed into an XOR-gateway via the MODIFY operation and Populate Retinoscopy is added to the process model using the INSERT operation. Finally, the activity Create Final RX is removed from the process model by performing the DELETE operation.

9.6 Inconsistencies in the Customization Process

A customized process model must preserve the behavioral relations (business logic) specified by a reference process model and the variability (configuration) relations expressed by a feature model. We distinguish between the following relations:

- A feature F depends on other features $F_1, \ldots, F_m$. This is expressed by a variability relation VR_F of F.
- A workflow relation on activities $A_1, \ldots, A_n$ in the reference process model is denoted by WR.
- A workflow relation over activities $A'_1, \ldots, A'_k$ of the customized process model is denoted by WR'.

9.6.1 Behavioral Inconsistencies

In order to investigate the behavioral inconsistencies, we use *trace semantics* of business processes. The trace semantics describe a process in terms of all possible executions (traces). A trace is a sequence of activities. We define the concept of a set of allowable execution combination for a workflow pattern.

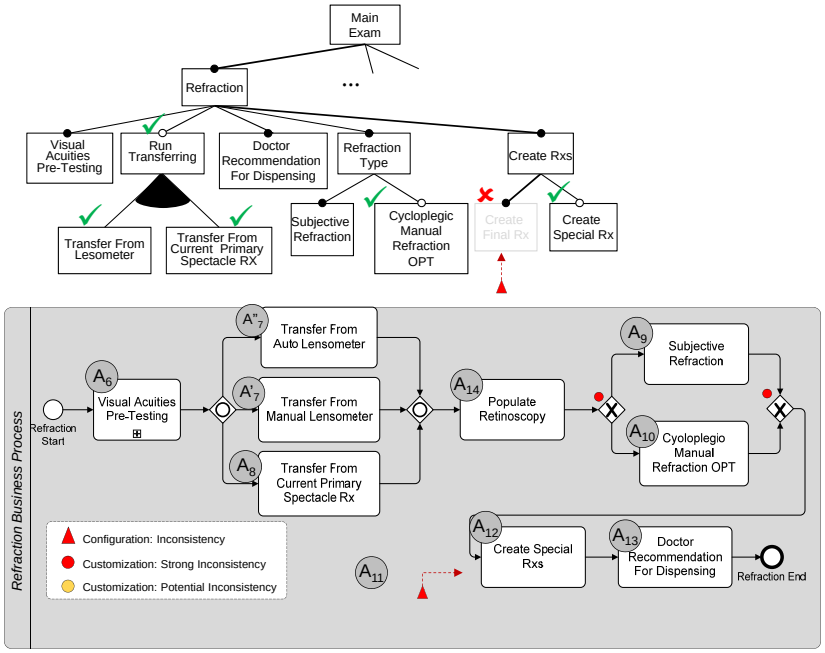


Figure 9.5: A customized process model.

Definition 9.5 (Allowable Execution Combination Set) *Let WR be a workflow relation defined over activities* $A_1, \ldots, A_n$. *The allowable execution combinations is the set of all execution combinations of activities* $A_1, \ldots, A_n$ *(Shown as* $\Gamma(WR)$*) that do not violate constraints defined by the workflow relation WR.*

For example, consider Figure 9.1, the allowable execution combinations of the AND workflow pattern are $\Gamma(AND) = \{\langle$ *Subjective Refraction, Cyoloplegio Manual Refraction OPT* $\rangle\}$ and allowable execution combinations for the OR workflow pattern are $\Gamma(OR) = \{\langle$ *Transfer From Lensometer* $\rangle, \langle$ *Transfer from Current Primary Spectacle RX* $\rangle, \langle$ *Transfer From Lensometer, Transfer from Current Primary Spectacle RX* $\rangle\}$ which show three possible execution situations might happen for the OR workflow.

In practice, the degree of inconsistency can be varied. According to our analysis based on industrial test cases, two types of behavioral inconsistency may occur in a customized process model with respect to a reference process model: 1) *strong inconsistency* and 2) *potential inconsistency*.

Definition 9.6 (Strong Inconsistency) *Assume there is a mapping between activities of workflows WR and WR′. A strong inconsistency between WR and WR′ occurs if there is no intersection between the allowable execution combination set of WR and the allowable execution combination set of WR′, i.e.,* $\Gamma(WR) \bigcap \Gamma(WR') = \emptyset$

Definition 9.7 (Potential Inconsistency) *Assume there is a mapping between activities of workflows WR and WR′. A potential inconsistency between WR and WR′ occurs if the allowable execution combination set of WR is a subset of the allowable execution combination set of WR′, but not equal to WR′, i.e.,* $\Gamma(WR) \subset \Gamma(WR')$ *and* $\Gamma(WR') \neq \Gamma(WR)$.

Execution combinations sets in Definitions 9.6 and 9.7 refer to the *trace semantics* of a process.

An example of a strong inconsistency is shown in Fig. 9.5. Consider activities Subjective Reflection and Cycloplegic Manual Refraction OPT in the customized process model Refraction Process (Fig. 9.5) and in the reference process model Refraction Process (Fig. 9.1.(b)). Intersection between the allowable execution combinations set of the activities in the XOR workflow pattern and corresponding activities in the AND workflow pattern in the reference process model is an empty set. In other words, the business logic (workflow pattern), which orchestrates activities in the reference process model, and the business logic (workflow pattern) in the customized process model are completely different. Therefore, they do not yield to the same execution traces.

Combinations of workflow relations WR and WR' are shown in Table 9.1. Strong and potential inconsistencies are marked by ($\lightning$) and ($\pm$), respectively. The comparison of workflow relations in Table 9.1 is based on the trace semantics, as previously introduced. If a set of workflow relations $WR'_1, WR'_2, \ldots, WR'_n$ in the customized process model are specializations of a workflow relation WR, then there is an inconsistency if at least one of the combinations $(WR'_1, WR), \ldots, (WR'_n, WR)$ is inconsistent.

9.6.2 Configuration Inconsistencies

A customized process model should also be valid with respect to the configuration relationships that are imposed by the feature model. The configuration relationships are formulated in the feature model in our approach. These relationships include hierarchical

Table 9.1: Correspondence between Workflow Patterns in Reference and Customized Business Process Models

	Workflow Patterns	Reference Process: AND-AND	AND-DISC	AND-XOR	OR-OR	OR-DISC	OR-XOR	XOR-XOR	Sequence
Customized Process	AND-AND	✓	✓	✓	✓	✓	✓	↯	✓
	AND-DISC	±	✓	✓	✓	✓	✓	↯	✓
	AND-XOR	±	±	✓	✓	±	✓	↯	✓
	OR-OR	±	±	±	✓	±	✓	±	±
	OR-DISC	±	±	±	✓	✓	✓	±	±
	OR-XOR	±	±	±	✓	±	✓	±	±
	XOR-XOR	↯	↯	↯	✓	±	✓	✓	±
	Sequence	✓	✓	✓	✓	✓	✓	↯	✓

Legend: No (✓), strong (↯) and potential (±) inconsistency

relations (i.e., mandatory, OR, alternative, and optional) and integrity constraints (include and exclude). The configuration step does not cause inconsistencies in the derived process model (i.e., configured process model), because we perform configuration of the feature model and then according to the mappings between features and activities, the DELETE operation is performed to automatically generate a configured process model. Moreover, several approaches and algorithms exist which ensure validity of a feature model configuration.

However, during the customization step when a process engineer applies change operations, configuration constraints might be violated. For example, in Fig. 9.5, activity Create Final RX (A_6) is removed in the customization step, while activity Create Special Rx (A_7) is still in the model. However, the corresponding feature of Create Final RX (A_6) is a mandatory feature in the feature model. Thus, the removal of this activity does not reflect the indented configuration behavior as it is given by the feature model. Therefore, it must be ensured that changes during the customization step do not lead to inconsistencies as defined in the feature model.

Schobbens et al. [145] and Gue et al. [61] defined well-formedness rules for feature models using Free Feature Diagrams (FFD). Among the well-formedness rules, we adopt

those rules which are relevant for the configuration of feature models: 1) there is a unique feature root that is a direct or indirect parent of all features in the feature model; and 2) all features, except the root feature, have one parent feature. For example, in our case-study, the root feature is Eye-care[5], which shows the core concept of the information system. Considering these two well-formedness rules, similar to Gue et al. [61], we define the following concepts in the feature models:

Definition 9.8 (Mandatory Path) *A mandatory path for a feature F is a path between the feature F and the root feature in a feature model in which each intermediate node (feature) is either a mandatory feature or the sole child in an Alternative- or Or-group.*

Definition 9.9 (Inclusion Chain) *An inclusion chain for feature F is a chain from start feature F where all features are connected by the include relations to each other.*

For example, a mandatory path for feature Create Final RX is(See figure 9.4): ManPath(Create Final RX) = {Create Final Rx, Create Rx, Refraction, Main Exam, Eye-care}. Inclusion chain is defined based on transitivity of *include* relation.

Mappings describe a concrete implementation of a feature by one or more activities. As the customization might add and remove activities, the selection and deselection of features in the feature model specialization changes correspondingly. In order to ensure both well-formedness rules for feature models and their specializations after the customization operations, we check configuration inconsistencies in the feature model specialization. We distinguish between *hierarchical inconsistency* (Definition 9.10) and *integrity inconsistency* (Definition 9.11). Both definitions rely on the mandatory path and inclusion chain that are given by integrity constraints, according to Definitions 9.8 and 9.9.

In these two definitions a *deselected feature* refer to a feature whose corresponding activity (the activity mapped to the feature) does not exist in the customized process model. A *selected feature* is a feature whose corresponding activity exist in the customized process model.

Definition 9.10 (Hierarchical Inconsistency) *A deselected feature F is hierarchically inconsistent if either F: a) is mandatory and there is a mandatory path from the*

[5] Eye-care is the root of the main feature model which Main Exam is its mandatory child.

feature F to the root feature; or b) is an OR- or alternative feature and there is a mandatory path from the feature F to the root feature and all feature with the same direct parent with the feature F are deselected.

Definition 9.11 (Integrity Inconsistency) *There is an integrity inconsistency in a selected feature F if: a) there is a deselected feature in the inclusion chain of the feature F; or b) there is a selected feature which is excluded by feature F.*

For example, in Fig. 9.5, activity Create Final RX is removed from the customized process model and consequently the corresponding feature is labeled as deselected after the customization step. Since the feature is mandatory and there is a mandatory path ManPath(Create Final RX) = {Create Final Rx, Create Rx, Refraction, Main Exam, Eyecare} from this feature, there is a hierarchical inconsistency induced by this feature.

9.7 Customization Knowledge Base

In order to automate detection of inconsistencies in the customization and configuration steps, we need a formal representation formalism that captures variability relations, workflow relations and mappings between activities and features. Furthermore, we need (automatic) means to automatically compare these relationships and pinpoint the source of an inconsistency. Following this line of argumentation, we propose a formal knowledge representation that offers reasoning (or inference) services to facilitate model validation.

There are several logical knowledge representation languages like propositional logic, first-order logic and temporal logic. Other formalisms might use traversal algorithms to compare process graphs, while further approaches compare execution traces of process models. In this paper, we use Description Logics (DL) [16], a decidable subset of first-order logic, as a common representation formalism. DL offer an expressive representation language, a well-defined semantics and practically efficient reasoning services.

9.7.1 Variability Relations

The knowledge base of the feature model Σ_{VR} represents each feature F by a concept. Variability relations of a feature F are expressed by a further concept VR_F. Algorithm 6 describes the transformation of variability relations.

Features $F_1 \ldots F_n$ that are mandatory child features of a feature F are conjunctively related to each other, i.e., either all of them are selected or none of them. The

Algorithm 6 Knowledge Base Σ_{VR} of Variability Relations

1: Input: Feature Model $(\mathcal{F}, \mathcal{F}_M, \mathcal{F}_{IOR}, \mathcal{F}_{XOR}, \mathcal{F}_{incl}, \mathcal{F}_{excl})$
2: **for all** $(F, \{F_1, \ldots, F_n\}) \in \mathcal{F}_M$ **do**
3: $VR_{F_j} := VR_F \sqcap \bigsqcap_{i=1,\ldots,n} \exists requires.F_i \ (j = 1, \ldots, n)$
4: **end for**
5: **for all** $(F, IOR, \{F_1, \ldots, F_n\}) \in \mathcal{F}_{IOR}$ **do**
6: $VR_{F_j} := VR_F \sqcap \bigsqcup_{i=1,\ldots,n} \exists requires.F_i \ (j = 1, \ldots, n)$
7: **end for**
8: **for all** $(F, XOR, \{F_1, \ldots, F_n\}) \in \mathcal{F}_{XOR}$ **do**
9: $VR_{F_j} := VR_F \sqcap \bigotimes_{F' \in \{F_1,\ldots,F_n\}} \exists requires.F' \ (j = 1, \ldots, n)$
10: **end for**
11: **for all** $(F, \Phi(\mathcal{F})) \in \mathcal{F}_{incl}$ **do**
12: $VR_F := VR_F \sqcap \Phi(\mathcal{F})$
13: **end for**
14: **for all** $(F, \Phi(\mathcal{F})) \in \mathcal{F}_{excl}$ **do**
15: $VR_F := VR_F \sqcap \neg\exists\Phi(\mathcal{F})$
16: **end for**

corresponding DL axiom is built in lines 2 – 4, where a concept intersection refers to a conjunction. Features that are related to each other by an inclusive disjunctive relation are represented by a concept union (lines 5 – 7). Exclusive disjunctive dependencies among features are transformed into the DL knowledge base as described in lines 8 – 10.[6] Includes and exclude constraints are transformed in lines 11 – 13 and 14 – 16, respectively [7].

Axioms 9.1–9.3 illustrate variability relations of some features from the example in Fig. 9.2.(a). Axiom 9.1 defines features Refraction and EyeHealthExam as conjunctively related features, since both are mandatory child features of feature MainExam. The variability relations for feature EyeHealthExam is built likewise. Optional features are neglected. The inclusive or-grouping of the features TransferFromLensometer and TransferFromCurrentPrimarySpectacleRX is depicted by Axiom 9.2. The same axiom is built for the feature TransferFromCurrentPrimarySpectacleRX. Finally, Axiom 9.3 describes an integrity constraint, i.e., feature CreateFinalRX includes feature TransferFromCurrentPrimarySpectacleRX.

[6] Please note that $\otimes$ is not a standard operator in DL. To improve readability, we use $\bigotimes_{F' \in \{F_1,\ldots,F_n\}} \exists requires.F'$ as an abbreviation for $\bigsqcup_{F' \in \{F_1,\ldots,F_n\}} \exists requires.F' \sqcap \neg(\bigsqcup_{F'',F''' \in \{F_1,\ldots,F_n\}} (\exists requires.F'' \sqcap \exists requires.F'''))$.

[7] Since $\Phi(\mathcal{F})$ is a propositional formula, it can easily be transfered into Description Logic statements.

$$VR_{Refraction} \equiv \exists\, requires.Refraction \sqcap \exists\, requires.EyeHealthExam \quad (9.1)$$

$$VR_{TranferFromLensometer} \equiv \exists\, requires.TransferFromLensometer \sqcup \exists\, requires.TransferFromCurrentPrimarySpectacleRX \quad (9.2)$$

$$VR_{TransferFromLensometer} \equiv \exists\, requires.TransferFromCurrentPrimarySpectacleRX \quad (9.3)$$

9.7.2 Workflow Relations

The reference process model, as well as the customized process model cover workflow relations among their activities. In both cases, we consider a process model as a structured model which is given by sequential ordering of activities and different types of gateways to describe activity branching.

