Modeling and Validation of Business Process Families

Gerd Gröner[a], Marko Bošković[b], Fernando Silva Parreiras[c], Dragan Gašević[d]

[a] *WeST Institute, University of Koblenz-Landau, Germany*
[b] *Research Studios Austria, Austria*
[c] *FUMEC University, Brazil*
[d] *Athabasca University, Canada*

Abstract

Process modeling is an expensive task that needs to encompass requirements of different stakeholders, assure compliance with different standards, and enable the flexible adaptivity to newly emerging requirements in today's dynamic global market. Identifying reusability of process models is a promising direction towards reducing the costs of process modeling. Recent research has offered several solutions. Such solutions promote effective and formally sound methods for variability modeling and configuration management. However, ensuring behavioral validity of reused process models with respect to the original process models (often referred to as reference process models) is still an open research challenge. To address this challenge, in this paper, we propose the notion of business process families by building upon the well-known software engineering discipline – software product line engineering. Business process families comprise (i) a variability modeling perspective, (ii) a process model template (or reference model), and (iii) mappings between (i) and (ii). For business process families, we propose a correct validation algorithm ensuring that each member of a business process family adheres to the core intended behavior that is specified in the process model template. The proposed validation approach is based on the use of Description Logics, variability is represented by using the well-known Feature Models and behavior of process models is considered in terms of control flow patterns. The paper also reports on the experience gained in two external trial cases and results obtained by measuring the tractability of the implementation of the proposed validation approach.

Keywords: business process families, control flow relations, validation, process model variability, process model configuration

1. Introduction

Process modeling is an expensive task [1], and the more detailed and fine-grained a process is the more effort is required to build the corresponding process model. This can be attributed to several (crosscutting) reasons. First, no organization is an isolated island in today's global economy. Rather, processes in individual organizations are typically based on collaboration with several other (partner) organizations. Therefore, any parts of processes prone to often changes in such a collaborative environment should carefully be considered and reflected in the models. Next, the process modeling task typically requires the involvement of stakeholders with different backgrounds (e.g., accounting, sales, security, information technology, or software development) who need to bring together critical (and sometimes conflicting) perspectives to processes being modeled. Finally, process models need to be adaptive in order to be able to serve to stakeholders who might not have the exact same, yet, rather similar requirements.

Recently, process modeling research and practice have been looking for novel methods and techniques that can help to tackle the complexity of the process modeling task. Obviously, process reusability [2] is one of the most desirable attributes, which process models need to have. Reusability of processes can reduce the costs of the involvement of different stakeholders all over in different situations. Reusability of process models can also scale up the development of software that enact the processes. Reusability can also increase the quality of processes by reducing the probability of potential defects, assure compliance to certain standards the processes need to adhere to, and ease change propagation.

Capturing and systematically managing variability in process models is one of the main prerequisites for their effective and scalable reusability [3]. By variability capturing we primarily refer to the representation of individual variation points allowing stakeholders to make their individual choices when reusing a process model. However, such variation points are typically not completely independent from the rest of a process model being reused. Often, choices made about some variability points have direct implications on other variation points (e.g., deciding

Email addresses: groener@uni-koblenz.de (Gerd Gröner), boskovic@researchstudios.at (Marko Bošković), fernando.parreiras@fumec.br (Fernando Silva Parreiras), dragang@athabascau.ca (Dragan Gašević)

to include a credit card payment option in a process has a direct implication about what additional activities must be included in order to assure security of such payments). This directly indicates that for an effective variability capturing and management in business processes, there is a need to have not only a representation of variability points, but also algorithms that can guide the stakeholders when making their choices [4].

Commonly, existing work considers the task of process model reusability in terms of configurable reference models [4, 5, 6], where a reference process model contains variation points. These variation points offer choices like the selection of a particular branch [4] and the hiding or blocking of certain activities [5, 6]. Instead of building several related business process models from scratch, the reference process models are narrowed down towards an individualized business process model. In order to perform the configuration of process models, such approaches require from stakeholders knowledge of process modeling and how elements of a process model depend on each other. To make the configuration process more understandable, the questionnaire-based approach [7] guides the stakeholders by asking questions of relevance to the particular choices they need to make. However, this approach does not offer an automated solution, which can ensure that for each possible set of valid choices, the stakeholders can make, there is a business process consistent with the behavior specified in the original reference business process. This challenge is exactly tackled in this paper.

We propose the notion of *Business Process Families*, by building upon the principles of the well-established software engineering discipline – software product line engineering (SPLE) [8]. Having the variability management in its very core, SPLE offers methods and techniques for variability modeling and configuration management in software artifacts. To this end, business process families comprise three critical elements (based on [9]): (i) a variability perspective that abstracts from the business logic of a process model; (ii) a process model template that specifies an intended core behavior; and (iii) mappings between the variability perspective and the process model template, so that configuration of process model templates can be driven by the choices made in the variability perspective. Our approach, presented in this paper, ensures that each member of the business process family, which can be derived from the configuration space of the variability perspective, adheres to the core intended behavior that is specified by the process model template. The intended behavior is treated in terms of control flow patterns [10]. Our proposed approach first lifts all the three elements of business process families to a common representation defined by using Description Logics [11]. It then offers a correct validation algorithm, which is experimented in two external trial cases and checked for its tractability.

We describe our approach as follows: in Section 2, we contextualize the problem and present the key challenges. In Section 3, we present foundations of business process families, followed by an analysis of inconsistencies between feature and control flow relations in Section 4. We lift relations between features and activities into a common Description Logics (DL) knowledge base in Section 5. In Section 6, we detail our framework for validating business process families. The validation is based on modeling and reasoning in Description Logics. Section 7 demonstrates the correctness of the proposed configuration and validation approach, followed by an evaluation in Section 8, which demonstrates the tractability within a proof-of-concept implementation. Section 9 investigates related work. Finally, we conclude the paper with Section 10.

2. Motivation and Problem Description

This section highlights the application context of our approach, followed by a description of the proposed idea and the investigated problem.

2.1. Context

Business processes within organizations are implemented and automated by process-aware information systems (PAIS) [12]. Their implementation relies on the specification of a process, described by a process model. As business processes are usually designed and optimized within individual organizations, vendors of process-aware information systems need to build their systems for several business processes that are quite related to each other, but still different in some parts.

Instead of designing each business process and their corresponding PAIS from scratch, it is a promising technique to start with a process model template (or reference process model) and then customize such a model for individual stakeholder's needs.

Individual stakeholders are not necessarily familiar with business process modeling and management. Thus, it is different for them to derive a particular process model for an intended system-to-be based on their needs.

2.2. Basic Idea

The problem of handling multiple (quite related) models is studied for several years in the realm of software product line engineering (SPLE) [8]. A product line or family describes commonality and variability of members of a family such that they abstract from the business logic and consider a system in terms of its features and relationships between features. This rather abstract perspective guides individual stakeholders in designing a final member of a family.

In this paper, we propose to adopt these principles from SPLE to handle business process families. In particular, we use feature models [13], the most commonly used variability modeling technique in SPLE, as a guidance for stakeholders to configure and customize an individual process model (and therefore the corresponding

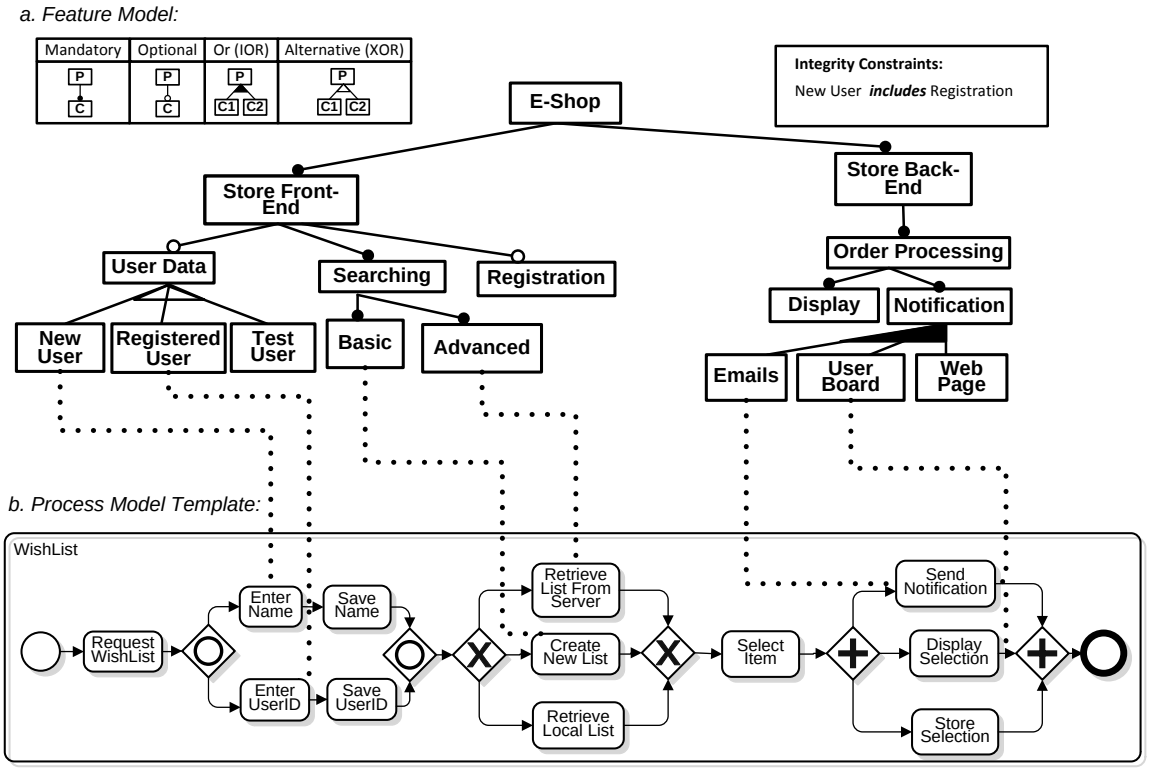

Figure 1: A business process family that consists of a feature model, a process model template and mappings between features of the feature model and activities of the process model template

PAIS). Stakeholders can decide how the process and the corresponding information system looks like based on the feature perspective, without having dedicated knowledge about process modeling and management.