Sequences restrict predecessors and successors of activities. These relations are transformed into concept WR_A of activity A, as depicted in lines 2–6 of algorithm 2. All dependencies among activities are denoted by the role *requires*.

Afterwards, relations within fragments $\mathcal{F}$ are considered. Each branch $B \in \mathcal{B}$ of a fragment $F \in \mathcal{F}$ is a set of activities. In lines 8–10, the algorithm restricts concept definitions WR_{A_i}, where A_i refers to activities in parallel branches. We use an intersection between activity sets of sibling branches, indicating the conjunctive relationship between sibling activities in parallel branches.

The representation of activities in branches of fragments that start with an AND gateway and end with a discriminator (lines 11 – 13), or an XOR gateway (lines 14 – 16). The transformation of the remaining fragments starting with IOR gateways (lines 17 – 19 and 20 – 22), and XOR gateways (lines 23 – 25) is straightforward.

Note, in each fragment, all activities of the same branch are not related to other activities of this branch; thus, we only represent these activities of a branch as a *set* in terms of a concept union in DL. Furthermore, due to the nesting of activities ($A \in \mathcal{A}$) in multiple fragments, the corresponding workflow relations (WR_A) are conjunctively extended.

Algorithm 7 Knowledge Base Σ_{WR} of Workflow Relations

Input: Structured process model $PM = (\mathcal{V}, \mathcal{E}, \mathcal{F}, \mathcal{E}_A)$
for all $E \in \mathcal{E}_A$ **do**
  if $\langle A_1, A_2 \rangle = E$ **then**
    $WR_{A_1} := WR_{A_1} \sqcap \exists requires.A_2$ and $WR_{A_2} := WR_{A_2} \sqcap \exists requires.A_1$
  end if
end for
for all $F \in \mathcal{F}$ **do**
  if $F = (and, and, \mathcal{B})$ **then**
    $WR_{A_i} := WR_{A_i} \sqcap \bigsqcap_{B_j \in \mathcal{B}} \exists\ requires.(\bigsqcup_{A_i \in B_J} A_i)$
  end if
  if $F = (and, disc, \mathcal{B})$ **then**
    $WR_{A_i} := WR_{A_i} \sqcap \bigsqcap_{B_j \in \mathcal{B}} \exists requires.(\bigsqcup_{A_i \in B_J} A_i) \sqcup \bigsqcup_{B_j \in \mathcal{B}} \exists requires.(\bigsqcup_{A_i \in B_J} A_i)$
  end if
  if $F = (and, xor, \mathcal{B})$ **then**
    $WR_{A_i} := WR_{A_i} \sqcap \bigsqcap_{B_j \in \mathcal{B}} \exists\ requires.(\bigsqcup_{A_i \in B_J} A_i) \sqcup \bigotimes_{B_j \in \mathcal{B}} \exists\ requires.(\bigsqcup_{A_i \in B_J} A_i)$
  end if
  if $F = (ior, ior, \mathcal{B}) \vee F = (ior, disc, \mathcal{B})$ **then**
    $WR_{A_i} := WR_{A_i} \sqcap \bigsqcup_{B_j \in \mathcal{B}} \exists\ requires.(\bigsqcap_{A_i \in B_J} A_i)$
  end if
  if $F = (ior, xor, \mathcal{B})$ **then**
    $WR_{A_i} := WR_{A_i} \sqcap \bigsqcup_{B_j \in \mathcal{B}} \exists\ requires.(\bigsqcup_{A_i \in B_J} A_i) \sqcup \bigotimes_{B_j \in \mathcal{B}} \exists\ requires.(\bigsqcup_{A_i \in B_J} A_i)$
  end if
  if $F = (xor, xor, \mathcal{B})$ **then**
    $WR_{A_i} := WR_{A_i} \sqcap \bigotimes_{B_j \in \mathcal{B}} \exists\ requires.(\bigsqcup_{A_i \in B_J} A_i)$
  end if
end for

Axioms 9.4 and 9.5 exemplify the branching parts of the workflow relations of activities SubjectiveRefraction and TransferFromLensometer. Axiom 9.4 expresses the parallel branching of activities SubjectiveRefraction and CyoloplegioManualRefractionOPT. Likewise, the inclusive branching is represented by a disjunctive relations (concept union in DL) in Axiom 9.5.

$$WR_{SubjectiveRefraction} := WR_{SubjectiveRefraction} \sqcap (\exists\ requires.SubjectiveRefraction \sqcap \exists\ requires.CyoloplegioManualRefractionOPT) \quad (9.4)$$

$$WR_{TransferFromLensometer} := WR_{TransferFromLensometer} \sqcap (\exists\ requires.TransferFromLensometer \sqcup \exists\ requires.TransferFromCurrentPrimarySpectacleRX) \quad (9.5)$$

9.8 Inconsistency Detection

From the customized process model, we expect that it does neither violate workflow relations, which are imposed by the reference process model nor violate variability relations of the feature model. Thus, we have to check whether there are behavioral or configuration inconsistencies.

9.8.1 Variability and Workflow Relationships

For each activity A' in the customized process model that has an ancestry activity A in the original process model, the workflow relations WR' of A' must be consistent with those relations WR of A.

Each activity A' in the customized process model might be restricted by three kinds of relations: (i) Workflow relations $WR'_{A'}$ restrict A' in the customized process model. (ii) If A' has an ancestry activity A in the reference process model (represented by the correspondence $c(A, A')$, workflow relations of A' has to be consistent with those of A (i.e., WR_A). (iii) If the ancestry activity A is mapped to a feature F (represented by a mapping $\psi(F, A)$), the variability relations VR_F has to be consistent wrt. $WR'_{A'}$.

According to Table 9.1, we distinguish between strong inconsistency, potential inconsistency and no inconsistency. Our aim is to detect these kinds of inconsistencies in the knowledge base. Based on the representation of the relations in Sect. 9.7, the final knowledge base combines the relations from each model and encodes the comparison between relations of Table 9.1. Relations of both models are represented by the same role *requires* in Description Logics in order to enable the comparison of relations.

The configuration step follows restrictions of the feature model such that variability relations VR are necessarily satisfied. However, the customization adds, removes and changes activities of the configured process model in order to meet delta requirements (cf. Sect. 9.5). Since activities were mapped to features in the feature model, the addition and removal of activities leads to the selection and deselection of features. Consequently, variability relations VR may not be satisfied after a customization process. We expect that even after the customization the corresponding mapped features do not violate well-formedness of the feature model. In order to guarantee this, we have to check for any activity of the configured process model that is mapped to a feature whether there is a hierarchical or an integrity inconsistency of this feature.

9.8.2 Knowledge Base for Relationship Comparison

The variability relations of the feature model are captured in the knowledge Σ_{VR} and the workflow relations of the reference process model WR and customized process model WR' are described in the knowledge bases Σ_{WR} and $\Sigma_{WR'}$. The knowledge bases are extended as described in Definition 9.12.

Definition 9.12 (Knowledge Base for Comparison) *The final knowledge base* $\Sigma := \Sigma_{VR} \cup \Sigma_{WR} \cup \Sigma_{WR'}$ *contains variability relationships VR of the feature model, workflow relations WR of the reference business process model and workflow relations* WR' *of the customized process model.*
Behavioral Knowledge: *For each correspondence* $c(A, A')$*, we add the axioms:*
(1) $Valid^{\checkmark}_{A,A'} \equiv \neg WR_{A'} \sqcup WR_A$
(2) $Valid^{\pm}_{A,A'} \equiv WR_{A'} \sqcap WR_A$
(3) $A \equiv A'$
Configuration Knowledge: *For each mapping* $\psi(F, A)$ *of a feature to an activity:*
(4) if there is an activity A' *with* $c(A, A')$ *then add the axiom* $F \equiv \top$
(5) else add the axiom: $F \equiv \bot$.

Axioms (1) – (3) allow for the comparison of workflow relations WR and WR' of an activity A and its corresponding activity A'. Correspondences are represented by the equality of the concepts A and A' (Axiom 3). We add concepts $Valid^{\checkmark}_{A,A'}$ and $Valid^{\pm}_{A,A'}$ to detect strong and potential inconsistencies. Those features that are mapped to an activity (represented by $\psi(F, A)$) are either marked as selected ($F \equiv \top$) if they are still in the customized process model or marked as deselected $F \equiv \bot$.

Based on this final knowledge base Σ, we classify concepts by standard reasoning services in order to detect behavioral and configuration inconsistencies as follows:

1. **Behavioral Inconsistencies** are indicated by concepts $Valid^{\checkmark}_{A,A'}$ and $Valid^{\pm}_{A,A'}$. We detect *strong* and *potential inconsistency* as follows: (i) If $Valid^{\checkmark}_{A,A'}$ is classified as equal to the universal concept $\top$, we can guarantee that there is no inconsistency of the workflow relations over activity A in the reference process model and activity A' in the customized process model. (ii) In the other case, there is either a strong inconsistency or a potential inconsistency. These two cases are indicated by the classification of the concept $Valid^{\pm}_{A,A'}$. If $Valid^{\pm}_{A,A'}$ is classified as a subconcept

of the bottom concept $\bot$ (i.e., $Valid^{\pm}_{A,A'} \sqsubseteq \bot$), there is a strong inconsistency, otherwise (i.e., $Valid^{\pm}_{A,A'} \not\sqsubseteq \bot$) there is a potential inconsistency.

2. **Configuration Inconsistencies** By the variability relations VR of the feature model. Selected features are described as equal to the universal concept $\top$ and deselected features are represented by the unsatisfiable concept $\bot$. The variability relationships VR are covered in Σ_{VR}, describing hierarchical relations and integrity constraints. Thus, after classifying all concepts in the knowledge base, a configuration inconsistency exists if for a feature F its corresponding relationships VR_F is classified equal to the unsatisfiable concept $\bot$[8].

9.8.3 Correctness of the Inconsistency Detection

The correctness and completeness of the inconsistency detection algorithm is ensured by the sound and complete Description Logics reasoning. We can reduce this to the DL reasoning characteristics due to the encoding of workflow and variability relations as logical formulas, represented by complex concepts in DL. We briefly consider how the different relationships are compared with each other.

Behavioral Inconsistencies

The workflow relationships $\mathtt{WR}$ and $\mathtt{WR}'$ of the reference process model and the customized process model are represented as logical formulas. The comparison of these formulas, according to Table 9.1, is done by comparing formulas $\mathtt{WR}$ and $\mathtt{WR}'$ of both models. In the logical sense, the comparison to detect a strong inconsistency is done by checking whether the *conjunction* of both formulas $Valid^{\pm}_{A,A'}$ is satisfiable or not. The potential inconsistency is logically an *implication* from the formula that encodes the logical formula of workflow relation of the customized process to the formula of the reference process model.

Configuration Inconsistencies

The variability relation of a feature F is represented as a logical formula by the complex concept $\mathtt{VR}_F$. Thus, the dependency of feature F on other features is captured by the

[8]Please note that $\top$ and $\bot$ in Description Logics refer to *true* and *false* in the propositional logic sense.

concept $\texttt{VR}_F$. Our inconsistency detection is correct, if the concept $\texttt{VR}_F$ is *not* classified (by reasoning) as a subconcept of the unsatisfiable concept $\bot$ when F is mapped to an activity A ($\psi(F, A)$) and there is a corresponding activity A' in the customized process model (i.e., A' is not removed within the customization step). There are only three cases when $\texttt{VR}_F$ is classified as subconcept of $\bot$:

1. The feature F is not selected in the configuration step. However, then neither activity A nor A' appear in the configured and customized process model. Hence, this case can not occur in a correct configuration procedure.

2. A dependent feature of F (e.g., a child feature) is not selected in the configuration step, but F is selected. However, this would be an incorrect feature model configuration that is not allowed in the configuration phase. Thus, we can exclude this case.

3. Finally, F and all dependent features of F must be selected in the configuration step. Obviously, $\texttt{VR}_F$ can only be a subconcept of the unsatisfiable concept $\bot$ if at least one of its dependent features $\hat{F}$ are equal to $\bot$. This can only occur if $\hat{F}$ is mapped to activity $\hat{A}$ and the corresponding activity $\hat{A}'$ in the customized process model is removed in the customization step. Hence, this is a configuration inconsistency that our validation algorithm has correctly detected.

9.9 Evaluation and Proof of Concept

In this section, we firstly exemplify the presented validation algorithm throughout our case study. This is followed by a performance analysis to achieve comprehensive evaluation by means of simulation and the explanation of tooling support.

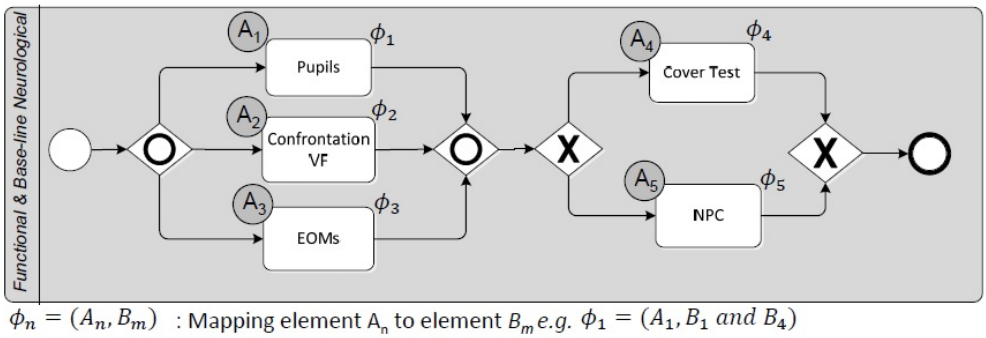
9.9.1 Case-Study Analysis

The core process models of the system were modeled using BPMN by domain experts and software developers of our industrial partner. The references process model consists of 27 sub-processes and 112 atomic activities.

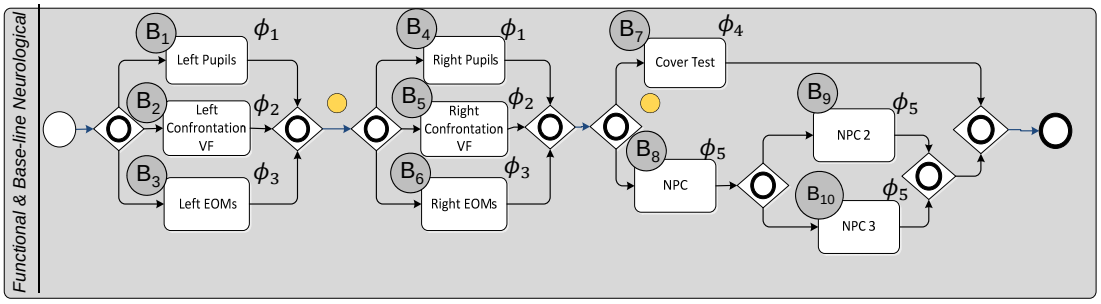
In the following, we use the process models of Figure 9.6 for further discussion. The process model Functional and Baseline Neurological (Figure 9.6(a)) describes several tests on patient eyes including puplis, Confrontation VF, cover test, EOMs, and NPC. According to

this process model, doctors first can preform puplis, Confrontation VF, EOMs tests and then continue with either cover test or NPC test.

During the customization, several customization operations are performed on these activities as follow. Activities puplis, Confrontation VF and EOMs are refined into two sets of activities for testing the left and right eyes. Furthermore, NPC is replaced with three activities. Finally, the XOR relation between cover test and NPC is changed into OR relation during the customization. Figure 9.6(b) shows the customized process model. In the customized process models Left puplis and Right puplis activities are mapped to puplis activity in the reference process model. Similarly other mappings are performed, which are shown in Figure 9.6(b).



(a) Reference Functional and Baseline Neurological process model



(b) Customized Functional and Baseline Neurological process model

Figure 9.6: The reference process model and its customized process model with inconsistencies

As an illustrative example, the customized Functional and Baseline Neurological process model contains two potential inconsistencies. The first inconsistency resulted from

changing an XOR workflow relation between cover test and NPC to an OR workflow relation. The second inconsistency is caused by the combination of workflow relations. During the customization, a process engineer creates an OR workflow relation between Left puplis, Left Confrontation VF, Left EOMs and an OR workflow relation between Right puplis, Right Confrontation VF, Right EOMs. Next, these activities are ordered sequentially (see Figure 9.6(b)). This design of the customized process model may lead to execution combinations such as $\langle$*Left puplis, Right Confrontation VF*$\rangle$ and $\langle$*Left puplis, Right EOMs*$\rangle$. Thus, in the run-time, there are process executions where from the first workflow relation only the Left puplis activity and from the second workflow relation only the Right Confrontation VF activity is executed. However, according to the mapping ($\Phi_1 = (Puplis,\ Left\ puplis\ and\ Right\ puplis)$), the executions of both Left puplis and Right puplis are required for the execution of Puplis in the reference process model.

Now we show how our algorithm can detect inconsistencies within the customized Functional and Baseline Neurological process model. For simplicity, we only explain axioms and validation procedure for the second inconsistency. Formulating the first OR workflow relation in the Functional and Baseline Neurological reference process model results in Axioms 9.6, 9.7 and 9.8.

$$\begin{aligned} WR_{Puplis} &:= WR_{Puplis}(\exists\ requires.Puplis \\ &\quad \sqcup\ \exists\ requires.ConfrontationVF \sqcup\ \exists\ requires.EOMs) && (9.6) \\ WR_{ConfrontationVF} &:= WR_{ConfrontationVF}(\exists\ requires.ConfrontationVF \\ &\quad \sqcup\ \exists\ requires.Puplis\ \sqcup\ \exists\ requires.EOMs) && (9.7) \\ WR_{EOMs} &:= WR_{EOMs} \sqcap (\exists\ requires.EOMs \\ &\quad \sqcup\ \exists\ requires.ConfrontationVF\ \sqcup\ \exists\ requires.Puplis) && (9.8) \end{aligned}$$

Axioms 9.9 to 9.16 show the process fragment in the customized Functional and Baseline Neurological process model, which corresponds to the first OR workflow relation in the reference process model. Axioms 9.9, 9.10, and 9.11 are transformations of OR workflow relation for Left puplis, Left Confrontation VF, Left EOMs and Axioms 9.12 to 9.14 are transformation of OR workflow relation for Right puplis, Right Confrontation VF, Right EOMs. Furthermore, in the customized process model, there is a sequential relation between activities of the two OR workflow relations. Axiom 9.17 reflects this for the successor relation of activity Left puplis. The axioms for the other activities are built in the same way.

$$
\begin{aligned}
WR_{LeftPuplis} &:= WR_{LeftPuplis} \sqcap \\
&(\exists\, requires.Leftpuplis \sqcup \exists\, requires.LeftConfrontationVF \\
&\sqcup \exists\, requires.LeftEOMs) && (9.9)\\
WR_{LeftConfrontationVF} &:= WR_{LeftConfrontationVF} \sqcap \\
&(\exists\, requires.LeftConfrontationVF \sqcup \exists\, requires.Leftpuplis \\
&\sqcup \exists\, requires.LeftEOMs) && (9.10)\\
WR_{LeftEOMs} &:= WR_{LeftEOMs} \sqcap \\
&(\exists\, requires.LeftEOMs \sqcup \exists\, requires.LeftConfrontationVF \\
&\sqcup \exists\, requires.Leftpuplis) && (9.11)\\
WR_{RightPuplis} &:= WR_{RightPuplis} \sqcap \\
&(\exists\, requires.Rightpuplis \sqcup \exists\, requires.RightConfrontationVF \\
&\sqcup \exists\, requires.RightEOMs) && (9.12)\\
WR_{RightConfrontationVF} &:= WR_{RightConfrontationVF} \sqcap \\
&(\exists\, requires.RightConfrontationVF \sqcup \exists\, requires.Rightpuplis \\
&\sqcup \exists\, requires.RightEOMs) && (9.13)\\
WR_{LeftEOMs} &:= WR_{LeftEOMs} \sqcap \\
&(\exists\, requires.RightEOMs \sqcup \exists\, requires.RightConfrontationVF \\
&\sqcup \exists\, requires.Rightpuplis) && (9.14)\\
WR_{NPC} &:= WR_{NPC} \sqcap (\exists\, requires.LeftNPC \sqcup \exists\, requires.CoverTest) && (9.15)\\
WR_{CoverTest} &:= WR_{CoverTest} \sqcap (\exists\, requires.CoverTest \sqcup \exists\, requires.NPC) && (9.16)\\
WR_{LeftPuplis} &:= WR_{LeftPuplis} \sqcap (\exists\, requires.(Rightpuplis \\
&\sqcup RightConfrontationVF \sqcup RightEOMs)) && (9.17)
\end{aligned}
$$

After generating axioms for the reference and customized Functional and Baseline Neurological process models, we also transfer mapping relations into Description Logic, as indicated in section 9.8. The Axioms 9.18,9.19 and 9.20 show the transformation of the mapping between Puplis and Left puplis and Right puplis. For other mapping relations similar axioms can be generated.

$$
\begin{aligned}
WR_{puplis} &\equiv WR_{Leftpuplis} \sqcap WR_{Rightpuplis} && (9.18)\\
Valid^{\checkmark}_{puplis,\langle Leftpuplis,Rightpuplis\rangle} &\equiv \neg(WR_{Leftpuplis} \sqcap WR_{Rightpuplis}) \sqcup WR_{puplis} && (9.19)\\
Valid^{\pm}_{puplis,\langle Leftpuplis,Rightpuplis\rangle} &\equiv WR_{Leftpuplis} \sqcap WR_{Rightpuplis} \sqcap WR_{puplis} && (9.20)
\end{aligned}
$$

After we generated the above axioms and we performed the reasoning, the results indicate $Valid^{\checkmark}_{puplis,\langle Leftpuplis,Rightpuplis\rangle}$ is not equal to the universal concept $\top$ (i.e.

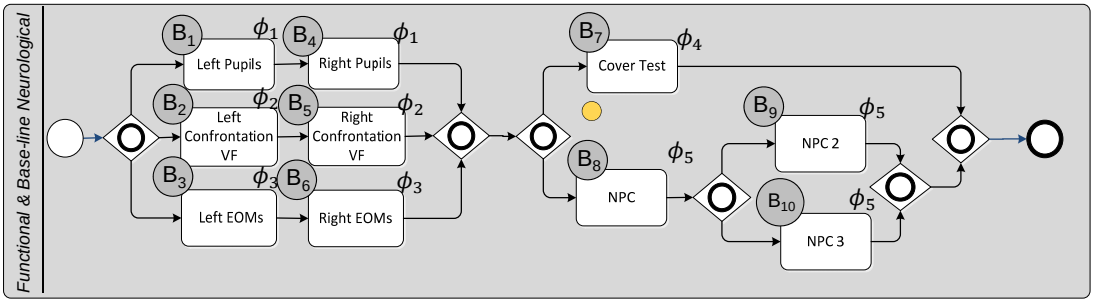


Figure 9.7: Functional and Baseline Neurological process model after resolving the detected inconsistencies.

$Valid^{\checkmark}_{puplis,\langle Leftpuplis,Rightpuplis\rangle} \neq \top$) and $Valid^{\pm}_{puplis,\langle Leftpuplis,Rightpuplis\rangle}$ is not equal to the bottom concept $\perp$ (i.e., $Valid^{\pm}_{puplis,\langle Leftpuplis,Rightpuplis\rangle} \sqsubseteq \perp$). This means that there is potential inconsistencies in the behavioral relations of Left puplis and Right puplis and the corresponding activity in the reference process model (i.e. puplis). Similarly, for other process elements such as Left EOMs, Right EOMs and Left Confrontation VF, Right Confrontation VF the potential inconsistency are identified.

After, discovering detecting potential inconsistencies in the customized process models, the domain expert would decide to change the model in order to resolve the first inconsistency and keep the second one based on the requirements. Figure 9.7 shows the result of customized process model.

9.9.2 Performance Evaluation

In this section, we evaluate the performance of the validation algorithm for different sizes of reference process models and different numbers of behavioral and configuration inconsistencies. The purpose of evaluation is to cover various validation situations and identify factors which have impact on the running time of the validation algorithm. Therefore, we derive a number of research questions and then preform a set of experiments to find out the answer for the research questions.

Research Questions

To study performance and scalability the inconsistency detection algorithm, we derive three research questions:

- RQ1: How does the execution time of the inconsistency detection algorithm scale-up as the size of reference process model increases?
- RQ2: Does the increase or decrease of behavioral inconsistencies have significant impact on the running time of the inconsistency detection algorithm?
- RQ3: Does the increase or decrease of configuration inconsistencies have significant influence on the running time of the inconsistency detection algorithm?

Experimental Setting

In order to evaluate our validation algorithm for several situations, we applied the simulation modeling technique by following guidelines similar to those proposed in [76]. We selected the simulation technique as it is commonly employed in the context of model validation and verification [59, 61, 140]. Moreover, using simulation techniques, we can gather the required data to statistically answer our research questions.

We generated family models (i.e., reference process models and feature models) and customized process models. For the generation of the family models, we firstly produced reference process models and then randomly produced feature models, expressing the configuration space with different distribution of configuration alternatives, i.e., optional/mandatory and alternative variations. The generator is developed based on the feature model generator of the FaMa Benchmarking System[9] and is extended in order to generate a corresponding reference process model.

After creating family models, we randomly generated several customized process models for every family model. To evaluate the validation algorithm, we adopt mutation testing technique, originally proposed in [41], which is a common fault-based technique and based on the assumption that a program is well tested if all simple faults are predicted and removed. Hence, configuration and behavioral inconsistencies are injected into the existing test cases to measure the effectiveness of our approach for verification of consistencies.

This size of process model has been the starting point for our analysis, which has been observed substantially through the realistic reference process model case exposed by our industrial partner and existing reference process models in the literature [101] such as the SAP reference process model. Hence, we generated the reference process models with

[9]FaMa Framework: http://www.isa.us.es/fama/

100, 300, and 500 activities. We considered equal distributions for the workflow patterns as we want to control the workflow distribution as a confounding variable. Next, feature models with the same size of reference process models were generated. With respect to the variability relations and integrity constraints, we distributed 50%, 25%, and 25% of features being in the AND, OR, and XOR, relations, respectively. Furthermore, 50% of features in the AND relation are optional features. These distribution were chosen based on the existing survey on feature models on realistic case-studies [62, 164]. Finally, the number of integrity constraints, which were produced for each feature model was 20% of the total number of features, based on the empirical analysis of existing feature models available in the SPLOT repository[10] as reported in [18].

For generating customized process models, we randomly configured feature models using the feature model plug-in (fmp)[11]. Next, the reference process model is configured and activities corresponding to the deselected features were removed from the reference process model to form a configured process model. In order to create configuration inconsistencies, we first computed the total number of mandatory paths and inclusion chains in the configured feature model; then, we generated models with 10%, 30%, and 50% configuration inconsistencies by removing a feature from mandatory path or inclusion chain. The distributions of inconsistencies were chosen not only to preserve diversity in the number of configuration inconsistencies, but also to have realistic view of the number of possible inconsistencies that might happen in real case-studies. We assume that it is rare to have a model with 80% configuration inconsistencies, as the customization step might only make small changes to cover delta requirements. The behavioral inconsistencies were created in a similar way, by computing the total number of workflow patterns (except the IOR-workflow pattern) and generating the models with 10%, 30%, and 50% of behavioral inconsistencies. The inconsistencies were injected in the model by changing the gateway in order to obtain inconsistencies according to Table 9.1. We neglected order variability during the experimentation to avoid the increasing the complexity of the experiments [84, 180].

Considering the reference process models size, configuration and behavioral inconsistencies distributions, we generated 27 cases including a reference process model, a feature model, and a customized process model with inconsistencies. For each case, 100

[10] http://splot-research.org/

[11] http://gp.uwaterloo.ca/node/18

Table 9.2: Charactrestics of Generated Models

Reference Models		Customized Models		
Reference Process [Activities]	Feature Model [Features]	Configured Model [Activities]	Configuration Inc.%	Behavioral Inc.%
100	100	75	[10, 30, 50]	[10, 30, 50]
300	300	225	[10, 30, 50]	[10, 30, 50]
500	500	375	[10, 30, 50]	[10, 30, 50]

samples were produced.

The knowledge base creation is implemented with OWL-API. For reasoning, we used the Pellet reasoner[12]. Our test system is a Notebook with an Intel Core 2 Duo T7300 CPU (2.0 GHz, 4 MB L2 cache and 2GB DDR2 RAM).

Experimental Results and Analysis

The results of the experiment are illustrated in Fig. 9.8, where the X-axis shows nine different combinations of configuration and behavioral inconsistencies and the Y-axis shows the running time of the algorithm for identifying inconsistencies. The distributions of inconsistencies are presented in Table 9.2. As shown in Fig. 9.8, the size of models have significant impact on the running time of the algorithm.

In order to answer the research questions, we hypothesized answers for the research questions and conducted statistical tests to accept or reject the answers. Since we have three groups for each independent variable (e.g. for model size variable we have 100, 300, and 500 values), we decided to apply the One-Way ANOVA test for comparing the average running time of these groups. We investigate the underlying assumption of the ANOVA test, i.e., a normal distribution of data. In the following statistical tests, M and SD denote mean value and standard deviation, respectively, and p is the statistical significant level indicating the p pre-determined threshold for significance of the results ($p < 0.05$). The tests were done using the JMP software[13].