The basic idea is depicted in Figure 1, where a feature model (Figure 1.a) captures the variability and offers stakeholders customization choices in well-known formalisms that describes configuration options. A process model template is depicted in the lower part (Figure 1.b). Mappings describe the implementation of certain features by a concrete activity of the process model template. All three artifacts, i.e., feature models, process model templates and mappings are designed by experts, while stakeholders use the feature oriented view to customize process models according to their needs.

A particular selection of features (also called a configuration) regulates which elements of a process model template stay and which are removed. For example, in Figure 1, the selection of the feature *New User* regulates whether the activity *Enter Name* is part of a particular business process model or not. We understand a configuration according to the description given in [9], as a selection of a subset of elements from a reference model (i.e., from a process model template).

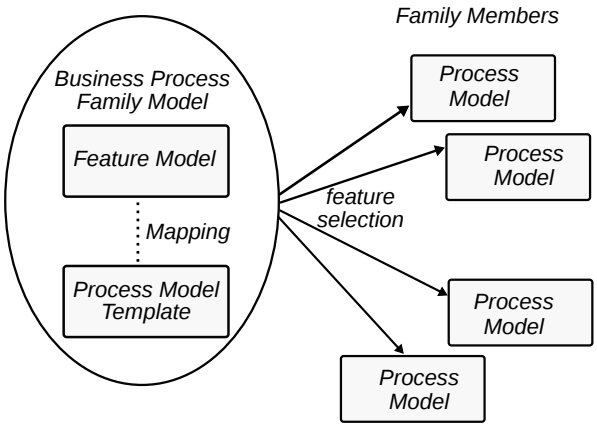

Figure 2: Members of a business process family are derived by feature selection from a business process family model

2.3. Investigated Problem

As depicted in Figure 1, a business process family is represented by a feature model, a process model template and mappings between features and activities. A particular member of the family, i.e., a process model is derived by selecting features of the feature model. This procedure is illustrated in Figure 2.

In the scope of this paper, we consider business process

models at the modeling perspective, i.e., we analyze and configure business process models at design time (modeling time), and we do not take into account run time aspects. A particular configured business process model is still at the modeling perspective, but the set of permitted executions might be restricted compared to the original process model template.

Since the configuration of particular business process models is based on feature selections, the business process model is influenced by two kinds of constraints or relations: (i) The selection of features might depend on the selection of other features according to feature relations. This dependency between features is carried to the corresponding activities of the business process model by mappings. (ii) Activities of the business process model depend on other activities regarding to the control flow relations.

Business process models that are configured according to feature selections (and therefore according to the feature relationships) do not necessarily satisfy the control flow relations of the given process model template. For instance, activity *Send Notification* is mapped to the feature *Emails* that characterizes a notification via email. The sibling activity *Display Selection* is mapped to the feature *User Board*, referring to a notification via the user board. Thus, a stakeholder could build a valid feature selection (regarding to feature relations) that contains only one of these features, leading to a business process model that only contains one of these activities (*Send Notification* or *Display Selection*), while both require each other according to their control flow relations.

We suggest to address the comparison of relationships in the process model template with the relationships of the corresponding mapped features in the feature model in order to ensure that each valid feature selection leads to a business process model that adheres to the control flow relations that are given by the process model template.

3. Business Process Families

This section formally defines business process families and their three constitutive elements.

3.1. Feature Models

The most common means for representing variability are feature models. A feature model is a tree-like structure whose nodes are features of the target software product line [1] [14]. It describes valid combinations of features. According to the distinction of Metzger et al. [15], variability used in our case can be considered as software variability, i.e., the ability to customize a system in a particular case.

In a feature model, there exists three kinds of relationships between features: 1) *parent-child relationships*; 2) *group relationships*; and 3) *cross-tree constraints* also known as *integrity constraints*.

A formal description of feature models is given in Definition 1. Parent child relationships are *mandatory* and *optional*. A *mandatory* parent-child relationship specifies that if a parent feature is selected in a certain configuration, its mandatory child feature has to be selected, too (e.g., *Store Front-End* and *Store Back-End* are both mandatory children of *E-Shop*). This is depicted by the relation $\mathcal{F}_M$ in Definition 1, where the set $\mathcal{F}$ refers to the parent features and the power set $\mathcal{P}(\mathcal{F})$ denotes the set of mandatory child features. An *alternative feature group*, or *xor feature group*, (e.g., *Basic* and *Advanced*), specifies that when their parent feature is selected, *exactly one* of the members of the group can be selected. Finally, an *or group* (e.g., *Emails*, *User Board* and *Web Page*) defines a set of features from which at least one has to be selected.

Integrity or cross-tree constraints between features are the ones that cannot be captured by the tree structure of feature diagrams. Generally, two cross-tree constraints exist, namely *includes* and *excludes*. *Includes* means that if an including feature is in a configuration, the included feature has to be as well (e.g., *New User* and *Registration*). *Excludes* is the opposite to *includes*.

Definition 1 (Feature Model). *A feature model* $\Phi = \langle \mathcal{F}, \mathcal{F}_M, \mathcal{F}_O, \mathcal{F}_{IOR}, \mathcal{F}_{XOR}, \mathcal{F}_{incl}, \mathcal{F}_{excl} \rangle$ *is a tree structure that consists of features* $\mathcal{F}$. $\mathcal{F}_M \subseteq \mathcal{F} \times \mathcal{P}(\mathcal{F})$ *and* $\mathcal{F}_O \subseteq \mathcal{F} \times \mathcal{P}(\mathcal{F})$ *are sets of parent features and the set of all their mandatory and optional child features, respectively.* $\mathcal{F}_{IOR} \subseteq \mathcal{F} \times \mathcal{P}(\mathcal{F})$ *and* $\mathcal{F}_{XOR} \subseteq \mathcal{F} \times \mathcal{P}(\mathcal{F})$ *are sets of pairs of child features and their common parent feature. Finally,* $\mathcal{F}_{incl} \subseteq \mathcal{F} \times \mathcal{F}$ *and* $\mathcal{F}_{excl} \subseteq \mathcal{F} \times \mathcal{F}$ *are sets of includes and excludes relationships (integrity constraints).*

3.2. Process Model Template

A process model template specifies the business logic perspective. In Figure 1, we use BPMN to specify a process model template. Such a template typically consists of process patterns like subprocesses, activities and gateways. We understand a process model template as a directed graph, according to Definition 2.

Definition 2 (Process Model Template). *A process model template is a directed graph* $\Omega_G = \langle \mathcal{V}, \mathcal{E} \rangle$ *with* $\mathcal{V}$ *denoting a finite set of vertices and* $\mathcal{E}$ *the edges between the vertices describing the flow of the process.*

- $\mathcal{V}$ *consists of a set of activities* $\mathcal{A}$ *and a disjoint set of gateways (or control vertices)* $\mathcal{G}$ *and exactly one start and one end vertex* ($\mathcal{V} = \mathcal{A} \uplus \mathcal{G}$).
- *Activities have exactly one incoming and one outgoing edge.*
- *Gateways have either exactly one incoming and at least two outgoing edges or exactly one outgoing edge and at least two incoming edges. The first ones refer*

[1] In this paper, we will use product line and software family interchangeably, even though one can argue that they can not be considered synonymous.

to opening gateways (decision / fork gateways) and the second ones to closing gateways (merge / join gateways).

According to the BPMN specification [16], activities ($\mathcal{A}$) are either atomic activities or sub-processes (decomposable activities).

We focus in our work on structured process models for two reasons. Firstly, for the class of structured models, structural constraints coincide with behavioral constraints (see the work on behavioral profiles that are derived from process structure trees [17]). Secondly, there are techniques to derive structured models for a broad class of unstructured models [18]. Structured models require that for each opening gateway there is a closing gateway, i.e., they create valid single-entry-single-exit (SESE) fragments.

Definition 3 extends the definition of process model template Ω_G towards structured process model templates, where for each opening gateway there is a corresponding closing gateway. Thus, a process model can be decomposed into SESE fragments.

Definition 3 (Structured Process Model Template). *A structured process model (or structured process model template) is triple $\Omega = \langle \mathcal{V}, \mathcal{E}, \mathcal{S}, \mathcal{D} \rangle$, which is based on a process model $\Omega_G = \langle \mathcal{V}, \mathcal{E} \rangle$. The set $\mathcal{S}$ explicitly represents SESE fragments of Ω, where each SESE fragment $S \in \mathcal{S}$ with $S \in \mathcal{V} \times \mathcal{V} \times \mathcal{P}(\mathcal{B})$ has an entry and an exit node and a set of branches ($B \in \mathcal{B}$). $\mathcal{D}$ is a set of decomposition relations $D \in \mathcal{D}$, where $D \subseteq \mathcal{A} \times \mathcal{A}$, denoting a subprocess that contains internal activities.*

3.3. Mappings

Mappings (Definition 4) connect features $\mathcal{F}$ and activities $\mathcal{A}$ of the structured process model template Ω that implement the business logic of particular configurations. For example, every configuration that contains the feature *Basic*, contains the activity *Create New List*. All activities of the process model template that are not mapped to any feature are contained in every business process model. Furthermore, a feature can be mapped to multiple activities and likewise an activity might realize multiple features. Formally, mappings are defined as follows.