- RQ1: To find out whether the size of reference process models has impact on

[12] http://clarkparsia.com/pellet/

[13] http://www.jmp.com/

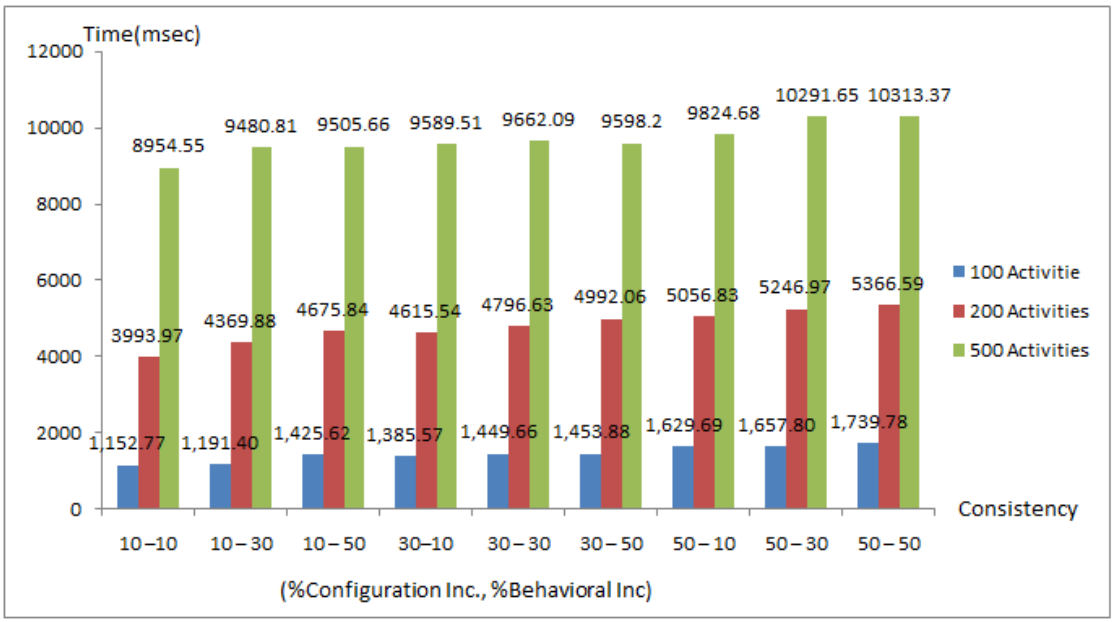


Figure 9.8: The results of the experiment

the execution time of the algorithm, we formulated the following hypothesis: *H1: the size of the reference process model significantly affects the execution time of validation.* In order to test the hypothesis, we investigate the effect of the reference model size on the running time and the results yielded an F ratio of $F(2, 899) = 100189.8$, $p < 0.0001$. This value indicates that there are significant differences between different sizes of reference process models. The further analysis using the Tukey-Karmar HSD test revealed that the mean value of the execution time was significantly higher in the reference process model with 300 process elements ($M = 4790.48, SD = 426.05$) than in the reference process model with 100 elements ($M = 1454.02, SD = 208.89$). Also, the mean value of the execution time was significantly higher in the reference process model with 500 elements ($M = 9691.17, SD = 487.31$) than in the reference process model with 300 elements ($M = 4790.48, SD = 426.05$). According to the results, we can conclude that the size of models has a significant impact on the running time. However, as we can observe the execution time of our algorithm for the largest experimented model in less than 12 seconds. As we mentioned in experiment description, a

common size of reference process models is around 500 activities.

- RQ2: As shown in Fig. 9.8, the change of behavioral inconsistencies does not significantly increase the execution time of the algorithm. In order to statistically verify this, we defined the following hypothesis: *H2: The increase of the behavioral inconsistencies significantly increases the execution time of the validation algorithm.* The results of tests for hypothesis H2 shows that the effect of source expertise yielded an F ratio of $F(2, 899) = 2.05$, $p < 0.13$, indicating that the mean value of the execution time was not significantly different between various distributions of behavioral inconsistencies. Thus, we can reject hypothesis H2. This result can be attributed to the fact that different distributions of inconsistencies do not affect the size of the customized process model (and thus, the size of our DL knowledge base) and only the types of workflow relations are changed.

- RQ3: In order to investigates the distribution impact of configuration inconsistencies over running time of the validation algorithm, we defined the following hypothesis: *H3: There is a significant difference in the execution time of the validation algorithm for diverse distribution of configuration inconsistencies.* The results of one-way ANOVA showed that there is a significant difference between diverse distributions of configuration inconsistencies $F(2, 899) = 9.88$, $p < 0.0001$. This confirms our hypothesis that the difference in configuration inconsistency distributions has impact on the execution time. Further analysis using Tukey-Kramar HSD revealed that increasing the distribution of configuration inconsistencies from 10% ($M = 4972.28, SD = 3334.15$) to 30% ($M = 5282.57, SD = 3369.94$) does not significantly decreases the running time ($p < 0.13$). However, the increase of configuration inconsistencies distribution from 30% ($M = 5282.57, SD = 3369.94$) to 50% ($M = 5680.82, SD = 3479.70$) increases the running time significantly ($p < 0.03$). This result may be because of the difference in the size of the customized process models and reference process models (i.e., their DL knowledge bases used in our validation algorithm), as the higher configuration inconsistencies means more deletion of process elements from a configured process model.

In these experiments, we mainly aimed at demonstrating that the modeling and reasoning approach is tractable in practical reference process models with various model sizes and inconsistency distributions. The size of 500 activities is a realistic size of the

reference process model to capture a real business process. The evaluation reveals that even if up to 50% of all possible inconsistencies occur the execution time is feasible (12 seconds) for the largest experimented model.

9.9.3 Tool Implementation

In order to support customization process and validation, we implemented a eclipse plug-in by utilizing the existing feature model plugin (*fmp*)[14], fmp2rsm[15], and TwoUse[16]. The tool supports both domain engineering and application engineering lifecycles.

During domain engineering lifecycle, using variability component of our tool (i.e., extended fmp plug-in), process developers can create features, their variability relations and dependencies. The tool supports modeling quality attributes for each feature [155]. The process modeling component embedded in TwoUse enables process developers to develop reference process models. While developing process models, mapping between the feature model and the process model is described using fmp2rsm tool.

During application engineering lifecycle, process developers use the tool for a manual configuration or an automatic configuration using either planning based configuration technique [155] or linear programming based configuration technique [106]. After, generating the configured process model, process developer do further customization of the process model using TwoUse tool and validation is performed by transferring the models into description logic by TwoUse tool.

We provided visualization and interaction techniques to facilitate manual configuration for process developers and preformed an empirical study on impacts of these techniques on improving performance and accuracy of process developers during configuration. The details of these techniques and their evaluation is outside the scope of this paper and can be found in [15].

9.9.4 Threats to Validity

An experimental evaluation is always subject to different threats that can affect the validity of the results. In this section, we investigate possible threats to validity of our results including external threats and internal threats.

[14] gp.uwaterloo.ca/fmp

[15] gp.uwaterloo.ca/fmp2rsm

[16] https://code.google.com/p/twouse/

Internal validity is mainly concerned with threats regarding the confounding variables (i.e., variables which are not considered as independent variables and might have impact on the dependent variable). An internally invalid experiment, results may not be necessarily inferred from a causal relationship between independent and dependent variables. We prevented these kinds of threats by controlling confounding variables. When we were investigating the impact of one independent variable (e.g., size of reference process models) on the running time, we controlled impacts of the other two variables (i.e., distributions of the configuration and behavioral inconsistencies), by only grouping the collected data based on the variable under-investigation (i.e., the size of reference process models). Additionally, we considered a fixed distribution of the workflow relations in reference process models and a fixed distribution of variability relations and integrity constraints in the feature models. These decisions help us control the impact of these variables and only concentrate on the variables related to our research questions. Also a study over industrial feature models [164] shows that the average number of variability relations in feature models is around twenty five percent which shows the similar average distributions of variabilities in the models.

External validity investigates if the results of the experiments are generalized. We identify two main factors influencing the external validity of our experiments: the size of the models and the modeling approach. With respect to the size of reference process models, our investigation over existing publications in process management and the case-study with our industrial partner confirmed that the sizes of generated models are aligned with the size of existing models in the literature. Regarding, the proposed modeling and formalization approach of process models and feature models, we may have slightly different execution times for each distribution of independent variables for different formal representations. Thus, our results can not be generalized to other formalisms, which could be used as alternative for implementation of the proposed validation approach. However, the experimental results show that employing description logic can cope with different kinds of distributions.

9.10 Related Work

This section discusses the related works and compare them with our proposed approach using a set of criteria found in the literature.

9.10.1 Process Model Customization and Validation Techniques

Our proposed approach can be related to the research on (1) reference process models configuration and customization and (2) compliance validation of a customized or specialized process model with respect to a reference model. Reference process configuration and customization has been of interest to researchers in Process-Aware Information Systems (PAIS) and Software Product Lines (SPL) domains.

Configurable Reference Process Models in PAIS

Configurable Workflows - In [55], Gottschalk et al. discuss the requirements of configurable process modeling languages and propose a generic model which integrates possible process variations into one model. They introduce enable, hide, and block language constructs into workflow models. An algorithm and tooling support are provided to derive a workflow from a reference workflow. The approach is applied for both design and execution models. It also provides an individualization algorithm for customizing the process model and applies constraints inference [166] and Partner Synthesis [168] to ensure the correctness of the models.

C-EPCs - Rosemann et al. [137] extend EPC with configurable nodes (called C-EPCs). The extensions allow to specify variability in EPC process models. C-EPC provides process engineers with a set of guidelines to derive a configured process from an EPC model. Rosa et al [83] consider variability existing in control-flow, data-flow, and roles and extend C-EPC with variability relations for data-flow and roles (called C-iEPC). They also provide a questionnaire-based approach [84] for configuring C-EPCs models. Rosemann et al. [137] validate the correctness of the configured EPC models by constructing logical expression from guidelines and requirements and checking satisfiability of the expression.

Provop - Hallerbach et al. [65, 64] propose an approach (called Provop) to manage and configure process variants. In the Provop approach, a set of change operations are defined, which can be used by process designers to create process variants from some reference model. The Provop approach [64] defines a context model and constraints (e.g., implication, mutual exclusion, and option hierarchy) for a process family. It ensures soundness of a process model with respect to the context model and constraints which may be applied after each change operation over the process model.

Application-based Domain Modeling (ADOM) - In [132, 131], Application-

based Domain Modeling (ADOM) is proposed which enables configuring a reference process model [132]. They employ EPC [132] and BPMN [131] for representing reference process models. ADOM [132] empowered with an approach for validating customized process model against the rules and policies formulated in the reference process model. They limit the validation to behavioral inconsistencies between customized and reference process models.

DeltaProcess - Soto et al [157, 159, 114] proposed a comparison based approach called DeltaProcess for a compliance validation of a customized or specialised process model with respect to standard process models. The approach transforms the reference model and process instances into a RDF representation, produce a comparison model by performing an identity-based comparison of the RDF models and identify the changes in process model by recognizing the change patterns in the model. This helps in the identification of change inconsistencies in the customized process model. The technique is realized in a tool called *Evolyzer* and evaluated on validation of specialized models of V-Model XT.

Process Model Configuration in SPL

Recently, many researchers have started applying product line concepts in service-oriented computing [105, 28, 87, 120, 52]. We concentrate on approaches which apply SPL techniques for process model configuration.

PESOA - The Process Family Engineering in Service-Oriented Application (PESOA) project [144] proposes a variability mechanism for modeling variability and defining so-called variant-rich process models. These models are extended with stereotype annotations in order to accommodate variability and provide configuration options. During the configuration variation points are bound with one or more variant according to the type of the variation point. The approach does not provide concrete guidance on configuring feature models based on the requirements of the target application.

Superimposed Variants - Czarnecki et al. [34] applied feature models for representing variability in activity diagrams and proposed the use of Boolean variables called Presence Conditions (PC) for annotating activity diagrams mapped to features. The configuration of activity diagrams is enabled via configuring feature models and evaluating the PCs against the configuration. In [106, 105], Mohabbati et al. adopt feature modeling and presence condition notion to model process variability represented

in BPMN models and guide the configuration of reference process models.

Feature Model Composition - Acher et al. [1] define variability in process models by using feature models. They considered a process model as collection of services, each activity mapped to one service, which are related via data dependencies. They propose a merge operation over feature models to form a composite service from a set of variable services in a process. The approach does not provide guidance for configuring feature models. They perform consistency check of the process models by analyzing the input and output dataports of services based on dependency rules.

COVAMOF Framework - Sun et al. [3] and Koning et al [80] apply the COVAMOF framework to represent variability in the service composition. They utilize the configuration technique of COVAMOF for deriving a configured service composition from a variable service composition model. They also extended UML and BPEL with variation points and variants to incorporate variability into process models.

BPEL Customization Process - Mitzer et al. [103] propose the notion of variability descriptor for modeling variation points in the process layer of service-oriented application. The approach employ BPEL for representing process models. In their approach, the process model configuration is performed according to user inputs for variation points. They validate the customization with respect to variability constraints defined in the feature models. An eclipse plug-in is developed to model variability and transfer the configured feature model into BPEL process models.

Web Service Variability Description Language (WSVL) - Nguyen et al. [112] extend BPMN to describe variability in process-based service composition. In the process models, they represent variability in control flows, data flows, and message flows. They adopt a feature-oriented approach to develop a new language called Web Service Variability Description Language (WSVL) to represent variability. Feature-based configuration approaches are utilized for the configuration of service composition models. They do not ensure the behavioral and configuration consistency of the process model.

9.10.2 Criteria Based Comparison With Related Works

To compare our approach with the existing works, we defined a set of criteria related to process model customization and validation. These criteria were collected by adopting the criteria set defined in [84] as well as the bottom up investigation of related works. We divide the criteria set into three main categories *process and variability modeling,*

customization support and *general criteria*.

The process and variability modeling set includes the criteria exploring capability of approaches in modeling process models and their variability. *Implicit* and *explicit* variability representation criteria investigates if an approach represents the variability in the process model or uses a separate model for formalizing the variability of the process model. *Control flow variability*, *resource variability*, and *object variability* criteria indicates if an approach captures variability in control flow, resource, and objects, respectively. That is, if customization approach considers all source of variabilities in the process model. Variability type sub-criteria investigates which variability patterns were modeled using customization approach. According to existing research [63, 12] and our investigation the common variability patterns are alternative, OR, optional, workflow, and order variabilities. *Conceptual* and *execution* criteria specify if the approach supports design models or execution models.

Customization support criteria set explores approaches capability with respect to the customization process. *Restriction* and *Extension* criteria refer to the operations employed by the approach to generate the customized process model. In the former, the approach restricts the behaviour of a given process model using the delete operation while in the later a given process model is extended using the insert and modification operators. *abstraction level* and *guidance* criteria refer to the required level of knowledge from users and provided mechanisms and guidelines such as recommendation and semi-automatic configuration during customization process. *Behavioral correctness* and *configuration correctness* criteria check if the approach ensures the behavioral and configuration correctness of the customized process model with regard to a reference process model and configuration constraints. *Correctness scalability* criterion evaluates the scalability of approaches for ensuring behavioral and configuration correctness in a customized process model.

The general criteria set refers to general aspects not directly related to the development and customization of process models. *Tooling support* criterion checks if any tool was implemented to support the proposed approach. *Industrial Validation* criterion concern with exploring if an approach is verified using industrial models. Finally, *formalization* criterion investigates the level of approach maturity in terms of describing concrete algorithms and rigorous definitions.

Table 9.3 and 9.4 illustrate the result of the comparison. With respect to modeling

Table 9.3: Comparisons of reference process customization approaches with respect to process and variability modeling criteria (+ means criterion is met with the approach and – means criterion is not met with the approach)

Approaches	Criteria (Process and Variability Modeling)											
	Modeling Strategy		Source of Variability			Variability Scope		Variability Patterns				
	Implicit	explicit	Control-flow	Object	Resource	Execution	Conceptual	Alternative	OR	Optional	Workflow Variability	Order Variability
Configurable Workflow	+	–	+	–	–	+	+	–	–	+	+	–
C-EPCs	+	–	+	+	+	–	+	–	–	+	+	–
Provop	+	–	+	±	±	–	+	–	–	+	–	–
ADOM	+	–	+	–	–	–	+	–	–	+	+	–
DeltaProcess	+	–	+	+	–	–	+	–	–	+	–	–
PESOA	+	+	±	+	–	–	+	+	+	+	–	–
Superimposed Variants	–	+	+	–	–	–	+	+	+	+	–	–
Feature Model Composition	–	+	–	+	+	–	+	+	+	+	+	–
COVAMOF Framework	+	+	+	–	+	+	+	+	+	+	–	–
BPEL Customization Process	–	+	+	–	–	–	+	+	+	+	–	–
WSVL	+	+	+	–	–	–	+	+	+	+	–	–
Our approach	+	+	+	–	–	–	+	+	+	+	+	+

Table 9.4: Comparisons of reference process customization approaches with respect to customization and general criteria (+ means criterion is met with the approach, – means criterion is not met with the approach, NF means not found, E means process entity, and A means activity).

Approaches	Criteria									
	Customization Support							General Criteria		
	Restriction	Extension	abstraction Level	Guidance	Config. Correctness	Behav. Correctness	Scalability	Tooling Support	Industrial Validation	Formalization
Configurable Workflow	+	–	+	+	–	+	≤ 100	+	+	+
C-EPCs	+	+	+	+	–	+	792E	+	+	+
Provop	+	+	+	–	+	–	NF	±	±	±
ADOM	+	+	–	–	–	–	NF	–	±	+
DeltaProcess	–	–	–	–	+	+	2000E	+	+	+
PESOA	+	+	–	+	–	–	NF	±	+	±
Superimposed Variants	+	–	+	–	+	–	NF	+	–	+
Feature Model Composition	+	–	–	–	–	+	NF	+	–	+
COVAMOF Framework	+	–	+	+	–	–	NF	+	–	+
BPEL Customization Process	+	–	+	–	+	–	NF	+	–	±
WSVL	+	–	+	+	–	–	NF	+	–	+
Our approach	+	+	+	+	+	+	500A	+	+	+

strategy, we utilized both explicit and implicit approaches for modeling customization options (i.e. variability). By representing activity variability in the feature model, 1) we manage complexity related to modeling variability by separating variability concern from process modeling concern and 2) similar to many product line based approaches we present more complex activity variability types (i.e. Or, alternative, and optional) than the variability types that implicit based approaches can present (i.e. hide and block). Moreover, we also take behavioral based variability (i.e., workflow and order variabilities) into account which is neglected by many approaches. Currently, similar to all UML based and BPMN based approaches, we support variability modeling of control flows and not resources and objects. This can be explained with the fact that BPMN language used in our approach has limited support for capturing resources [84].

Similar to all approaches, our approach supports restriction strategy for process model customization. Additionally, similar to EPC, Provop, ADOM, DeltaProcess, and PESOA, we also apply extension strategy for the customization. Hence, our approach gives the modeler freedom to add or modify parts of the customized process model to satisfy *delta requirements*. In our approach, we mainly rely on restriction based to handle most of requirements via configuring feature models and removing deselected activities. Afterward, the defined change operations can be executed to derive the final customized process model.

Similar to configurable workflows and C-EPCs, COVAMOF, PEOSA, and WSVL, we provide step-by-step guideline for customizing reference process models. For the configuration step we provided planning and linear programming techniques [155, 106] to derive a configured process from a reference process model based on the stakeholders' requirements. For the customization step, we described a set of operations that can be used by process models to develop the final customized process model. Through mapping between feature models and process models, we provide an end-user perspective for customizing process models which abstract stakeholders from process model elements.

All the approaches including our approach support customization during design time and based on conceptual modeling languages (i.e. UML, BPMN, EPC). All the investigated approach except configurable workflows and COVAMOF framework do not support executable model customization.

With respect to correctness, we considered both behavioral and configuration correctness and evaluated the scalability of our approach for large size business process models

(process models with 500 activities). Majority of approaches concentrated on the behavioral and structural correctness and less attention has been paid to configuration correctness. Configuration correctness is important in the extension based approaches that extensions may violate the configuration constraints defined over the process models.

9.11 Conclusions

In this paper, we have presented a novel approach for handling inconsistencies resulting from the customization of business process models, which were originally derived from references process through configuration procedures. By automatically identifying inconsistencies, we are able to detect if customized process models are consistent with the behavioral and configuration knowledge encoded in reference process models. Moreover, the presented approach is based on a formal framework which further enables to support complex reasoning for verification tasks. Our contribution extends the body of research knowledge by expanding the perspective to automated consistency checking of customized process models and not only of configurable process models. We plan to extend this approach in combination with our existing work on validation of user intentions (expressed as goal-oriented models), so that a complete development life-cycle for configuration and customization of reference process models is supported.

Chapter 10

Conclusions and Future Works

This chapter concludes the thesis, highlights the limitations, and provides an outlook of future works. First, we revisit the challenges mentioned in introduction chapter and explain how the proposed techniques address these challenges. Next, open research problems for the future works are explained.

10.1 Summary and Main Contributions

Following the research method, described in introduction chapter, we preformed theoretical analysis [12] and systematic mapping study [104] and identified five main challenges in the context of customizable process models: *variability complexity, modeling complexity, configuration complexity, delta requirements,* and *customization validation.* Afterward, our research endeavor concentrates on addressing these challenges by proposing customization and validation techniques.

Variability complexity: Variability plays a pivotal role in systematic reusability in customizable process models. Hence, it becomes necessary to carefully investigate the variability notion in the context of customizable process models. In order to address this need, we developed a formal framework based on the ontological theory to enable a theoretical analysis of variability. Ontological understanding of variability is beneficial from several aspects. First, based on the results of ontological analysis, process engineers can clearly identify sources of variability in process aware information systems and consider those sources during domain engineering lifecycle. Second, the possible patterns of

variability (i.e., types of differences) among processes can be identified. After the formal analysis of variability and the investigation of variability in the existing literature on customizable process models and service oriented product lines [104], we presented different variability patterns and dependency relations in customizable process models (see [13]).

Modeling Complexity: After the analysis of variability, we concentrated on the representations of variability. We had two possible options for illustrating the variability of customizable process models: Similar to [55, 83], incorporate variability into customizable process models; or formulate variability in a separate variability language. We choose a combination of both approaches where a part of variability (e.g., activity variability) is represented in a variability dimension and part of variability (e.g., workflow variability) is represented in process models. This decision helps manage the modeling complexity of customizable process models by separating the parts of the variability from customizable process models. Moreover, a unified representation of variability facilitates the adaptation of customizable process models. Among existing variability modeling languages, we selected feature models which are commonly used in the context of software product lines and according to our evaluation in [12] provide rich constructs for representing variability patterns. Representing variability in feature models requires establishing the mapping between features and process elements in the customizable process models. To develop the mapping between process elements (activities and subprocesses) and the features, we based our work on existing works in software product lines community and applied an annotation based mapping mechanisms proposed by Czarnecki et al [33].

Configuration Complexity: Configuration process refers to resolving variabilities and adapting a customizable process model based on the given requirements of stakeholders. Unlike the development of customizable process models which is executed once, the configuration process is executed whenever a new process is needed. Therefore, facilitating process engineers' tasks during configuration process is very important in the context of customizable process models. Several challenges regarding to the configuration process needs to be resolved: 1) stakeholders express their requirements in terms of their goal and objectives while customizable process models are defined in technical point of view; 2) industrial customizable process models are very large with many activities and variability relations; 3) activities may have different quality attributes

which need to be considered during the configuration process. In order to resolve these challenges, inspired from software product lines research [35], we proposed *two staged configuration process, employed goal oriented requirements engineering techniques,* and *automated parts of configuration process.* By utilizing goal models, we provided a convenient way for capturing stakeholders intentions, mapping them as concrete requirements into features, and feeding the process of the process configuration with a more accurate set of stakeholders desired features. In the second stage of the configuration, a decision making technique (Stratified Analytical Hierarchy Process (S-AHP)) was proposed which utilizes multi-criteria decision making techniques and computes the ranks of features based on the preferences of stakeholders. Afterward, a model transformation technique was devised to transfer feature model to PDDL planning problem. Finally, existing planners were used to find an optimal plan which is corresponding to the optimal feature model configuration.

Delta Requirements: In order to handle delta requirements, we introduced a set of customization steps which can be used for developing final customized process models. The customization steps are empowered with validation mechanisms which inform process engineers about possible behavioral or configuration inconsistencies.

Customization Validation: In our approach, three main artifacts were developed: family goal models, customizable process models, and feature models. These artifacts are linked to each other through mapping relations. The validation component in our approach verifies the alignment of these artifacts by considering the mapping between them. The validation techniques support both domain engineering and application engineering lifecycles. In the domain engineering lifecycle, the proposed validation assures that intentional variability, captured in goal models, does not violate constraints of technical variability captured in feature models. The proposed validation approach compares the influence of intentional relations given by a goal model with feature relations from the feature model. Also, inconsistencies in mappings between goal models and business process models are handled. With respect to the alignment between goal model and process model, we distinguish between two kinds of inconsistencies. (i) Inconsistencies of the process orchestration are caused by contradicting control flow relationships between activities in the process model and between relationships of user intentions in the goal model. (ii) Choreography inconsistencies occur if dependencies among actors in the goal model are not reflected by message exchange between the corresponding activities in the

process models. By automatically identifying these inconsistencies, we are able to detect executable processes that did not meet user requirements or lead to undesired executions. Additionally, we allow for goal models and process models to evolve independently. In application engineering, the validation technique concentrates on handling inconsistencies resulting from the customization of configured process models, which were originally derived from customizable process through configuration procedures. By automatically identifying inconsistencies, we are able to detect if customized process models are consistent with the behavioral and configuration knowledge encoded in customizable process models. Moreover, the presented approach is based on a formal framework which further enables to support complex reasoning for verification tasks.

10.2 Limitations

The work on developing and validating customizable process models presented in this dissertation includes the following limitations:

Design time customization: We only focused on the process model customization and validation in the design time. This means that in our approach the binding time for variation points is before run-time. However, the context and environment of a process-aware information system may change during its execution. Hence, in addition to the design time customization of process models, it becomes important to adapt process models at run-time to their changing context and environment. Managing run-time adaptation requires changing the perspective from viewing the product line as a whole to viewing one product at a time [121]. Furthermore, run-time reconfiguration techniques must automatically identify configuration requirements and change a system according to the requirements without breaking the execution of the system. Some researchers in software product lines area initiated the idea of Dynamic Software Product Lines (DSPL) [87, 121, 122] which aims at investigating mechanisms to address run-time variability and dynamic reconfiguration.

Variability in other sources: In addition to activities and control flow relations between activities, process models represent resources, roles, and data objects [136]. Hence, process models may vary in their data objects, resources, and roles. The variability management in our research is limited to activities and control-flows. To this end, a comprehensive investigation on the representation of variability in other sources and the integration of these variability aspects are required.

Requirements representation: We represented the requirements using goal models and captured the preferences of stakeholders in terms of relative importance. This may lead to two limitations of our approach: first, goal models might become very complex and hard to understand for large size process-aware information systems. Second, stakeholders are limited to a specific way of expressing their preferences. Goal models are commonly used in the requirements engineering phase and many efforts have been done to improve their understandability by developing tooling support and applying visualization [7]. Moreover, the formalization of requirements in terms of goal models enables the validation of process models with respect to intentional relations in goal models.

Mapping development: We provided validation mechanisms to ensure the correctness of mappings between different models. However, no facilities have been provided in our approach to help process engineers in developing mappings between models. We assumed that the mapping between these models are developed by process engineers. For large size models, developing mapping relations is a cumbersome task for process engineers and semi-automatic techniques are required for developing the mappings. To this end, a future direction could be utilizing the ontologies for semi-automating the mapping process.

Industrial evaluation: We evaluated parts of our approach using two case-studies (i.e., online shopping and health care case study) and simulations. However, a comprehensive evaluation of the approach in the industrial setting has not been done.

10.3 Future Works

There are several research directions and possible extensions to this body of research.
Mining variability patterns: In an organization, several process models may exist in different organization unites. Furthermore, within an organizational unit, a process model may be executed differently based on unique customers' requirements. Each unique execution instance is also referred as a case [107]. The comprehension of (1) these process models, (2) their differences and similarities, (3) and interrelationships between customers requirements and cases, is essential and challenging to develop customizable process models. A future line of research may involve discovering, analyzing, and tracking the executions of processes and cases in order to mine variability patterns from execution logs.

Process model recommendation: Our proposed configuration approach concentrates on the optimization of the customized process model based on the given requirements of stakeholders. As we mentioned previously, the execution of process models provides knowledge for discovering variability patterns. Additionally, process mining approaches can be utilized to 1) mine configuration association rules which can be applied during the configuration of customizable process models; and 2) recommend future steps during the process execution based on the case information. In this regard, recommendation algorithms can be explored and applied in the context of customizable process models.

Quality aggregation for customizable process models: Quality evaluation is a challenging task in process aware information systems. Several techniques have been proposed to aggregate quality values over process models [30, 69]. Considering variability notion in customizable process models, a new set of quality aggregation techniques is required which consider both variability and workflow relations when aggregating quality attributes. Mohabbati et al. [105] proposed some initial ideas on quality aggregation which can be a basis for further investigation.

Bibliography

[1] Mathieu Acher, Philippe Collet, Philippe Lahire, and Robert France. Managing variability in workflow with feature model composition operators. In *Proceedings of the 9th International Conference on Software Composition*, SC'10, pages 17–33, Berlin, Heidelberg, 2010. Springer-Verlag. 34, 220

[2] Mathieu Acher, Philippe Collet, Philippe Lahire, and Robert B. France. Separation of concerns in feature modeling: support and applications. In *Proceedings of the 11th annual international conference on Aspect-oriented Software Development*, AOSD '12, pages 1–12, New York, NY, USA, 2012. ACM. 142

[3] Chang ai Sun, Rowan Rossing, Marco Sinnema, Pavel Bulanov, and Marco Aiello. Modeling and managing the variability of web service-based systems. *Journal of Systems and Software*, 83(3):502 – 516, 2010. 220

[4] Marco Aiello, Pavel Bulanov, and Heerko Groefsema. Requirements and tools for variability management. In *Proceedings of the 2010 IEEE 34th Annual Computer Software and Applications Conference Workshops*, COMPSACW '10, pages 245–250, Washington, DC, USA, 2010. IEEE Computer Society. 186, 187, 189

[5] Raian Ali, Fabiano Dalpiaz, and Paolo Giorgini. A goal-based framework for contextual requirements modeling and analysis. *Requir. Eng.*, 15(4):439–458, November 2010. 176, 177, 178

[6] Raian Ali, Fabiano Dalpiaz, and Paolo Giorgini. Reasoning with contextual requirements: Detecting inconsistency and conflicts. *Information and Software Technology*, 55(1):35 – 57, 2013. ¡ce:title¿Special section: Best papers from the 2nd International Symposium on Search Based Software Engineering 2010¡/ce:title¿. 176

[7] Daniel Amyot, Sepideh Ghanavati, Jennifer Horkoff, Gunter Mussbacher, Liam Peyton, and Eric Yu. Evaluating Goal Models within the Goal-Oriented Requirement Language. *International Journal on Intelligent Systems*, 25:841–877, 2010. 13, 178, 229

[8] B. Andersson, M. Bergholtz, A. Edirisuriya, T. Ilayperuma, and P. Johannesson. A Declarative Foundation of Process Models. In *Advanced Information Systems Engineering*, pages 233–247. Springer, 2005. 110

[9] Annie I. Anton. Goal-based requirements analysis. In *Proceedings of the Second International Conference on Requirements Engineering*, pages 136–144, Apr 1996. 62

[10] Sandra António, João Araújo, and Carla Silva. Adapting the i* Framework for Software Product Lines. In *Proc. of the ER 2009 Workshops on Advances in Conceptual Modeling - Challenging Perspectives*, pages 286–295. Springer, 2009. 62, 76, 77, 144, 175, 177