Definition 4 (Mapping). *For a feature model $\Phi = \langle \mathcal{F}, \mathcal{F}_M, \mathcal{F}_O, \mathcal{F}_{IOR}, \mathcal{F}_{XOR}, \mathcal{F}_{incl}, \mathcal{F}_{excl} \rangle$ and a structured process model template $\Omega = \langle \mathcal{V}, \mathcal{E}, \mathcal{S}, \mathcal{D} \rangle$ with $\mathcal{A} \subseteq \mathcal{V}$, a mapping M is a relation $M \subseteq \mathcal{F} \times \mathcal{A}$, that is defined as $M := \{(f, a) : f \in \mathcal{F} \wedge a \in \mathcal{A}\}$.*

3.4. Business Process Families

The combination of feature models Φ, structured process model templates Ω and mappings M constitute a business process family model.

Definition 5 (Business Process Family Model). *For a feature model $\Phi = \langle \mathcal{F}, \mathcal{F}_M, \mathcal{F}_O, \mathcal{F}_{IOR}, \mathcal{F}_{XOR}, \mathcal{F}_{incl}, \mathcal{F}_{excl} \rangle$, a structured process model template $\Omega = \langle \mathcal{V}, \mathcal{E}, \mathcal{S}, \mathcal{D} \rangle$ with $\mathcal{A} \subseteq \mathcal{V}$ and Mappings $M \subseteq \mathcal{F} \times \mathcal{A}$ between both models a business process family model $\Psi = \langle \Phi, \Omega, M \rangle$ is the combination of them.*

4. Inconsistencies in Business Process Families

The feature model describes variability relationships among selectable features of a system-to-be, the process model template represents control flow relationships that are required to be satisfied by all family members. Control flow relations are imposed by control flow patterns, as described in [10, 19, 20]. In business process family models, mappings combine features and activities. Thus, features and activities might be affected by different relationships.

4.1. Mapping Influence on Control Flow Relations

In our representation formalism for business process families, the intention is that the selection of a feature determines whether the corresponding mapped activity is part of the particular business process model or not. Obviously, the resulting business process model is built according to feature relations, but they do not necessarily coincide with the control flow relations given by the template.

As already outlined, our approach aims at detecting inconsistencies between feature relationships and control flow relationships, based on a comparison of the different relationships of mapped features and activities. While the kinds of relations in the feature model are explicitly given by feature groups (*and*, *ior* and *xor*), as well as by integrity (cross-tree) constraints, we have to identify the kinds of relationships between activities in the business process model.

Following this line of argumentation, the focus in the remainder of this section is on an analysis of control flow patterns as best practise building blocks for process models. For each pattern, we discuss which kind of control flow relationship is imposed to activities, and whether this relationship is influenced by feature relationships if activities are mapped to features.

4.1.1. Basic Control Flow Patterns

The basic control flow patterns provide fundamental modeling primitives for business processes. They are summarized in [10, 19, 20] and are imposed by business process modeling languages like BPMN.

Sequence. The sequence is a basic construct to describe the sequential routing of activities. An activity can be executed after the execution of its predecessor activity. Sequences restrict the ordering of activities, but the existence and co-existence is not required by a sequential ordering (cf. [21]). Thus, in our configuration context, no feature relation influences the ordering of activities and we can neglect sequences in the remainder of this work.

Parallel Branching. If activities occur in parallel branches, they have to be executed commonly, i. e., if an activity of the first branch is executed, then also the activity of the second branch is executed.

Parallel branchings are started by parallel opening gateways, while they can be closed by several allowed types of closing gateway. In general, a parallel branching is closed by a closing parallel gateway. In our configuration context, activities in all these parallel sibling branches represent a conjunctive relation in the logical sense.

Exclusive Branching. Activities that appear in exclusive sibling branches are supposed to be exclusive, i. e., it is not allowed to execute activities from alternative branches in a process execution.

Subprocess. BPMN models facilitate decomposition of activities into subprocesses. The configuration of particular business process models allows for a certain degree of freedom as to whether an activity within a subprocess is selected. However, if an internal activity is part of a particular business process model its corresponding subprocess have to be part of this business process model, too.

4.1.2. Advanced Branching and Synchronization Patterns

Advanced branching and synchronization pattern offer constructs for a more concise representation of process models, but they can be reduced to equivalent constructs that only use basic patterns.

Inclusive Branching. Inclusive branches offer execution choices in which at least one branch has to be executed. In contrast to exclusive branching, multiple sibling branches can be executed. This kind of constraint is imposed for all activities within sibling branches in every particular configuration of a business process model.

Discriminator and N out of M Join. The discriminator and the n out of m join are special joins that facilitate a kind of race situation between different incoming branches [16]. The successors of the gateway are enabled for execution if at least one predecessor is executed (in case of a discriminator) or at least n (in case of an n out of m join) is/are executed.

In our case, as we assume a correct process model template and we remain at the modeling perspective. The crucial aspect of these complex joins for the configuration is to ensure the minimum number of predecessors in order to stick to the meaning that is given by a discriminator and n out of m join.

4.1.3. Structural Patterns

The focus of our work is on the structural description of processes using process models. Structural patterns allow further specification of the process behavior in a process models. They capture the structure of loops and explicit statements about the termination of process executions.

Arbitrary Cycles. In contrast to structural cycles, arbitrary cycles (or unstructured loops) describe loops with multiple entry or exit points. With regard to mapping inconsistencies, there is no distinction between structural and arbitrary cycles. Arbitrary cycles can be reduced to structured cycles, as demonstrated in [10]. Cycles are composed of branchings, where some of the branches return to the part of the process that needs to be repeated. Therefore, we treat branches of arbitrary cycles like inclusive branchings in our approach.

Implicit Termination. An explicit termination is represented in a process model by an *end* event (i. e., an *end* node in the process model). Implicit termination states that each subprocess should terminate if there is no further activity to execute, i. e., without an explicit *end* node after the last activity of a subprocess.

As the particular purpose of our validation is on the relations between activities and implicit termination does not impose any constraints or relations between activities, implicit termination will not cause any inconsistency during the process model configuration. However, we assume that if the last activity of a (sub-) process is removed and there is no explicit *end* event, the predecessor of the removed element is considered as the last activity with respect to implicit termination.

4.1.4. Multiple Instances (MI) Patterns

Multiple instances patterns describe the situation when there are multiple instances of an activity active. This is rather an implementation oriented aspect for the process management system. BPMN supports three multiple instances pattern: (i) *MI without synchronization* allows for an activity to have multiple instances within one process execution; (ii) *MI with a priory design time knowledge* provides the possibility to have a fixed number of instances of an activity within one process, whereby the number is already known at design time; while the pattern (iii) *MI with a priory runtime knowledge* enables a variable number of instances, but the number is known at runtime. In BPMN, these patterns are represented as annotations of activities. As every runtime instance of a process is a copy of the process depicted in a model, therefore, if the process model does not have inconsistencies, neither its instances will have.

4.1.5. State-based Patterns

Deferred Choice. This pattern is related to an exclusive branching, but the choice of the exclusive opening gateway is made explicitly, i. e., it is not based on control flow decisions. Once one branch of a deferred choice is executed (i. e., at least the first activity of this branch is executed), the remaining sibling branches are withdrawn for execution. This pattern is rather relevant for the process execution.

Interleaved parallel routing. This pattern enables a set of activities to be executed in an arbitrary order. The execution order of these activities is decided at runtime, but at any time only one activity can be executed. The configuration at design time does not affect this pattern, as the selection is done at runtime.

Milestone. This pattern is an explicit statement that an activity cannot be executed unless a certain specified state is reached. It is a dependency between between activities, but it is rather a test of certain (state) conditions than a control flow dependency. Thus, this pattern is not affected in our configuration.

4.1.6. Cancellation Patterns

The cancellation patterns enable the cancellation of an activity (cancel activity) or of a complete process instance (cancel case). In both patterns, an activity (or a process instance) that is enabled can be withdrawn before the execution. This pattern is meaningful if there are multiple instances of an activity (or of a process) and in case one instance has executed a certain activity (or a certain part of the process) further executions have to be avoided. As both cancellation patterns are relevant for process executions and not in the modeling perspective, they are not affected within our feature-based configuration approach.

4.2. Relationships and Inconsistencies

Mappings between features and activities connect different relationships of the mapped elements. Usually, domain experts map features to activities of a process model template to link variability and control flow relations from different views. However, the relationships (or constraints) of the mapped features and activities do not necessarily coincide with each other, or they might be even contradictory. From a logical point of view, we deal with basic logical connectives, as described in Definition 6.

Definition 6 (Relationships between Elements). *Control flow relations CR describe relationships between activities $A_j \in \mathcal{A}$ (of a structured process model template Ω) and feature relations FR describe relationships between features $F_i \in \mathcal{F}$ (of a feature model Φ). We refer to elements in these relationships in the terms of predicate logics: (1) conjunctive connected elements are called conjuncts, (ii) disjunctive (inclusive or exclusive) elements are called (inclusive or exclusive) disjuncts, and (iii) implications have elements as antecedents and consequences.*

The key focus of our work is to represent and compare relations of features and activities, according to the logical connectors of Definition 6. We follow mapping principles between features and activities in our validation [22].

In the following, we assume a business process family model $\Psi = \langle \Phi, \Omega, M \rangle$ that consists of a structured process model template $\Omega = \langle \mathcal{V}, \mathcal{E}, \mathcal{S}, \mathcal{D} \rangle$ ($\mathcal{A} \subseteq \mathcal{V}$), a feature model $\Phi = \langle \mathcal{F}, \mathcal{F}_M, \mathcal{F}_O, \mathcal{F}_{IOR}, \mathcal{F}_{XOR}, \mathcal{F}_{incl}, \mathcal{F}_{excl} \rangle$ and mappings M ($M \subseteq \mathcal{F} \times \mathcal{A}$).