[11] Mohsen Asadi, Ebrahim Bagheri, Dragan Gasevic, and Marek Hatala. Goal-Driven Software Product Line Engineering. In *26th ACM Symposium on Applied Computing*, 2011. 142, 149, 151, 153

[12] Mohsen Asadi, Dragan Gasevic, Yair Wand, and Marek Hatala. Deriving variability patterns in software product lines by ontological considerations. In *31st International Conference on Conceptual Modeling*, volume 7532 of *Lecture Notes in Computer Science*, pages 397–498. Springer Berlin Heidelberg, 2012. 4, 30, 84, 182, 186, 190, 221, 225, 226

[13] Mohsen Asadi, Bardia Mohabbati, Gerd Grner, and Dragan Gasevic. Development and validation of customized process models. *Journal of Systems and Software*, (0):–, 2014. 29, 42, 226

[14] Mohsen Asadi, Bardia Mohabbati, Gerd Grner, and Dragan Gasevic. Validation of user intentions in feature-oriented software families. *Journal of Software and Systems*, (0):–, 2014. 39

[15] Mohsen Asadi, Samaneh Soltani, Dragan Gasevic, and Marek Hatala. The effects of visualization and interaction techniques on feature model configuration. *Empirical Software Engineering Journal*, 37:33, Accepted 2014. 216

[16] F. Baader, D. Calvanese, D.L. McGuinness, D. Nardi, and P. Patel-Schneider. *The Description Logic Handbook*. Cambridge University Press, 2007. 22, 120, 199

[17] Ebrahim Bagheri, Mohsen Asadi, Dragan Gasevic, and Samaneh Soltani. Stratified analytic hierarchy process: Prioritization and selection of software features. In Jan Bosch and Jaejoon Lee, editors, *SPLC*, volume 6287 of *Lecture Notes in Computer Science*, pages 300–315. Springer, 2010. 5, 73, 78, 81, 87, 88, 104, 105, 106, 107

[18] Ebrahim Bagheri and Dragan Gasevic. Assessing the maintainability of software product line feature models using structural metrics. *Software Quality Journal*, 19(3):579–612, 2011. 97, 212

[19] Alistair Barros, Marlon Dumas, and Arthur Ter Hofstede. Service Interaction Patterns: Towards a Reference Framework for Service-based Business Process Interconnection. Technical report, 2005. 118

[20] Victor R. Basili. The experimental paradigm in software engineering. In *Proceedings of the International Workshop on Experimental Software Engineering Issues: Critical Assessment and Future Directions*, pages 3–12, London, UK, UK, 1993. Springer-Verlag. 6, 7

[21] Don Batory. Feature models, grammars, and propositional formulas. In J. Henk Obbink and Klaus Pohl, editors, *SPLC*, volume 3714 of *Lecture Notes in Computer Science*, pages 7–20. Springer, 2005. 9, 20, 21, 66, 184, 189

[22] Don Batory, David Benavides, and Antonio Ruiz-Cortes. Automated Analysis of Feature Models: Challenges Ahead. *Communications of ACM*, 49:45–47, 2006. 141, 146, 167

[23] David Benavides, Pablo Trinidad Martn-Arroyo, and Antonio Ruiz Corts. Automated reasoning on feature models. In Oscar Pastor and Joo Falco e Cunha, editors, *CAiSE*, volume 3520 of *Lecture Notes in Computer Science*, pages 491–503. Springer, 2005. 5, 9, 21, 34, 82, 103, 105, 106, 107

[24] Maria Bergholtz, Prasad Jayaweera, Paul Johannesson, and Petia Wohed. A Pattern and Dependency Based Approach to the Design of Process Models. In *Conceptual Modeling - ER, 23rd Int. Conference*, volume 3288 of *LNCS*, pages 724–739. Springer, 2004. 110

[25] Clarissa Borba and Carla Silva. A Comparison of Goal-Oriented Approaches to Model Software Product Lines Variability. In *Proc. of the ER 2009 Workshops on Advances in Conceptual Modeling - Challenging Perspectives*, pages 244–253. Springer, 2009. 63, 77, 174, 175, 177

[26] J. Bosch, G. Florijn, D. Greefhorst, J. Kuusela, J. Obbink, and K. Pohl. Variability issues in software product lines. *Software Product-Family Engineering*, pages 303–338, 2002. 3, 181

[27] Jan Bosch. *Design and use of software architectures - adopting and evolving a product-line approach.* Addison-Wesley, 2000. 20, 56

[28] Oldrich Bubak and Hassan Gomaa. Applying software product line concepts in service orientation. *Int. J. Intell. Inf. Database Syst.*, 2(4):383–396, November 2008. 219

[29] M Bunge. *Treatise on Basic Philosophy: Volume 3: Ontology I: The Furniture of the World.* Reidel, Boston, MA, 1977. 13

[30] Jorge Cardoso, Amit Sheth, John Miller, Jonathan Arnold, and Krys Kochut. Quality of Service for Workflows and Web Service Processes. *Journal of Web Semantics*, 1(3), 2004. 230

[31] Lawrence Chung, Brian A. Nixon, Eric Yu, and John Mylopoulos. *Non-Functional Requirements in Software Engineering.* Springer, 1st edition, 1999. 13

[32] Dave Clarke and José Proença. Towards a theory of views for feature models. In *SPLC Workshops*, pages 91–98, 2010. 142

[33] Krzysztof Czarnecki and Michal Antkiewicz. Mapping features to models: A template approach based on superimposed variants. In *GPCE'05*, volume 3676 of *LNCS*, pages 422–437. Springer, 2005. xvii, 64, 65, 67, 68, 73, 77, 85, 96, 149, 226

[34] Krzysztof Czarnecki and Ulrich W. Eisenecker. *Generative programming: methods, tools, and applications*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 2000. 3, 9, 20, 21, 178, 182, 189, 219

[35] Krzysztof Czarnecki, Simon Helsen, and Ulrich W. Eisenecker. Staged configuration through specialization and multilevel configuration of feature models. *Software Process: Improvement and Practice*, 10(2):143–169, 2005. 70, 73, 184, 191, 193, 227

[36] Krzysztof Czarnecki and Krzysztof Pietroszek. Verifying Feature-based Model Templates Against Well-formedness OCL Constraints. In *GPCE*, pages 211–220, 2006. 178

[37] Gero Decker and Frank Puhlmann. Extending BPMN for Modeling Complex Choreographies. In *Proceedings of the 2007 OTM Confederated international conference on On the move to meaningful internet systems: CoopIS, DOA, ODBASE, GADA, and IS - Volume Part I*, OTM'07, pages 24–40, Berlin, Heidelberg, 2007. Springer-Verlag. 118

[38] Ken Decreus and Geert Poels. A Goal-Oriented Requirements Engineering Method for Business Processes. In *CAiSE Forum*, volume 72 of *LNCS*, pages 29–43, 2010. 137

[39] Sybren Deelstra, Marco Sinnema, and Jan Bosch. Product derivation in software product families: a case study. *J. Syst. Softw.*, 74(2):173–194, January 2005. 3, 181

[40] Sybren Deelstra, Marco Sinnema, and Jan Bosch. Variability assessment in software product families. *Inf. Softw. Technol.*, 51(1):195–218, January 2009. 26

[41] R. A. DeMillo, R. J. Lipton, and F. G. Sayward. Hints on Test Data Selection: Help for the Practicing Programmer. *Computer*, 11(4):34–41, April 1978. 130, 211

[42] Olfa Djebbi and Camille Salinesi. Criteria for comparing requirements variability modeling notations for product lines. In *Proceedings of the Fourth Internationa Workshop on Comparative Evaluation in Requirements Engineering*, CERE '06, pages 20–35, Washington, DC, USA, 2006. IEEE Computer Society. 48, 59

[43] Marlon Dumas, Jan Recker, and Mathias Weske. Management and Engineering of Process-aware Information Systems: Introduction to the Special Issue. *Information Systems*, 37(2):77 – 79, 2012. 17, 109

[44] Marlon Dumas, Marcello La Rosa, Jan Mendling, and Hajo A. Reijers. *Fundamentals of Business Process Management*. Springer, 2013. 1, 2

[45] Marlon Dumas, Wil M. van der Aalst, and Arthur H. ter Hofstede. *Process-aware Information Systems: Bridging People and Software Through Process Technology*. John Wiley & Sons, Inc., New York, NY, USA, 2005. 1, 2, 80

[46] Neil A. Ernst, Alexander Borgida, John Mylopoulos, and Ivan J. Jureta. Agile requirements evolution via paraconsistent reasoning. In *Proceedings of the 24th international conference on Advanced Information Systems Engineering*, CAiSE'12, pages 382–397, Berlin, Heidelberg, 2012. Springer-Verlag. 176, 177

[47] Joerg Evermann and Yair Wand. Ontology based object-oriented domain modelling: Fundamental concepts. *Requir. Eng.*, 10(2):146–160, May 2005. 13, 48, 57

[48] P.H. Feiler and W.S. Humphrey. Software Process Development and Enactment: Concepts and Definitions. In *2nd International Conference on the Software Proces, Continuous Software Process Improvement*, pages 28 –40, feb 1993. 17

[49] Maria Fox and Derek Long. Pddl2.1: An extension to pddl for expressing temporal planning domains. *J. Artif. Int. Res.*, 20(1):61–124, December 2003. 22, 89

[50] Ganna Frankova, Ganna Frankova, Fabio Massacci, Fabio Massacci, Magali Seguran, and Magali Seguran. From Early Requirements Analysis towards Secure Workflows. Technical Report TR DIT-07-036, University of Trento, 2007. 137

[51] Bernd Freimut, Teade Punter, Stefan Biffl, and Marcus Ciolkowski. State-of-the-art in empirical studies authors:, 2002. 8

[52] Matthias Galster and Armin Eberlein. Identifying potential core assets in service-based systems to support the transition to service-oriented product lines. In *ECBS*, pages 179–186, 2011. 219

[53] Paolo Giorgini, John Mylopoulos, Eleonora Nicchiarelli, and Roberto Sebastiani. Reasoning with goal models. In *Proceedings of the 21st International Conference on Conceptual Modeling*, ER '02, pages 167–181, London, UK, UK, 2002. Springer-Verlag. 34, 73

[54] Paolo Giorgini, John Mylopoulos, and Roberto Sebastiani. Goal-oriented Requirements Analysis and Reasoning in the TROPOS Methodology. *Engineering Applications of Artificial Intelligence*, 18:159–171, 2005. 13, 151, 178

[55] F. Gottschalk, W. M.P Van Der Aalst, M. H Jansen-Vullers, and M. La Rosa. Configurable workflow models. *Int. J. of Cooperative Inf. Syst.*, 17(2):177221, 2008. 3, 5, 6, 9, 81, 102, 103, 106, 107, 138, 182, 183, 184, 218, 226

[56] Florian Gottschalk, Teun A. Wagemakers, Monique H. Jansen-Vullers, Wil M. Aalst, and Marcello Rosa. Configurable process models: Experiences from a municipality case study. In *Proceedings of the 21st International Conference on Advanced Information Systems Engineering*, CAiSE '09, pages 486–500, Berlin, Heidelberg, 2009. Springer-Verlag. 3, 182

[57] Gerd Grner, Mohsen Asadi, Bardia Mohabbati, Dragan Gaevi, Marko Bokovi, and Fernando Silva Parreiras. Validation of user intentions in process orchestration and choreography. *Information Systems*, 43(0):83 – 99, 2014. 38, 155

[58] Gerd Groner, Christian Wende, Marko Bošković, Fernando Silva Parreiras, Tobias Walter, Florian Heidenreich, Dragan Gašević, and Steffen Staab. Validation of Families of Business Processes. In *23rd Int. Conf. on Advanced Information Systems Engineering*, LNCS, pages 551–565, 2011. 3, 5

[59] Gerd Groner, Christian Wende, Marko Bošković, Fernando Silva Parreiras, Tobias Walter, Florian Heidenreich, Dragan Gašević, and Steffen Staab. Validation of Families of Business Processes. In *23rd Int. Conference on Advanced Information Systems Engineering (CAiSE)*, volume 6741 of *LNCS*, pages 551–565. Springer, 2011. 82, 139, 184, 211

[60] Jianmei Guo, Yinglin Wang, Pablo Trinidad, and David Benavides. Consistency maintenance for evolving feature models. *Expert Syst. Appl.*, 39(5):4987–4998, 2012. 40, 41

[61] Jianmei Guo, Yinglin Wang, Pablo Trinidad, and David Benavides. Consistency maintenance for evolving feature models. *Expert Syst. Appl.*, 39(5):4987–4998, 2012. 130, 167, 197, 198, 211

[62] Jianmei Guo, Jules White, Guangxin Wang, Jian Li, and Yinglin Wang. A genetic algorithm for optimized feature selection with resource constraints in software product lines. *J. Syst. Softw.*, 84(12):2208–2221, December 2011. 96, 102, 104, 105, 106, 212

[63] Jilles Van Gurp, Jan Bosch, and Mikael Svahnberg. On the notion of variability in software product lines. In *Proceedings of the Working IEEE/IFIP Conference on Software Architecture*, WICSA '01, pages 45–, Washington, DC, USA, 2001. IEEE Computer Society. 49, 221

[64] Alena Hallerbach, Thomas Bauer, and Manfred Reichert. Guaranteeing Soundness of Configurable Process Variants in Provop. In *Proceedings of the 2009 IEEE Conference on Commerce and Enterprise Computing*, CEC '09, pages 98–105, Washington, DC, USA, 2009. IEEE Computer Society. 10, 185, 218

[65] Alena Hallerbach, Thomas Bauer, and Manfred Reichert. Capturing Variability in Business Process Models: the Provop Approach. *J. Softw. Maint. Evol.*, 22(6-7):519–546, 2010. 2, 5, 9, 36, 81, 84, 181, 182, 184, 187, 192, 218

[66] Florian Heidenreich, Jan Kopcsek, and Christian Wende. FeatureMapper: Mapping Features to Models. In *Companion Proceedings of the 30th International Conference on Software Engineering (ICSE'08)*, pages 943–944. ACM, 2008. 77, 151

[67] Patrick Heymans, Pierre-Yves Schobbens, Jean-Christophe Trigaux, Raimundas Matulevicius, Andreas Classen, and Yves Bontemps. Towards the comparative evaluation of feature diagram languages. In *Proceedings of the Software and Services Variability Management Workshop - Concepts, Models and Tools*, Helsinki, Finland, April 2007. 48, 59

[68] ITU-T. Recommendation Z.151 (09/08): User Requirements Notation (URN) Language Definition, 2008. 13, 14, 148

[69] Michael C. Jaeger, Gregor Rojec-Goldmann, and Gero Muhl. Qos aggregation for web service composition using workflow patterns. In *Proceedings of the Enterprise Distributed Object Computing Conference, Eighth IEEE International*, EDOC '04, pages 149–159, Washington, DC, USA, 2004. IEEE Computer Society. 230

[70] Mikolás Janota and Goetz Botterweck. Formal Approach to Integrating Feature and Architecture Models. In *FASE*, volume 4961 of *LNCS*, pages 31–45, 2008. 178

[71] Richard Johnson, David Pearson, and Keshav Pingali. The Program Structure Tree: Computing Control Regions in Linear Time. In *PLDI*, pages 171–185, 1994. 18

[72] I.J. Jureta, A. Borgida, N.A. Ernst, and J. Mylopoulos. Techne: Towards a new generation of requirements modeling languages with goals, preferences, and inconsistency handling. In *Requirements Engineering Conference (RE), 2010 18th IEEE International*, pages 115 –124, 27 2010-oct. 1 2010. 176, 177, 178