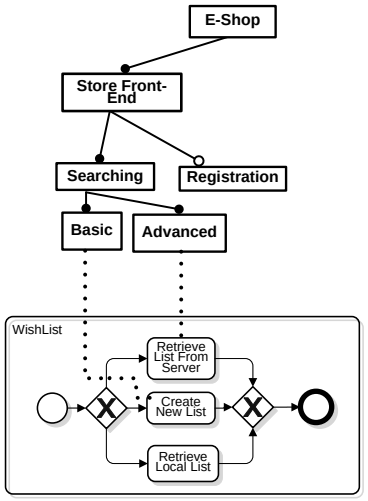

Figure 3: Strong inconsistency due to exclusive branching mismatch

In a comparison of the feature relations (FR) and the relations given by the control flow relations (CR) only mapped elements are relevant since unmapped element can not be involved in contradicting relations, given that we always assume correct feature models and correct process model templates.

Definition 7 (Strong Inconsistency). *Assume a control flow relation CR on activities $A_1, \ldots, A_n \in \mathcal{A}$ and a feature relation FR on $F_1, \ldots, F_m \in \mathcal{F}$, and mappings $(F_i, A_j) \in M$ of some features F_i ($i \in \{1, \ldots, m\}$) and activities A_j ($j \in \{1, \ldots, n\}$). A strong inconsistency between FR and CR occurs if there is no feature selection possible that leads to non-contradicting control flow relations of activities in CR.*

Figure 3 depicts a strong inconsistency of activities in exclusive sibling branches. The activities *Retrieve List From Server* and *Create New List* appear in exclusive sibling branches. We require that there is no contradicting perspective in the feature model that would suggest (and also allow) feature configurations that contradict to the exclusive behavior, as it is imposed by the process model template. The inconsistency in Figure 3 is due to the mapping of these activities to the features *Basic* and *Advanced* that can only appear together in each possible feature configuration, i.e., either both features are selected or none of them.

A weaker notion is the potential inconsistency. There are feature selections allowed in FR that lead to contradicting execution combinations of activities in CR. Obviously, each strong inconsistency is also a potential inconsistency.

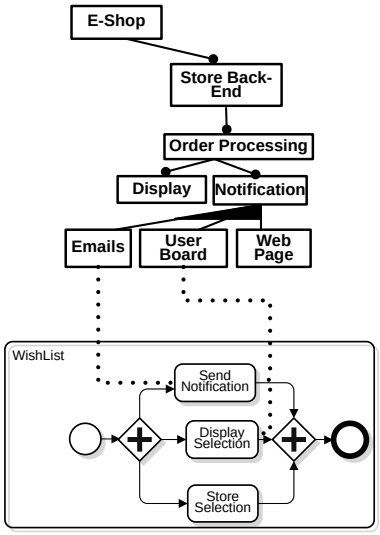

Figure 4: Potential inconsistency due to parallel branching

Figure 4 depicts an example for a potential inconsistency due to mappings between features that are (inclusive) or-siblings (*Emails* and *User Board*) and activities in parallel branches. Thus, a feature selection does not guarantee that all activities in parallel branches (*Send Notification* and *Display Selection*) occur together in a business process model. However, this is imposed by the process model template.

Definition 8 (Potential Inconsistency). *Assume a control flow relation CR on activities* $A_1, \ldots, A_n \in \mathcal{A}$ *and a feature relation FR on* $F_1, \ldots, F_m \in \mathcal{F}$*, and mappings* $(F_i, A_j) \in M$ *of* F_i *(*$i \in \{1, \ldots, m\}$*) and activities* A_j *(*$j \in \{1, \ldots, n\}$*). A potential inconsistency means there can be a feature selection within FR that result in a contradicting control flow relation of activities in CR.*

Figure 5 depicts another example of a potential inconsistency. The subprocess *WishList* is mapped to an optional feature *Registration*, while an internal activity *Retrieve List From Server* is mapped to the feature *Advanced* and *Advanced* is a mandatory child feature of feature *Searching*. The selection of feature *Store Front-End* requires the selection of its mandatory child *Searching* and therefore also the selection of *Advanced*, but the feature *Registration* is not necessarily selected. Thus, the subprocess *WishList*, which is mapped to the optional feature *Registration*, is removed, while at the same time the internal activity *Retrieve List From Server* should remain in the business process model.

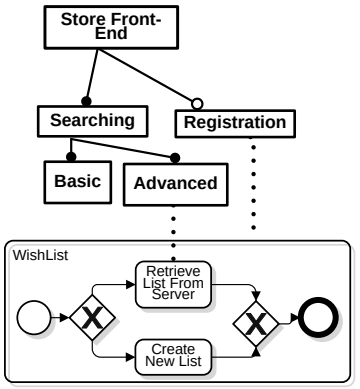

Figure 5: Potential inconsistency due to potential removal of the subprocess *WishList*, while the activity *Retrieve List From Server* is not removed

4.3. Overview of Inconsistencies

Table 1 summarizes the interrelationships between feature relationships and control flow relationships. We use four symbols to indicate whether we can identify an inconsistency or not: (i) '✓' indicates no inconsistency between feature relations *FR* and control flow relations *CR*, (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 features and activities, the corresponding cells are marked with '−'.

The comparison between feature and control flow relations in Table 1 refers to the following relations between elements: (i) Feature relations among sibling features (*and*, *xor*, *ior*) are logical connection among conjuncts or disjuncts. (ii) Integrity constraints of features (includes and excludes) are implications (and negations) from a feature as antecedent to a feature as consequent.

(iii) Branching patterns of the control flow patterns, the deferred choice and interleaved parallel routing pattern denote a logical combination of conjuncts or disjuncts, which are activities of sibling branches. (iv) The relations in a subprocess depicts an implications from a logical point of view. If we compare two implications, we mean the situation that the antecedents are mapped to each other, and likewise the consequences. Other combinations have no influence on the inconsistency in our approach.

5. Knowledge Base for Business Process Families

According to Section 4, the basic idea of our approach is to compare relationships between elements of both models. Thus, a knowledge base has to cover the feature model in terms of feature relations, the process model template by control flow relations, and mappings between both

Table 1: Interrelationships between Feature Relations and Control Flow Relations

Control Flow Relations	Feature Relations				
	AND	XOR	IOR	incl. (cr)	excl. (cr)
Sequence	–	–	–	–	–
Subprocess	✓	±	±	✓	↯
AND-AND Parallel split - Synchronization	✓	↯	±	±	↯
AND-OR Parallel split - Multi merge	✓	↯	±	±	↯
AND-XOR Parallel split - Simple merge	✓	↯	±	±	↯
AND-DISC Parallel split - Discriminator	✓	↯	±	±	↯
XOR-XOR Exclusive - Simple merge	↯	✓	±	±	✓
OR-DISC Multi choice - Discriminator	✓	✓	✓	±	±
OR-OR Multi choice - Synchronizing merge	✓	✓	✓	±	±
OR-XOR Multi choice - Simple merge	✓	✓	✓	±	±
Arbitrary Cycles	–	–	–	–	–
Implicit Termination	–	–	–	–	–
Multiple Instances (MI) Pattern	–	–	–	–	–
Deferred Choice	↯	✓	±	±	✓
Interleaved parallel routing	✓	↯	±	±	↯
Milestone	–	–	–	–	–
Cancellation Pattern	–	–	–	–	–

Symbols: no inconsistency (✓), strong (↯), potential (±) inconsistency, and no influence (–)

models by an equivalence between the corresponding entities.

As Description Logics are our underlying modeling formalism, this section starts with preliminaries of Description Logics, followed by the DL knowledge base for business process families.

5.1. Foundations of Description Logics

Description Logics are a decidable subset of first-order logic (FOL). A DL knowledge base is established by a set of terminological axioms (TBox) and assertions (ABox). 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, supplemented by the corresponding (FOL) expressions.

The basic elements of a TBox are atomic concepts and atomic roles. They are used to compose more complex concept expressions. Roles are used to state binary relations. Concept inclusion axioms $C \sqsubseteq D$ means that each individual of the concept C is also an individual of D. There are two special concepts in Description Logics, 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 that can not contain any individuals. A concept equivalence (or also called a definition) $C \equiv D$ is an abbreviation for two concept inclusion axioms $C \sqsubseteq D$ and $D \sqsubseteq C$. The same holds for subrole axioms (line 4 in Table 2).

A concept union is a complex concept expression and refers to a disjunction in FOL, i.e., an individual of an concept union has to be an individual of at least one concept C_i of the union. Likewise, a concept intersection refers to a conjunction in FOL. A concept negation $\neg C$ is the set of all individuals that are not individuals of the concept C. C can be an arbitrary complex concept.

Universal and existential quantifiers either restrict or require relations with other concepts (C in Table 2). These other referenced concepts can be arbitrary complex concepts.

Due to the well defined semantics of Description Logics, there are practically efficient reasoning algorithms and systems that offer several reasoning services. In the remainder of this paper, we use subsumption checking, which is one of the basic reasoning service. Subsumption checking refers to the question whether a concept C is subsumed by D, i.e., $C \sqsubseteq D$ holds in the knowledge base, whereby C and D can be complex concepts.

In this paper, Description Logics (DL) [11] are used to formalize the constraints of interests in models of interests and enable validation services. We could have used some other formalism, but we opted for DL as it is precise and expressive enough to serve our purpose – formalize constraints that need to hold between our models of interest. We built upon standard DL reasoning services in order to classify dependencies and constraints of elements of business process models and to validate the well-formedness of members of a business process family. Discussions of expansiveness of DL over some other options, although an important research topic, is outside of the scope of this paper.

Table 2: Constructs and Notations in DL and FOL Syntax

Construct Name	**DL Syntax**	**FOL Syntax**
atomic concept	C	$C(x)$
atomic role	R	$R(x, y)$
concept inclusion axiom	$C \sqsubseteq D$	$\forall x.C(x) \rightarrow D(x)$
role inclusion axiom	$R \sqsubseteq S$	$\forall x, y.R(x, y) \rightarrow S(x, y)$
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	$\exists P.C$	$\exists y.(P(x, y) \wedge C(y))$

5.2. Feature Relationships

We represent feature relations in a DL knowledge base Σ_Φ. The DL representation is based on the general modeling principles of Wang et al. [23]. However, the purpose of the work of Wang et al. is on validation of feature model configurations, while our target is an analysis of constraint influence of mapped goals to potential feature model configurations. Thus, we adapt some modeling principles according to our particular validation purpose.

For each feature F, we introduce a DL concept C_F, referred to as constraint concept of F. This concept collects all relationships how feature F is related to other features. The DL representation is depicted in Algorithm 1.

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 a subsumption checking between the different concepts that represent feature and control flow relations (cf. validation principles in Section 6).

In case a feature F_i is a mandatory child feature (lines 4–6), it is in an *and* relationship with its siblings in the logical sense, since we know that the selection of one mandatory sibling also requires the selection of all other mandatory sibling features. Thus, we add this conjunctive relation to the concept C_{F_i} for each mandatory sibling feature F_i.

An inclusive *or* decomposition of a feature F into features $F_1, \ldots, F_n$ is represented by a concept union over the sibling features of the *or* decomposition (lines 7–9). A disjunctive connection is added to the concept description C_{F_i} of each sibling feature F_i.

Likewise, exclusive *or* decomposition are described by concept unions. However, we need a further restriction that the selection of only one feature is allowed. This is described in the second part of the concept expression (line 10–12).

The *includes* integrity constraint specifies that the selection of a feature F also requires the selection of another feature F'. In DL, we define C_F dependent on the feature F' (lines 13–15). The *excludes* integrity constraint is defined similarly (lines 16–18). The concept negation expresses the exclusiveness of features F and F'.

Algorithm 1 Representation of the Feature Model Knowledge Base Σ_Φ

1: Input: $\langle \mathcal{F}, \mathcal{F}_M, \mathcal{F}_{IOR}, \mathcal{F}_{XOR}, \mathcal{F}_{incl}, \mathcal{F}_{excl} \rangle$
2: $\Sigma_\Phi := \emptyset$.
3: **for all** $F \in \mathcal{F}$ **do**
4: **for all** $(F, \{F_1, \ldots, F_n\}) \in \mathcal{F}_M$ **do**
5: $C_{F_i} := C_{F_i} \sqcap \bigsqcap_{i=1,\ldots,n} \exists requires.F_i$
6: **end for**
7: **for all** $(F, IOR, \{F_1, \ldots, F_n\}) \in \mathcal{F}_{IOR}$ **do**
8: $C_{F_i} := C_{F_i} \sqcap \bigsqcup_{i=1,\ldots,n} \exists requires.F_i$
9: **end for**
10: **for all** $(F, XOR, \{F_1, \ldots, F_n\}) \in \mathcal{F}_{XOR}$ **do**
11: $C_{F_i} := C_{F_i} \sqcap (\bigsqcup_{F' \in \{F_1,\ldots,F_n\}} \exists requires.F' \sqcap \neg(\bigsqcup_{F'',F''' \in \{F_1,\ldots,F_n\}} (\exists requires.F'' \sqcap \exists requires.F'''))$
12: **end for**
13: **for all** $(F, F') \in \mathcal{F}_{incl}$ **do**
14: $C_F := C_F \sqcap \exists requires.F'$
15: **end for**
16: **for all** $(F, F') \in \mathcal{F}_{excl}$ **do**
17: $C_F := C_F \sqcap \neg\exists requires.F'$
18: **end for**
19: $\Sigma_\Phi := \Sigma_\Phi \cup C_F$
20: **end for**

Features and their relations to other features are represented by concept definitions F and C_F. An excerpt of the e-store feature model is presented in Axioms 1 – 3.

$$C_{StoreFront\text{-}End} \equiv \exists\, requires.StoreFront - End \sqcap \exists\, requires.StoreBack - End \quad (1)$$

$$C_{Emails} \equiv \exists\, requires.Emails \sqcup \exists\, requires.UserBoard \sqcup \exists\, requires.WebPage \quad (2)$$

$$C_{NewUser} \equiv \exists\, requires.Registration \quad (3)$$

Axiom 1 defines *StoreFront-End* and *StoreBack-End* as conjunctive related elements, expressed by a constraint concept $C_{StoreFront\text{-}End}$. The same constraint concept is used for the feature *StoreBack-End*. We use the role *requires* to describe this relationship. There is no such axiom for optional features.

Algorithm 2 Representation of Branching Fragments in Σ_Ω

1: Input: $\Omega = \langle \mathcal{V}, \mathcal{E}, \mathcal{S}, \mathcal{D} \rangle$
2: $Rel_A \equiv \top$
3: **for all** $S \in \mathcal{S}$ **do**
4: **if** $S = (and, and, \mathcal{B}) \vee S = (and, or, \mathcal{B}) \vee S = (and, xor, \mathcal{B}) \vee S = (and, disc, \mathcal{B})$ **then**
5: $C_{A_i} := C_{A_i} \sqcap \bigsqcap_{B_j \in \mathcal{B}} \exists\ requires.(\bigsqcup_{A_k \in B_J} A_k)$
6: **end if**
7: **if** $S = (ior, ior, \mathcal{B}) \vee S = (ior, disc, \mathcal{B}) \vee S = (ior, xor, \mathcal{B})$ **then**
8: $C_{A_i} := C_{A_i} \sqcap \bigsqcup_{B_j \in \mathcal{B}} \exists\ requires.(\bigsqcup_{A_k \in B_J} A_k)$
9: **end if**
10: **if** $S = (xor, xor, \mathcal{B})$ **then**
11: $C_{A_i} := C_{A_i} \sqcap (\bigsqcup_{B_j \in \mathcal{B}} \exists\ requires.(\bigsqcup_{A_k \in B_J} A_k)) \sqcap \neg(\bigsqcap_{B_j \in \mathcal{B}} \exists\ requires.(\bigsqcup_{A_k \in B_J} A_k))$
12: **end if**
13: $\Sigma_\Omega := \Sigma_\Omega \cup C_{A_i}$
14: **end for**

For inclusive and exclusive disjuncts, a concept union is used. For instance, Axiom 2 defines *Emails*, *User Board* and *Web Page* as disjuncts. The same constraint concept is built for features *User Board* and *Web Page*. The axiom ensures that at least one feature is selected. Axiom 3 depicts an integrity constraint where the feature *New User* includes *Registration*. For integrity constraints, we use the same role *requires*. At the end, there is one single concept definition for each feature that contains all relationships of this feature.

The feature model representation in DL is different to the modeling principles of Wang et al. [23] due to the different scope of our validation. We are not interested in the validation of a particular feature configuration with respect to a given feature model. Instead, our focus is the validation of the process model template in which the relations of particular features of the feature model are taken into account. We use concept definitions to define feature relationships in order to ease the comparison. Furthermore, we only use one role *requires*, which also simplifies the relationship comparison for our validation purpose.

5.3. Control Flow Relationships

We represent business process model templates in terms of control flow relations *CR* between activities. Algorithm 2 describes how the corresponding knowledge base Σ_Ω is built.

There might be an overlapping of activity relations. Accordingly, we build relations of activities C_A as a conjunction (intersection in DL) of activities from the different control flow patterns. Initially, each relation concept C_A is defined as equivalent to the universal concept $\top$ (line 2).

In lines 4–6, the algorithm restricts the concept definitions C_{A_i}, in which A_i are activities 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 within the same branch by a concept union, independent of the kind of branching.

Algorithm 3 Subprocess Representation in Σ_Ω

1: Input: Σ_Ω
2: **for all** $D \in \mathcal{D}$ **do**
3: **if** $(A_{sub}, A_{inner}) \in D$ **then**
4: $C_{A_{inner}} := C_{A_{inner}} \sqcap \exists requires.A_{sub}$
5: $\Sigma_\Omega := \Sigma_\Omega \cup C_{A_{inner}}$
6: **end if**
7: **end for**

Multi choices are treated in lines 7–9, including synchronizing merge, simple merge and discriminator. Logically, activities of sibling branches are connected by a concept union. In case of exclusive branchings (lines 10–12), the concept definitions C_{A_i} contain a further restriction that allows only the execution of one branch. Like in feature models, this is expressed by a concept negation ($\neg$). In both cases, activities of the same branch are treated like in the parallel case, i.e., they are represented by a concept union since *ior* and *xor* relations refer to activities of sibling branches, but not to activities in the same branch.

Algorithm 3 extends the constraint concepts C_A of activities A that are subprocesses, i. e., they are decomposed into subactivities. The input is the result of the Algorithm 2. In this case, we require that if a particular configured business process model that contains an internal subactivity of A then A must be in the business process model as well.

The following axioms illustrate the DL representation of the control flow relations from the introduced example of Figure 1. Axiom 4 depicts the conjunctive relationships between activities *Display Selection*, *Send Notification* and *Store Selection*, represented by the concept expression $C_{DisplaySelection}$. The constraint concepts for the other activities ($C_{SendNotification}$ and $C_{StoreSelection}$) are the same.