[73] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-oriented domain analysis (foda) feasibility study. Technical report, Carnegie-Mellon University Software Engineering Institute, November 1990. 8, 47, 48, 55, 56, 82, 141, 183, 189

[74] Kyo C. Kang, Sajoong Kim, Jaejoon Lee, Kijoo Kim, Euiseob Shin, and Moonhang Huh. Form: A feature-oriented reuse method with domain-specific reference architectures. *Ann. Softw. Eng.*, 5:143–168, January 1998. 146, 147, 167, 178

[75] Raman Kazhamiakin, Marco Pistore, and Marco Roveri. A Framework for Integrating Business Processes and Business Requirements. In *Proc. of IEEE EDOC Conference*, pages 9–20, 2004. 31, 38, 113, 119, 137

[76] Marc I. Kellner, Raymond J. Madachy, and David M. Raffo. Software Process Simulation Modeling: Why? What? *Journal of Systems and Software*, 46:91–105, 1999. 96, 130, 167, 211

[77] Bartek Kiepuszewski, Arthur H. M. ter Hofstede, and Christoph Bussler. On Structured Workflow Modelling. In *12th Int. Conference on Advanced Information Systems Engineering*, CAiSE, pages 431–445. Springer, London, UK, 2000. 18

[78] Barbara A. Kitchenham, Shari Lawrence Pfleeger, Lesley M. Pickard, Peter W. Jones, David C. Hoaglin, Khaled El Emam, and Jarrett Rosenberg. Preliminary guidelines for empirical research in software engineering. *IEEE Trans. Softw. Eng.*, 28(8):721–734, August 2002. 8

[79] George Koliadis, Aleksandar Vranesevic, Moshiur Bhuiyan, Aneesh Krishna, and Aditya K. Ghose. Combining i* and BPMN for Business Process Model Lifecycle Management. In *Business Process Management Workshops*, volume 4103 of *LNCS*, pages 416–427. Springer, 2006. 31, 38, 110, 113, 119, 137

[80] Michiel Koning, Chang-ai Sun, Marco Sinnema, and Paris Avgeriou. Vxbpel: Supporting variability for web services in bpel. *Inf. Softw. Technol.*, 51(2):258–269, February 2009. 220

[81] Peter Kueng and Peter Kawalek. Goal-based Business Process Models: Creation and evaluation. *Business Process Management Journal*, pages 17–38, 1997. 110

[82] Jochen Malte Kuster, Christian Gerth, Alexander Forster, and Gregor Engels. Detecting and Resolving Process Model Differences in the Absence of a Change Log. In *Business Process Management, 6th Int. Conference, BPM*, volume 5240 of *LNCS*, pages 244–260. Springer, 2008. 18

[83] Marcello La Rosa, Marlon Dumas, Arthur H. M. ter Hofstede, and Jan Mendling. Configurable multi-perspective business process models. *Inf. Syst.*, 36(2):313–340, April 2011. 2, 3, 4, 5, 182, 218, 226

[84] Marcello La Rosa, Wil van der Aalst, Marlon Dumas, and Arthur ter Hofstede. Questionnaire-based Variability Modeling for System Configuration. *Software and Systems Modeling*, 8:251–274, 2009. 81, 103, 138, 181, 182, 212, 218, 220, 223

[85] Alexei Lapouchnian, Yijun Yu, Sotirios Liaskos, and John Mylopoulos. Requirements-driven design of autonomic application software. In *Proceedings of the 2006 Conference of the Center for Advanced Studies on Collaborative Research*, CASCON '06, Riverton, NJ, USA, 2006. IBM Corp. 62, 66, 76

[86] Alexei Lapouchnian, Yijun Yu, and John Mylopoulos. Requirements-Driven Design and Configuration Management of Business Processes. In *Business Process Management, 5th Int. Conference, BPM*, volume 4714 of *LNCS*, pages 246–261. Springer, 2007. 137, 142, 174, 177, 178

[87] Jaejoon Lee and Gerald Kotonya. Combining service-orientation with product line engineering. *IEEE Softw.*, 27(3):35–41, May 2010. 219, 228

[88] Kwanwoo Lee, Kyo Chul Kang, and Jaejoon Lee. Concepts and guidelines of feature modeling for product line software engineering. In *Proceedings of the 7th International Conference on Software Reuse: Methods, Techniques, and Tools*, ICSR-7, pages 62–77, London, UK, UK, 2002. Springer-Verlag. 66

[89] Richard Lenz and Manfred Reichert. It support for healthcare processes - premises, challenges, perspectives. *Data Knowl. Eng.*, 61(1):39–58, 2007. 181

[90] S. Liaskos, A. Lapouchnian, Yiqiao Wang, Yijun Yu, and S. Easterbrook. Configuring Common Personal Software: a Requirements-Driven Approach. In *13th IEEE Int. Conference Requirements Engineering,*, pages 9 – 18, 2005. 76, 145, 176, 177

[91] Sotirios Liaskos, Alexei Lapouchnian, Yijun Yu, Eric Yu, and John Mylopoulos. On goal-based variability acquisition and analysis. In *Proceedings of the 14th IEEE International Requirements Engineering Conference*, RE '06, pages 76–85, Washington, DC, USA, 2006. IEEE Computer Society. 28, 111, 149, 151

[92] Sotirios Liaskos, Marin Litoiu, Marina Daoud Jungblut, and John Mylopoulos. Goal-based behavioral customization of information systems. In *Proceedings of the 23rd international conference on Advanced information systems engineering*, CAiSE'11, pages 77–92, Berlin, Heidelberg, 2011. Springer-Verlag. 27, 138, 145

[93] Sotirios Liaskos, Sheila A. McIlraith, Shirin Sohrabi, and John Mylopoulos. Representing and reasoning about preferences in requirements engineering. *Requir. Eng.*, 16(3):227–249, September 2011. 168

[94] Ruopeng Lu, Shazia Sadiq, and Guido Governatori. On managing business processes variants. *Data Knowl. Eng.*, 68(7):642–664, July 2009. 186

[95] Carsten Lutz, Frank Wolter, and Michael Zakharyaschev. Temporal Description Logics: A Survey. In *15th International Symposium on Temporal Representation and Reasoning (TIMETIME)*, pages 3–14. IEEE Computer Society, 2008. 137

[96] Ayman Mahfouz, Leonor Barroca, Robin C. Laney, and Bashar Nuseibeh. From Organizational Requirements to Service Choreography. In *World Conference on Services-I*, pages 546–553. IEEE, 2009. 31, 113, 119

[97] Ayman Mahfouz, Leonor Barroca, Robin C. Laney, and Bashar Nuseibeh. Requirements-Driven Collaborative Choreography Customization. In *ICSOC/ServiceWave*, pages 144–158, 2009. 138

[98] Dewi Mairiza, Didar Zowghi, and Nurie Nurmuliani. An investigation into the notion of non-functional requirements. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC '10, pages 311–317. ACM, 2010. 97, 102

[99] Ivan Markovic and Marek Kowalkiewicz. Linking business goals to process models in semantic business process modeling. In *The 12th International IEEE Enterprise Distributed Object Computing Conference, EDOC '08*, pages 332 –338, 2008. 138

[100] J.D. McGregor, D. Muthig, K. Yoshimura, and P. Jensen. Successful Software Product Line Practices. *Software, IEEE*, 27(3):16 –21, 2010. 178

[101] Jan Mendling. *Metrics for Process Models: Empirical Foundations of Verification, Error Prediction, and Guidelines for Correctness*. Springer, 1 edition, 2008. 97, 131, 211

[102] A. Metzger, K. Pohl, P. Heymans, P. Schobbens, and G. Saval. Disambiguating the documentation of variability in software product lines: A separation of concerns, formalization and automated analysis. In *Requirements Engineering Conference, 2007. RE '07. 15th IEEE International*, pages 243–253, Oct 2007. 25, 27, 49, 56, 142, 145

[103] R. Mietzner and F. Leymann. Generation of bpel customization processes for saas applications from variability descriptors. In *Services Computing, 2008. SCC '08. IEEE International Conference on*, volume 2, pages 359 –366, july 2008. 220

[104] Bardia Mohabbati, Mohsen Asadi, Dragan Gašević, Marek Hatala, and Hausi A. Muller. Combining service-orientation and software product line engineering: A systematic mapping study. *Inf. Softw. Technol.*, 55(11):1845–1859, November 2013. 225, 226

[105] Bardia Mohabbati, Dragan Gašević, Marek Hatala, Mohsen Asadi, Ebrahim Bagheri, and Marko Bošković. A quality aggregation model for service-oriented software product lines based on variability and composition patterns. ICSOC'11, pages 436–451, 2011. 33, 184, 191, 219, 230

[106] Bardia Mohabbati, Marek Hatala, Dragan Gašević, Mohsen Asadi, and Marko Bošković. Development and configuration of service-oriented systems families. SAC '11, pages 1606–1613. ACM, 2011. 3, 5, 9, 33, 82, 184, 191, 216, 219, 223

[107] Hamid Reza Motahari-Nezhad and Claudio Bartolini. Next best step and expert recommendation for collaborative processes in it service management. In *Proceedings of the 9th International Conference on Business Process Management*, BPM'11, pages 50–61, Berlin, Heidelberg, 2011. Springer-Verlag. 229

[108] Dominic Muller, Joachim Herbst, Markus Hammori, and Manfred Reichert. It support for release management processes in the automotive industry. In *Proceedings of the 4th International Conference on Business Process Management*, BPM'06, pages 368–377, Berlin, Heidelberg, 2006. Springer-Verlag. 181

[109] Gunter Mussbacher, Daniel Amyot, João Araújo, and Ana Moreira. Transactions on aspect-oriented software development vii. chapter Requirements modeling with

the aspect-oriented user requirements notation (AoURN): a case study, pages 23–68. Springer-Verlag, Berlin, Heidelberg, 2010. 168

[110] Gunter Mussbacher, JoAo Arajo, Ana Moreira, and Daniel Amyot. Aourn-based modeling and analysis of software product lines. *Software Quality Journal*, pages 1–43. 10.1007/s11219-011-9153-8. 175, 176, 177

[111] John Mylopoulos, Lawrence Chung, and Eric Yu. From Object-oriented to Goal-oriented Requirements Analysis. *Commun. ACM*, 42:31–37, 1999. 142

[112] Tuan Nguyen, Alan Colman, and Jun Han. Modeling and managing variability in process-based service compositions. In *Proceedings of the 9th international conference on Service-Oriented Computing*, ICSOC'11, pages 404–420, Berlin, Heidelberg, 2011. Springer-Verlag. 9, 82, 220

[113] Khairul B.M. Noor. Case study: A strategic research methodology. *American Journal of Applied Sciences*, 5(11):1602–1604, March 2009. 8

[114] Alexis Ocampo and Martn Soto. Connecting the rationale for changes to the evolution of a process. In *Product-Focused Software Process Improvement*, volume 4589 of *Lecture Notes in Computer Science*, pages 160–174. Springer Berlin Heidelberg, 2007. 192, 219

[115] Ivana Ognjanovic, Dragan Gašević, Ebrahim Bagheri, and Mohsen Asadi. Conditional preferences in software stakeholders' judgments. In *Proceedings of the 2011 ACM Symposium on Applied Computing*, SAC '11, pages 683–690. ACM, 2011. 34

[116] Ivana Ognjanovic, Bardia Mohabbati, Dragan Gasevic, Ebrahim Bagheri, and Marko Boskovic. A metaheuristic approach for the configuration of business process families. In *IEEE SCC*, pages 25–32, 2012. 96, 104, 106, 107

[117] T. William Olle, A. A. Verrijn Stuart, and H. G. Sol. *Information Systems Design Methodologies; A Comparative Review: Proceedings of the IFIP WG 8.1 Working Conference on Comparative Review of Information Systems Design Methodologies, Noordwijkerhout, the Netherlands, 10-14 May 1982.* Elsevier Science Inc., New York, NY, USA, 1982. 49

[118] Object Management Group (OMG). Business Process Model and Notation (BPMN), Version 2.0. Technical Report formal/2011-01-03, 2011. 17

[119] Chun Ouyang, Marlon Dumas, Wil M. P. Van Der Aalst, Arthur H. M. Ter Hofstede, and Jan Mendling. From Business Process Models to Process-oriented Software Systems. *ACM Trans. Softw. Eng. Methodol.*, 19:2:1–2:37, August 2009. 17

[120] Joonseok Park, Mikyeong Moon, and Keunhyuk Yeom. Variability modeling to develop flexible service-oriented applications. *Journal of Systems Science and Systems Engineering*, 20:193–216, 2011. 10.1007/s11518-011-5164-z. 219

[121] Carlos Parra, Xavier Blanc, and Laurence Duchien. Context awareness for dynamic service-oriented product lines. In *Proceedings of the 13th International Software Product Line Conference*, SPLC '09, pages 131–140, Pittsburgh, PA, USA, 2009. Carnegie Mellon University. 228

[122] Carlos Parra, Anthony Cleve, Xavier Blanc, and Laurence Duchien. Feature-based composition of software architectures. In *Proceedings of the 4th European Conference on Software Architecture*, ECSA'10, pages 230–245, Berlin, Heidelberg, 2010. Springer-Verlag. 228

[123] Chris Peltz. Web Services Orchestration and Choreography. *Computer*, 36(10):46–52, October 2003. 17

[124] Marco Pistore, Marco Roveri, and Paolo Busetta. Requirements-driven verification of web services. *Electronic Notes in Theoretical Computer Science*, 105:95–108, 2004. 138

[125] Klaus Pohl, Günter Böckle, and Frank J. van der Linden. *Software Product Line Engineering: Foundations, Principles, and Techniques*. Springer, 2005. 8, 48, 61, 66, 84, 141, 183

[126] Klaus Pohl and Andreas Metzger. Variability management in software product line engineering. In *Proceedings of the 28th International Conference on Software Engineering*, ICSE '06, pages 1049–1050, New York, NY, USA, 2006. ACM. 47, 48, 49, 55, 56

[127] Alireza Pourshahid, Daniel Amyot, Liam Peyton, Sepideh Ghanavati, Pengfei Chen, Michael Weiss, and Alan J. Forster. Business Process Management with the User Requirements Notation. 9(4):269–316, 2009. 109

[128] Rick Rabiser, Paul Grunbacher, and Deepak Dhungana. Requirements for Product Derivation Support: Results from a Systematic Literature Review and an Expert Survey. *Information and Software Technology*, 52(3):324 – 346, 2010. 6, 36, 183, 194

[129] Jan C. Recker, Marta Indulska, Michael Rosemann, and Peter Green. How good is bpmn really? insights from theory and practice. In Jan Ljungberg and Magnus Andersson, editors, *14th European Conference on Information Systems*, Goeteborg, Sweden, 2006. 13

[130] Manfred Reichert and Barbara Weber. *Enabling Flexibility in Process-Aware Information Systems: Challenges, Methods, Technologies*. Springer Publishing Company, Incorporated, 2012. 1, 2

[131] I. Reinhartz-Berger, Pnina Soffer, and Arnon Sturm. Organizational Reference Models: Supporting an Adequate Design of Local Business Processes. *Int. J. of Business Process Integration and Management*, 4, 2009. 3, 33, 181, 191, 218, 219

[132] Iris Reinhartz-Berger, Pnina Soffer, and Amon Sturm. Extending the adaptability of reference models. *Trans. Sys. Man Cyber. Part A*, 40(5):1045–1056, September 2010. 3, 9, 181, 182, 184, 191, 218, 219