Axiom 5 depicts the disjunctive connection of a choice with two branches, whereby the first branch contains activities *Enter Name* and *Save Name* and the second the activities *Enter UserID* and *Save UserID*. The disjunctive connection refers to activities of sibling branches, but not to activities within the same branch. Activities of the same branch are represented as a union (second part of the axioms). This representation of predecessors and successors of a branch is same for each kind of branching. Obviously, in this representation, the ordering among activities of one branch is neglected, as it is not influence by a configuration

in our case (see sequence pattern in Section 4).

$$C_{Display\ Selection} \equiv \exists\ requires.DisplaySelection \sqcap \exists\ requires.SendNotification \sqcap \exists\ requires.StoreSelection \quad (4)$$

$$C_{EnterName} \equiv \exists\ requires.(EnterName \sqcup SaveName) \sqcup \exists requires.\ (EnterUserID \sqcup SaveUserID) \quad (5)$$

6. Inconsistency Detection

In Section 5, we transformed feature models and business process models, which act as process model templates, into a knowledge base that specifies feature relations *FR* and control flow relations *CR*. This section proposes validation principles in order to analyze the influence of feature relations on control flow relations according to the comparison in Table 1 of Section 4.3.

Feature relations *FR* and control flow relations *CR* are both represented by a set of axioms in DL knowledge bases Σ_Φ and Σ_Ω. We assume that the feature model Φ and the business process model template Ω are correct models. Our validation approach comes into play if features are mapped to activities, describing the realization of features by activities in particular business process models.

The validation approach relies on the comparison of feature relations *FR* and control flow relations *CR* of mapped features and activities. We use Description Logic reasoning to test concept subsumption as well as concept satisfiability checking to detect inconsistencies between mapped features and activities. Especially for potential inconsistencies, DL reasoning is a dedicated mean to compare concepts (constraint concepts C_F and C_A) and distinguish between cases where certain concepts (referring to relations) are unsatisfiable, potential satisfiable or even always satisfied.

The transformation of relations to DL in Section 5 is dedicated for concept comparison. All constraint concepts (C_F and C_A) are defined by DL concept definitions and we use the same role to express relationships between features and activities. Finally, a mapping of a feature to an activity is represented by an equivalence between concepts of the feature and activity concept in the knowledge base.

So far, we are aware of the idea and modeling capabilities to compare relations *FR* and *CR* of the knowledge base. A further consideration is how the relations *FR* and *CR* can be compared in order to detect inconsistencies according to the classification in Table 1. For this purpose, we break down the interrelationships of Table 1 for strong and potential inconsistencies to a comparison between logical formulas, represented by constraint concepts in DL:

- A potential inconsistency is detected if the satisfaction of the feature relation *FR* of feature F does not necessarily imply the satisfaction of the corresponding control flow relation *CR* of activity A. Thus, the concept C_F (feature relations of F) is not subsumed by C_A, which represents the control flow relations of A.
- A strong inconsistency is recognized by contradicting relations of feature F and activity A. Thus, in DL, the intersection of the corresponding constraint concepts C_F and C_A is unsatisfiable. This means the intersection $C_F \sqcap C_A$ is subsumed by the empty concept $\bot$.

We build the final knowledge base Σ by adding validation concepts $C_{F \Rightarrow A}$ (for potential inconsistency detection) and $C_{F \wedge A}$ (for strong inconsistency detection) for each mapping $m(F, A)$ between feature F and activity A. The validation concepts $C_{F \Rightarrow A}$ and $C_{F \wedge A}$ are composed as described in Definition 9.

Definition 9 (Final Knowledge Base). *The final knowledge base Σ is constructed from the knowledge bases of the feature model, the process model template and the mappings, i.e., $\Sigma := \Sigma_\Phi \cup \Sigma_\Omega$. Moreover, for each mapped activity A and its corresponding feature F ($m(F, A)$), which is represented in the mapping knowledge base Σ_M by an axiom $F \equiv A$, we insert the following axioms into Σ:*

- $C_{F \Rightarrow A} \equiv \neg C_F \ \sqcup \ C_A$
- $C_{F \wedge A} \equiv C_F \ \sqcap \ C_A$

Given the final knowledge base Σ, we get the validation result *en passant*. The validation concepts $C_{F \Rightarrow A}$ and $C_{F \wedge A}$ are classified by DL reasoning. The concept classification leads to the following insights:

1. The validation concept $C_{F \Rightarrow A}$ indicates a potential inconsistency. If $C_{F \Rightarrow A}$ is classified equal to the universal concept $\top$ we can guarantee that the satisfaction of feature relations *FR* in each particular feature configuration ensures the fulfillment of control flow relations *CR*. *FR* are feature relations of feature F and *CR* are control flow relations of the corresponding activity A.
2. Otherwise, $C_{F \Rightarrow A} \not\equiv \top$ holds, and we know that there is either a strong or a potential inconsistency. We identify the first case if the validation concept $C_{F \wedge A}$ is classified equal to the empty concept $\bot$.

7. Correctness of the Validation

This section demonstrates that the detection of inconsistencies between feature and control flow relations can be reduced to subsumption and classification of the validation concepts $C_{F \Rightarrow A}$ and $C_{F \wedge A}$. We start with a consideration of the relationship coverage by these concepts. Afterwards, we show that the classification of these validation concepts by DL reasoning comply with the criteria for relationship comparison of Table 1 (Section 4.3).

7.1. Relationship Coverage

The relationships of both models are represented in a common DL knowledge base Σ. Furthermore, we know that both models are correct on their own. Thus, if an activity A is not mapped to any feature there is no violation of control flow relations of A. Therefore, only cases where activities are mapped to features are relevant for the validation. If an activity is mapped to multiple features, we expect that none of these mappings causes any inconsistency.

Feature relations FR of a feature F are represented by a concept C_F, and likewise control flow relations CR of an activity A are described by a concept C_A in the knowledge base. Mappings between feature and activities $m(F, A)$ are represented by equivalence axioms in the knowledge base. For each mapping, the corresponding concepts C_F and C_A are compared in order to determine the influence of feature relations FR on control flow relations CR.

7.2. Validation Principle

In the following, we show that the detection of strong and potential inconsistency is correctly achieved in the validation by DL reasoning. The detection of inconsistencies is solved by the classification of the validation concepts $C_{F \Rightarrow A}$ and $C_{F \wedge A}$. Lemma 1 summarizes these statements.

Lemma 1 (Correctness of the Validation). *For mappings from an activity A to a feature F, C_F are the feature relations of feature F and C_A the control flow relation of activity A. The following statements hold:*

- *The concept $C_{F \Rightarrow A}$ is classified equal to the universal concept $\top$ iff all dependent activities of activity A appear in a particular business process model, whenever feature F appears in a feature configuration.*
- *The concept $C_{F \wedge A}$ is classified equal to the empty concept $\bot$, iff there is no particular business process model possible where each activity satisfies the control flow relations, whenever feature F is selected in a particular feature configuration.*

Proof. Looking to the different types of relations in both models, we basically deal with *implication*, *and*, *ior* and *xor*, as already outlined in Definition 6. Hence, we have to consider all possible combinations in both models and check whether either $C_F \Rightarrow C_A$ is a tautology (i.e., $C_{F \Rightarrow A} \equiv \top$) or $C_F \wedge C_A$ is satisfiable (i.e., $C_{F \wedge A} \not\sqsubseteq \bot$).

This kind of logical problem is in the nature of propositional logic. Hence, in the knowledge base Σ, we define the DL expressions C_F and C_A in a propositional style. The term connectors are the DL counterparts, e.g., the intersection ($\sqcap$) for an and ($\wedge$). Instead of propositional variables, there are concept expressions like $\exists requires.F$ containing features or activities, as well as the role *requires* from Σ_Φ, Σ_Ω. The concept equivalence of F and A for each mapping $m(F, A)$ ensures the comparability of constraint concepts C_F and C_A. □

8. Proof-of-Concept and Discussion

The evaluation of our approach has been conducted by providing a proof-of-concept, which has been implemented by integrating the FeatureMapper [24] and the transformation of the control flow parts of BPMN to DL, as described in [25].

For the proof-of-concept, we use the e-store product line [26], which has been introduced in Section 2, and an external trial case, the video post-production business process model [27]. This trial case has already been used for validation of solutions to similar problems, e.g., in [28]. The external trial case is presented in Subsection 8.1, followed by the evaluation details in Subsection 8.2. After the evaluation details, a discussion about modeling and validation rationale is presented (Subsections 8.3–8.5).

8.1. External Trial Case

In order to validate our approach, i.e., to demonstrate that it is useful in use, we have migrated the business process model described in [28] from EPC to the solution where variability is explicitly described in a feature model, and business logic in one separate business process model. The feature model is presented in Figure 6.

The video post-production is a process for creating final versions of movies, and consists of features *Spotting session*, *Design*, *Progress Update*, *Premixing*, *Picture Editing* and *Final Mixing*, whereby *Progress Update* and *Premixing* are optional features. These feature are further decomposed. Small boxes are used for denoting mappings in the business process model template, described in Figure 7.

While the feature model is created by analyzing the variability in the case study presented in [28], the business process template has been created separately, by specifying an activity that is carried out by several roles (e.g., *Progress Update* by *Director* and *Producer* in [28]), as parallel activities carried out by lanes representing those roles (e.g., *Producer Progress Update* and *Director Progress Update* in Figure 7).

8.2. Performance Evaluation and Observation

Efficiency and tractability of the DL validation is applied in both test cases. Besides showing the practical scalability of the proposed validation approach, we also want (1) to investigate whether/how the size of the business process family model increases the performance and (2) to test whether/how the number of inconsistencies influences the performance.