[133] Iris Reinhartz-Berger, Arnon Sturm, and Yair Wand. External variability of software: Classification and ontological foundations. In *Proceedings of the 30th International Conference on Conceptual Modeling*, ER'11, pages 275–289, Berlin, Heidelberg, 2011. Springer-Verlag. 59

[134] Andreas Rogge-Solti, Matthias Kunze, Ahmed Awad, and Mathias Weske. Business Process Configuration Wizard and Consistency Checker for BPMN 2.0. In Terry Halpin, Selmin Nurcan, John Krogstie, Pnina Soffer, Erik Proper, Rainer Schmidt, Ilia Bider, Wil Aalst, John Mylopoulos, Michael Rosemann, Michael J. Shaw, and Clemens Szyperski, editors, *Enterprise, Business-Process and Information Systems Modeling*, volume 81 of *Lecture Notes in Business Information Processing*, pages 231–245. Springer, 2011. 10, 185

[135] Marcello Rosa, Marlon Dumas, Arthur H. Hofstede, Jan Mendling, and Florian Gottschalk. Beyond control-flow: Extending business process configuration to roles and objects. In *Proceedings of the 27th International Conference on Conceptual Modeling*, ER '08, pages 199–215, Berlin, Heidelberg, 2008. Springer-Verlag. 186

[136] Marcello La Rosa, Wil M.P. van der Aalst, Marlon Dumas, and Fredrik P. Milani. Business process variability modeling : A survey, 2013. ACM Computing Surveys. 2, 4, 81, 85, 102, 103, 105, 228

[137] M. Rosemann and W. M. P. van der Aalst. A configurable reference modelling language. *Inf. Syst.*, 32(1):1–23, March 2007. 3, 4, 9, 29, 81, 102, 106, 107, 182, 184, 218

[138] N. Russel, A.H.M. ter Hofstede, W.M.P. van der Aalst, and N. Mulyar. Workflow Control-Flow Patterns: A Revised View. Technical report, BPM Center Report BPM-06-22, BPMcenter.org, 2006. 110, 120

[139] Thomas L. Saaty. How to make a decision: The analytic hierarchy process. *European Journal of Operational Research*, 48(1):9–26, September 1990. 35, 86

[140] S.W. Sadiq, M.E. Orlowska, and W. Sadiq. Specification and Validation of Process Constraints for Flexible Workflows. *Information Systems*, 30(5):349–378, 2005. 10, 130, 185, 187, 189, 211

[141] Emanuel Santos, Jaelson Castro, Juan Sánchez, and Oscar Pastor. A Goal-Oriented Approach for Variability in BPMN. In *Workshop em Engenharia de Requisitos, WER*, 2010. 110, 137

[142] Lidiane Santos, Lyrene Silva, and Thais Batista. On the Integration of the Feature Model and PL-AOVGraph. In *Proc. of the 2011 Int Ws on Early Aspects*, pages 31–36, 2011. 175, 177

[143] Klaus Schmid, Rick Rabiser, and Paul Grunbacher. A comparison of decision modeling approaches in product lines. In *Proceedings of the 5th Workshop on Variability Modeling of Software-Intensive Systems*, VaMoS '11, pages 119–126. ACM, 2011. 105

[144] Arnd Schnieders and Frank Puhlmann. Variability mechanisms in e-business process families. In *Int. Conf. on Business Information Systems (BIS)*, pages 583–601, 2006. 219

[145] P.-Y. Schobbens, P. Heymans, and J.-C. Trigaux. Feature diagrams: A survey and a formal semantics. In *Requirements Engineering, 14th IEEE International Conference*, pages 139 –148, sept. 2006. 40, 146, 197

[146] Roberto Sebastiani, Paolo Giorgini, and John Mylopoulos. Simple and minimum-cost satisfiability for goal models. In Anne Persson and Janis Stirna, editors, *Advanced Information Systems Engineering*, volume 3084 of *Lecture Notes in Computer Science*, pages 20–35. Springer Berlin Heidelberg, 2004.

[147] Sergio Segura, José A. Galindo, David Benavides, José A. Parejo, and Antonio Ruiz-Cortés. Betty: benchmarking and testing on the automated analysis of feature models. In *Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems*, VaMoS '12, pages 63–71. ACM, 2012. 97

[148] Norbert Siegmund, Marko Rosenmüller, Martin Kuhlemann, Christian Kästner, Sven Apel, and Gunter Saake. SPL Conqueror: Toward optimization of non-functional properties in software product lines. *Software Quality Journal*, pages 1–31, June 2011. 21, 34, 85, 103, 105, 106

[149] Carla T. L. L. Silva, Fernanda M. R. Alencar, João Araújo, Ana Moreira, and Jaelson Brelaz de Castro. Tailoring an aspectual goal-oriented approach to model features. In *SEKE*, pages 472–477, 2008. 174, 175, 177

[150] Marco Sinnema, Sybren Deelstra, Jos Nijhuis, and Jan Bosch. Covamof: A framework for modeling variability in software product families. In *In Proceedings of the Third International Software Product Line Conference (SPLC*, 2004. 3, 181

[151] Dag I. K. Sjoberg, Jo E. Hannay, Ove Hansen, Vigdis By Kampenes, Amela Karahasanovic, Nils-Kristian Liborg, and Anette C. Rekdal. A survey of controlled experiments in software engineering. *IEEE Trans. Softw. Eng.*, 31(9):733–753, September 2005. 8

[152] P. Soffer, I. R-Berger, and A.. Sturm. Facilitating Reuse by Specialization of Reference Models for Business Process Design. In *CAiSE 07 Workshop*, pages 339–347, 2007. 33, 191

[153] Pnina Soffer and Yair Wand. Goal-Driven Analysis of Process Model Validity. In *Advanced Information Systems Engineering*, volume 3084 of *LNCS*, pages 229–319. Springer, 2004. 138

[154] Pnina Soffer, Yair Wand, and Maya Kaner. Semantic Analysis of Flow Patterns in Business Process Modeling. In *5th International Conference on Business Process Management (BPM)*, volume 4714 of *LNCS*, pages 400–407. Springer, 2007. 138

[155] Samaneh Soltani, Mohsen Asadi, Dragan Gasevic, Marek Hatala, and Ebrahim Bagheri. Automated planning for feature model configuration based on functional and non-functional requirements. In *SPLC (1)*, pages 56–65, 2012. 5, 9, 21, 82, 86, 88, 96, 104, 105, 106, 184, 193, 216, 223

[156] Ian Sommerville, Ian Sommerville, Pete Sawyer, and Pete Sawyer. Viewpoints: principles, problems and a practical approach to requirements engineering. *Annals of Software Engineering*, 3:101–130, 1997. 97, 102

[157] Martin Soto, Kaiserslautern Fraunhofer, Dieter Rombach, Peter Liggesmeyer, and Frank Bomarius. *The DeltaProcess Approach to Systematic Software Process Change Management*. Fraunhofer IRB Verlag, Germany, 2010. 219

[158] Martín Soto and Jürgen Münch. Using model comparison to maintain model-to-standard compliance. In *Proceedings of the 2008 International Workshop on Comparison and Versioning of Software Models*, CVSM '08, pages 35–40, New York, NY, USA, 2008. ACM. 182

[159] Martín Soto and Jürgen Münch. Using model comparison to maintain model-to-standard compliance. In *Proceedings of the 2008 International Workshop on Comparison and Versioning of Software Models*, CVSM '08, pages 35–40, New York, NY, USA, 2008. ACM. 219

[160] Alfred Tarski. *The Semantic Conception of Truth and the Foundations of Semantics*. Philosophy and Phenomenological Research, 1944. 111

[161] T. Ternite. Process lines: A product line approach designed for process model development. In *Software Engineering and Advanced Applications, 2009. SEAA '09. 35th Euromicro Conference on*, pages 173–180, 2009. 3, 9, 182, 184

[162] Sahil Thaker, Don S. Batory, David Kitchin, and William R. Cook. Safe Composition of Product Lines. In *Proc. of the GPCE '06*, pages 95–104, 2007. 178

[163] Oliver Thomas. Understanding the term reference model in information systems research: History, literature analysis and explanation. In *Proceedings of the Third International Conference on Business Process Management*, BPM'05, pages 484–496, Berlin, Heidelberg, 2006. Springer-Verlag. 29

[164] Thomas Thum, Don Batory, and Christian Kastner. Reasoning about edits to feature models. In *Proceedings of the 31st International Conference on Software Engineering*, ICSE '09, pages 254–264, Washington, DC, USA, 2009. IEEE Computer Society. 212, 217

[165] K. Uno, S. Hayashi, and M. Saeki. Constructing feature models using goal-oriented analysis. In *Quality Software, 2009. QSIC '09. 9th International Conference on*, pages 412–417, Aug 2009. 66

[166] Wil M. P. van der Aalst, Marlon Dumas, Florian Gottschalk, Arthur H. M. ter Hofstede, Marcello La Rosa, and Jan Mendling. Preserving correctness during business process model configuration. *Form. Asp. Comput.*, 22(3-4):459–482, May 2010. 3, 139, 181, 218

[167] Wil M. P. van der Aalst and Arthur H. M. ter Hofstede. YAWL: Yet Another Workflow Language. *Information Systems*, 30(4):245–275, 2005. 17

[168] Wil M.P. van der Aalst, Niels Lohmann, and Marcello La Rosa. Ensuring correctness during process configuration via partner synthesis. *Inf. Syst.*, 37(6):574 – 592, 2012. 10, 185, 218

[169] W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow Patterns. In *Distributed and Parallel Databases*, 2003. 17, 29, 120

[170] F. Van Der Linden, K. Schmid, and E. Rommes. *Software product lines in action: the best industrial practice in product line engineering*. Springer-Verlag New York Inc, 2007. 3, 181

[171] T. Van Der Storm. Generic Feature-based Software Composition. In *6th Int. Conference on Software Composition*, volume 4829 of *LNCS*, pages 66–80, 2007. 178

[172] J. van Gurp, J. Bosch, and M. Svahnberg. On the notion of variability in software product lines. In *Software Architecture, 2001. Proceedings. Working IEEE/IFIP Conference on*, pages 45–54, 2001. 26

[173] Axel van Lamsweerde. *Requirements Engineering: From System Goals to UML Models to Software Specifications*. Wiley, 2009. 13, 148

[174] Jussi Vanhatalo, Hagen Volzer, and Frank Leymann. Faster and More Focused Control-Flow Analysis for Business Process Models Through SESE Decomposition. In *Service-Oriented Computing - ICSOC*, volume 4749 of *LNCS*, pages 43–55. Springer, 2007. 18

[175] Patrícia Varela, João Araújo, Isabel Brito, and Ana Moreira. Aspect-oriented analysis for software product lines requirements engineering. In *Proceedings of the 2011 ACM Symposium on Applied Computing*, SAC '11, pages 667–674, New York, NY, USA, 2011. ACM. 176

[176] Yair Wand and Ron Weber. On the deep structure of information systems. *Information Systems Journal*, 5(3):203–223, 1995. 12, 13, 47, 48, 53, 55

[177] H.H. Wang, Y.F. Li, J. Sun, H. Zhang, and J. Pan. Verifying Feature Models using OWL. *J. of Web Semantics*, 5(2):117–129, 2007. 157

[178] Barbara Weber, Manfred Reichert, and Stefanie Rinderle-Ma. Change patterns and change support features - enhancing flexibility in process-aware information systems. *Data Knowl. Eng.*, 66(3):438–466, September 2008. 191

[179] Ron Weber and Yanchun Zhang. An analytical evaluation of niam's grammar for conceptual schema diagrams. *Information Systems Journal*, 6(2):147–170, 1996. 13

[180] Matthias Weidlich, Mathias Weske, and Jan Mendling. Change propagation in process models using behavioural profiles. In *Proceedings of the 2009 IEEE International Conference on Services Computing*, SCC '09, pages 33–40, Washington, DC, USA, 2009. IEEE Computer Society. 190, 212

[181] David M. Weiss and Chi Tau Robert Lai. *Software Product-line Engineering: A Family-based Software Development Process*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999. 4, 27, 49, 145, 182

[182] Jules White, David Benavides, Douglas C. Schmidt, Pablo Trinidad, Brian Dougherty, and Antonio Ruiz Cortés. Automated diagnosis of feature model configurations. *Journal of Systems and Software*, 83(7):1094–1107, 2010. 167

[183] Jules White, Brian Dougherty, and Douglas C. Schmidt. Selecting highly optimal architectural feature sets with filtered cartesian flattening. *Journal of Systems and Software*, 82(8):1268–1284, August 2009. 96, 104, 105, 106

[184] Jules White, Brian Dougherty, Doulas C. Schmidt, and David Benavides. Automated reasoning for multi-step feature model configuration problems. In *Proceedings of the 13th International Software Product Line Conference*, SPLC '09, pages 11–20. IEEE, 2009. 103, 105, 106

[185] S.A. White and D. Miers. *BPMN Modeling and Reference Guide*. Future Strategies Inc., 2008. 19

[186] Eric Yu, Paolo Giorgin, Neil Maiden, and John Mylopoulos, editors. *Social Modeling for Requirements Engineering*. MIT, 2011. 13, 144, 148

[187] Eric S. K. Yu. Towards modeling and reasoning support for early-phase requirements engineering. In *Proceedings of the 3rd IEEE International Symposium on Requirements Engineering*, RE '97, pages 226–, Washington, DC, USA, 1997. IEEE Computer Society. 73

[188] Eric S. K. Yu and John Mylopoulos. From e-r to "a-r" - modelling strategic actor relationships for business process reengineering. In *Proceedings of the13th International Conference on the Entity-Relationship Approach*, ER '94, pages 548–565, London, UK, UK, 1994. Springer-Verlag. 62

[189] Eric S. K. Yu and John Mylopoulos. Understanding "Why" in Software Process Modelling, Analysis and Design. In *Proceedings of the 16th international conference on Software engineering*, ICSE '94, pages 159–168, Los Alamitos, CA, USA, 1994. IEEE Computer Society Press. 14, 119

[190] Yijun Yu, Julio Cesar Sampaio do Prado Leite, Alexei Lapouchnian, and John Mylopoulos. Configuring Features with Stakeholder Goals. In *Proceedings of the 2008 ACM Symposium on Applied Computing*, SAC '08, pages 645–649, New York, NY, USA, 2008. ACM. 63, 64, 74, 76, 142, 151, 174, 177

[191] Yijun Yu, Alexei Lapouchnian, Sotirios Liaskos, John Mylopoulos, and Julio Leite. From Goals to High-Variability Software Design. In *Foundations of Intelligent Systems*, volume 4994 of *LCNS*, pages 1–16. Springer, 2008. 62, 63, 66, 76, 142, 145, 174

[192] Marvin V. Zelkowitz. An update to experimental models for validating computer technology. *Journal of System and Software*, 82(3):373–376, March 2009. 8

[193] Guoheng Zhang, Huilin Ye, and Yuqing Lin. Quality attribute modeling and quality aware product configuration in software product lines. *Software Quality Journal*, pages 1–37, 2013. 104, 105, 106

[194] Steffen Zschaler, Pablo Sánchez, João Santos, Mauricio Alférez, Awais Rashid, Lidia Fuentes, Ana Moreira, João Araújo, and Uirá Kulesza. Vml* – a family of languages for variability management in software product lines. In *Proceedings of the Second International Conference on Software Language Engineering*, SLE'09, pages 82–102, Berlin, Heidelberg, 2010. Springer-Verlag. 77