Influence of the Model Size. We build models from the two mentioned trial cases in order to analyze how the size of the model influences the performance. First, the e-store consists of a large feature model with 287 features, 192 of the features are leaf features. There are 21 cross-tree constraints. The process model template has 84–120 activities. Secondly, the post-production trial case has a

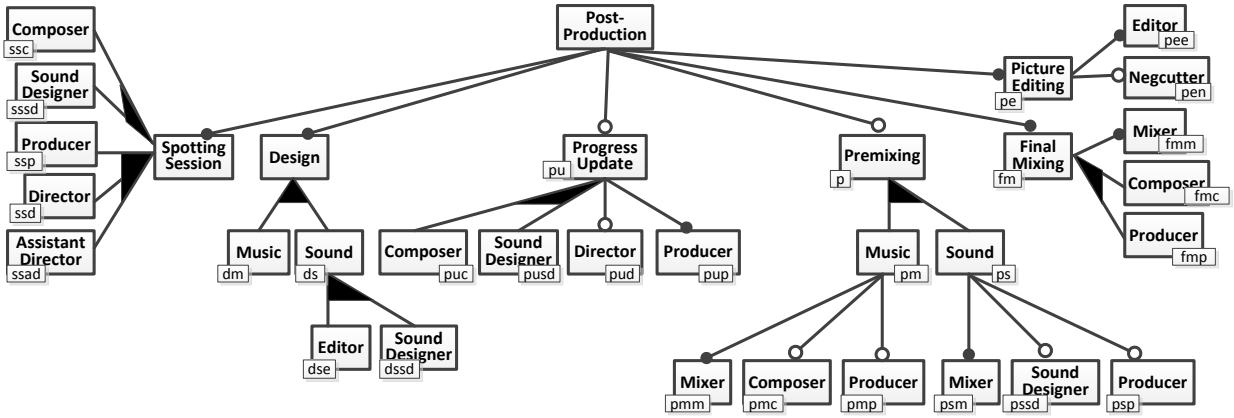

Figure 6: Video post-production feature model

Table 3: Validation time for different model sizes without inconsistencies

No.	Features	Activities	Mappings	Time [msec.]	
				Av.	Max.
1	150	84	40	3010	3310
2	150	84	60	4010	4190
3	150	120	40	3100	3280
4	150	120	60	4080	4260
5	287	84	40	3590	3780
6	287	84	60	4070	4220
7	287	120	40	3430	3560
8	287	120	60	4270	4420
9	25	15	15	1970	2020
10	25	23	23	2040	2100
11	34	15	15	1980	2060
12	34	23	23	2096	2140

feature model with 34 features and a process model template with 23 activities.

In both cases, we used multipliers in order to obtain 48 different business process family models in eight different settings (six for each setting) for the e-store trial case, and 24 different business process family models (six for each setting, four settings) for the video post-production trial case. The mappings are established manually.

The performance for each setting is depicted in Table 3, No. 1–12. Settings 1–8 refer to the e-store case study and settings 9–12 describe the results for models based on the video post-production trial case.

The result indicates that the size of the business process family model in terms of features, activities and mappings has impact on the running time. The number of mappings has the main impact on the reasoning performance. This is quite intuitive since for each mapping we introduce two validation concepts as complex concept expressions (using concept intersection, union and negation). The reasoner must classify these concepts check whether they are equal to the universal or empty concept. As we can observe, the execution time of our algorithm for the largest experimented model is less than 4.5 seconds.

Influence of the Number of Inconsistencies. The setting with the largest business process family models (No. 8 in Table 3) is used in order to test whether the number of strong and potential inconsistencies might influence the validation time. Table 4 shows the different settings and results.

Each setting contains six different business process family models. The setting in the first row is the same as in row 8 in Table 3. Each business process family model (row 1) contains 60 mappings. Obviously, each mapping might cause an inconsistency of the activity and the mapped feature due to their different relationships. Thus, the idea of this test is to manually change only the mappings in each business process family model (for each setting), such that some mappings (0, 20 or 40) might cause inconsistencies. Thus, in each business process family model there are either 0, 20 or 40 inconsistencies.

This is reflected in each row of Table 4. In row 2, 20 of the total 80 mappings cause a potential inconsistencies of activities (and their features). In row 3, 20 mappings lead to strong inconsistencies, while each strong inconsistency is also a potential inconsistency. Row 4 describes settings with 20 potential and 20 strong (and potential) inconsistencies. Row 5 and 6 capture cases with 40 potential and 40 strong (and potential) inconsistencies, respectively.

As indicated by the average and maximum validation time, there are only minor differences regarding the time. This is not surprising as the reasoning algorithms classifies for each mapping $m(F, A)$ the corresponding concepts

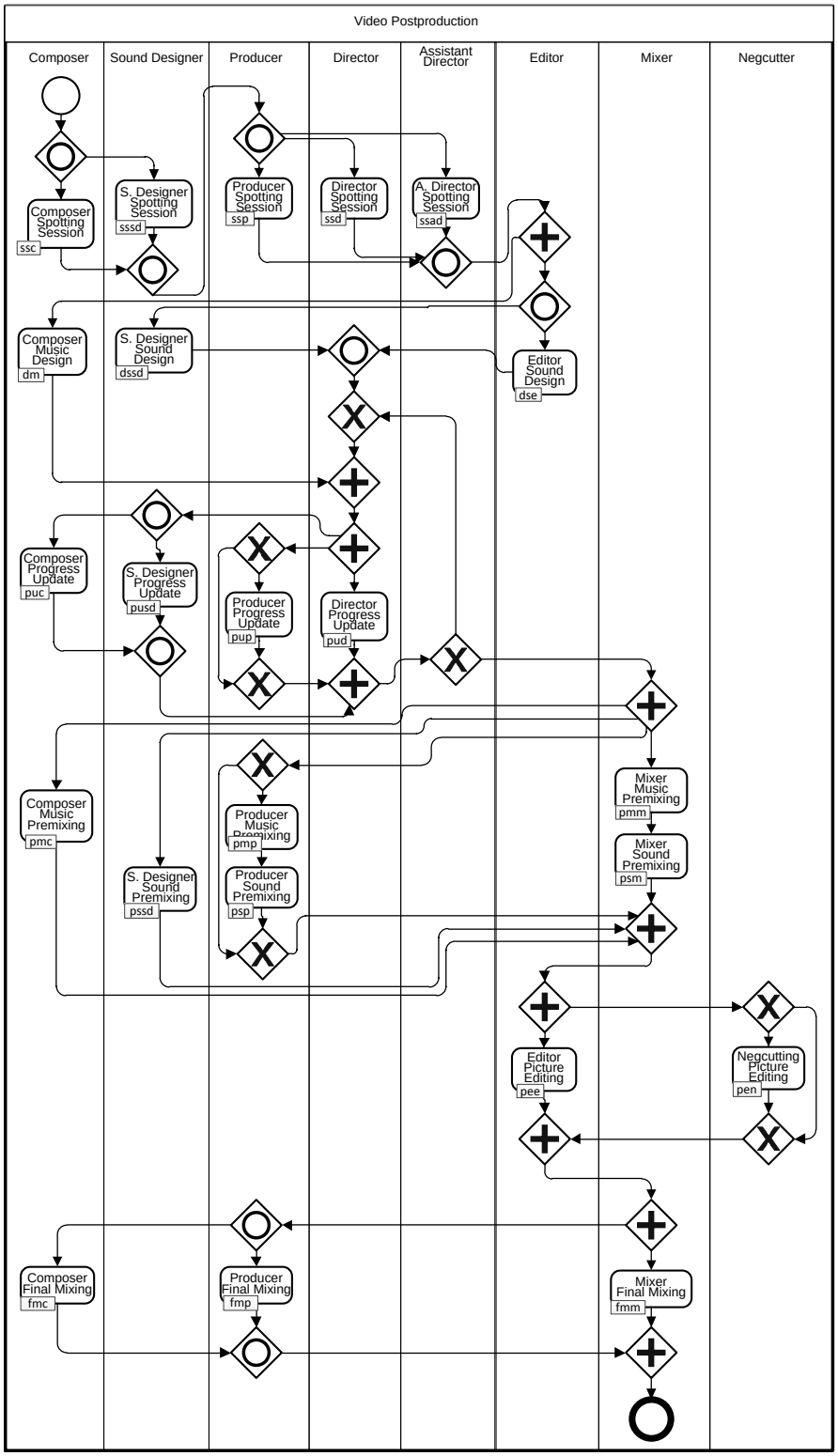

Figure 7: Video post-production superimposed business process model

Table 4: Validation Time with fixed Model Size

No.	Number of Inconsistencies		Time [msec.]	
	potential	strong	Av.	Max.
1	0	0	4270	4420
2	20	0	4310	4480
3	0	20	4320	4500
4	20	20	4360	4490
5	40	0	4350	4470
6	0	40	4370	4420

$C_{F \Rightarrow A}$ and $C_{F \wedge A}$ and checks whether $C_{F \Rightarrow A}$ is equal the universal concept $\top$ and compares $C_{F \wedge A}$ with the empty concept $\bot$, but this is done for each mapping and not only for those that cause an inconsistency.

All inconsistencies are correctly recognized by DL reasoning. The classification of the validation concepts directly pinpoints the source of an inconsistency, i.e., we directly know which mapping between feature and activity causes an inconsistency in the knowledge base.

Implementation Details. The time for the transformation to DL is less than the validation time. This is based on the fact that we use the DL-oriented feature model of [23] and we only transform the relevant control flow information of BPMN to DL.

The ontology creation is implemented with the OWL-API[1]. For reasoning, we used the Pellet reasoner[2]. Our test system is a Notebook with an Intel Core 2 Duo 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.

8.3. Validation Exemplified

We demonstrate the validation for the example of Figure 1 for a potential and strong inconsistency.

Potential Inconsistency. If we take the current situation in Figure 1 as an example, activities *Send Notification*, *Display Selection* and *Store Selection* are parallel siblings, i. e., conjunctively connected in their control flow relations. *Send Notification* and *Display Selection* are mapped to features *Emails* and *User Board*, which are both inclusive *or*-siblings, i. e., inclusive disjuncts in the feature relation.

There is no strong inconsistency, but a potential inconsistency, since there are feature configuration where only one of these sibling features is selected, but the corresponding process model template assumes that both activities are in parallel branches and thus both need to be executed.

Axiom 6 depicts the feature relation of feature *Emails*, and Axiom 7 the axiom for the control flow relation of activity *Send Notification*. Both are mapped to each other (Axiom 8). The validation concept $C_{Emails \wedge SendNotification}$ (i.e., $C_{Emails} \sqcap C_{SendNotification}$) is satisfiable. Thus, there is no strong inconsistency.

$$C_{E\text{-}mails} \equiv \exists\, requires.Emails \sqcup \exists\, requires.UserBoard \sqcup \exists requires.WebPage \quad (6)$$

$$C_{SendNotification} \equiv \exists\, requires.DisplaySelection \sqcap \exists\, requires.SendNotification \sqcap \exists\, requires.StoreSelection \quad (7)$$

$$Emails \equiv SendNotification \quad (8)$$

However, the validation concept $C_{Emails \Rightarrow SendNotification}$ is not equal to the universal concept $\top$, i.e., logically the implication of the feature relation to the control flow relations is not a tautology. Therefore, we can not guarantee that for each feature selection that contains the feature *Emails* the control flow relation of activity *Send Notification* holds.

Strong Inconsistency. For a strong inconsistency, we slightly modify Axiom 6 to an XOR-group of features, as depicted in Axiom 9. The other axioms remain unchanged. In this case, the validation concept $C_{Emails \wedge SendNotification}$ is unsatisfiable as we can see by the concept negations in Axiom 9 compared to Axiom 6.

$$\begin{aligned} C_{Emails} \equiv\ & (\exists\, requires.Emails \sqcup \exists\, requires.UserBoard \\ & \sqcup \exists requires.WebPage) \\ & \sqcap \neg(\exists\, requires.Emails \sqcap \exists\, requires.UserBoard) \\ & \sqcap \neg(\exists\, requires.Emails \sqcap \exists\, requires.WebPage) \\ & \sqcap \neg(\exists\, requires.UserBoard \\ & \quad \sqcap \exists\, requires.WebPage) \end{aligned} \quad (9)$$

8.4. Modeling and Validation Rationale

Following the principles from software product line engineering, we use feature models to align between system features and their realization. They serve as a configuration means by selecting features that remain in a particular configuration and those that will be removed. According to this principle of removing features and therefore also activities, our validation has to incorporate the case where relationships are contradicting and where the removal of activities might cause some inconsistencies. However, the configuration does not change certain relations of activities like sequential ordering among activities, and therefore certain control flow relations are not affected in our configuration approach (cf. Section 4).

The presented modeling principles for feature and control flow relations are dedicated for this kind of relationship comparison, where elements of these relations (i.e., features and activities) are mapped to each other. Unmapped elements are neglected in the comparison, as only mapped elements cause interrelationships among element relations. This is reflected in the knowledge base as only for mapped elements (expressed as an equivalence of elements) an inconsistency can occur, otherwise the constraint concept will not be unsatisfiable since the expressions are not related to each other.

[1] OWL-API site: http://owlapi.sourceforge.net/
[2] Pellet reasoner site: http://clarkparsia.com/pellet/

8.5. 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 for the validation is threefold:

- We use reasoning to recognize inconsistencies between feature relations and control flow relations.
- 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 (feature and activity) is part of an inconsistency. This is crucial in large models, where various mappings exist.
- We use several notions how a reasoner can classify a concept expression in DL, i.e., whether a concept expression is always satisfied, is satisfiable or is unsatisfiable. This is exploited in order to distinguish between realization equivalence, strong and potential inconsistency.

Compared to other logical formalisms, Description Logics is quite efficient in practical settings, is expressive enough to capture complex feature and process models, 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 logics, Description Logics is more expressive and allows the integration of further background knowledge like annotations of elements.

9. Related Work

Due to the increasing need of business processes customization, several approaches for the development of families of business processes have been introduced like Schnieders et al. [29], Boffoli et al. [30], La Rosa et al. [31, 7, 32] and van der Aalst et al. [5, 6]. Schnieders et al. [29] model families of business process models as a variant-rich business process model. A configuration of such a family is performed by directly selecting business process elements of variant-rich processes. In order to support such an approach, Schnieders et al. extend BPMN with concepts for modeling variation. However, in order to perform it, such an approach requires from a customer knowledge of business process modeling.

Boffoli et al. [30] and La Rosa et al. [31, 7, 32] also distinguish between business process models and variability models. Boffoli et al. model problem space as variability table, while La Rosa et al. provide variability by questionnaires. They provide guidance to derive valid configurations, while our aim is to guarantee that for each possible and valid feature configuration there is a corresponding valid process model that satisfies the well-formedness constraints.

Metzger et al [15] introduce one more approach for reasoning and configuring variants of variant-rich software. In their approach they distinguish a product line variability and software variability. Product line variability, captured with the OVM variability modeling language [8, 33] is used to represent members of a product line, i.e., product planned by the management. Therefore, OVM represents variability within late requirements and design models. Software variability, in Metzger et al.'s approach, captures variability within systems that can be built from the existing platform. Our approach uses business process models to model service oriented systems, and therefore, corresponds to what Metzger et al. consider software variability.

More similar to our objective is the approach for process configuration from van der Aalst et al. [5, 6]. Their framework ensures correctness-preserving configuration of (reference) process models. In contrast to our work, they capture the variability directly in the workflow net by variation points of transitions. Accordingly, a configuration is built by assigning a value to the transitions, while our approach uses feature selections.

Reinhartz-Berger et al. [34, 35] use a reference modeling approach, which is based on the proposed reference modeling language of Rosemann et al. [4]. Their approach supports the specification of guidelines and constraints for process model configuration. The guidelines and constraints are represented within the reference model, which refers to the process model template in our case. Like in the previously mentioned approaches, the configuration of process models for a given reference model requires expert knowledge in business process modeling.

Slightly related to our work is the validation approach for flexible workflows [36], in which a model template offers alternatives and constraints impose certain (runtime) restrictions on these alternatives.

Weidlich et al. [17, 21] derive behavior profiles to describe the essential behavior in terms of activity relations like exclusivity, interleaving and ordering of activities. A set algebra for behavior profiles is presented in [37]. The purpose is to manage several process variants in terms of set-theoretic relationships. Weber et al. [38] extend process models by semantic annotations and use them for the validation of process behavior correctness that captures control-flow interaction and behavior of activities. In contrast to our work, their focus is on behavioral constraints, while we consider structural well-formedness constraints. Moreover, our particular emphasis is on the feature-oriented process family representation.

A different way of variability management is proposed in terms of specified change operations, as described by Hallerbach et al. [39] and Reichert et al. [40]. Likewise, Weber et al. [41, 42] introduced several change patterns to increase process management flexibility, but ensure a

certain behavior preservation.

In the context of SPLs several approaches have been introduced, in order to ensure the well-formedness of solution space models. Czarnecki et al. [43] specify constraints on solution space model configurations using OCL constraints. Problem space models, solution space models with OCL constraints, and mappings between them are transformed to Binary-Decision Diagrams.

Thaker et al. [44] introduce an approach for the verification of type safety, i.e., the absence of references to undefined classes, methods, and variables, in solution space models w.r.t. all possible problem space configurations. They specify the models and their relations as propositional formulas and use SAT solvers to detect inconsistencies. Janota et al. [45] and van der Storm [46] introduce approaches to validating the correctness of mappings between feature and component models. They use propositional logics too.

Shobbens et al. [47] provide a generic semantics of feature diagrams that should serve as a basis for implementing reasoning procedure. In this paper, due to reuse of the solution that satisfies our requirements, we reuse the DL representation of feature diagrams introduced by Wang et al. [23]. Our representation is dedicated for the comparison of relationships between features and activities. Thus, we cover all relationships of a mapped feature by a single complex concept expression.

In addition to the more detailed coverage of all of its aspects, this paper extends our original approach [48] with advanced branching patterns, since these patterns frequently appear in contemporary business process models. Furthermore, a deeper investigation in the tractability evaluation and related work has been conducted.

10. Conclusion

As shown in the related work section, our contribution is primarily related to the modeling and validation of families of business processes. While basic ideas of business process families were previously introduced and even covered in our own work [22], there have been very limited (if any) attempts to propose a validation approach of such families.

Our proposal validates business process models with respect to their control flow relations; mappings to feature models; and dependencies in the feature models. Hence, unlike other approaches on validation of (model-driven) software product lines, our approach also considers the very nature of business process models through the set of business process practices encoded in control flow relations. Even though, in this paper, we used BPMN for defining the solution space of business process families, our approach is easily generalizable to other types of business process modeling languages. This can be deduced from control flow and branching patterns used in this paper and the control flow support analyzes presented in the relevant literature [20].

We evaluated our work with the largest publicly available case study, for which we were able to find both types of models. Furthermore, we used a further trial case that has been represented in both feature and business process models. While these case studies have a realistic size, we would like to have a benchmarking framework, which will allow for simulating larger models for business process families. This is similar to what has been already proposed for feature models [49], but now to be enriched for the generation of business process model templates of different characteristics. We plan to extend our validation formalism towards behavioral constraints in the process model template.

References

[1] M. Weske, Business Process Management: Concepts, Languages, Architectures, Springer, 2007.

[2] J. Mendling, Three Challenges for Process Model Reuse, in: Business Process Management Workshops (2), 2011, pp. 285–288.

[3] A. Hallerbach, T. Bauer, M. Reichert, Capturing Variability in Business Process Models: the Provop Approach, Journal of Software Maintenance 22 (6-7) (2010) 519–546.

[4] M. Rosemann, W. M. P. van der Aalst, A Configurable Reference Modelling Language, Inf. Syst. 32 (1) (2007) 1–23.

[5] W. M. P. van der Aalst, M. Dumas, F. Gottschalk, A. H. M. ter Hofstede, M. L. Rosa, J. Mendling, Preserving Correctness during Business Process Model Configuration, Formal Asp. Comput. 22 (3-4) (2010) 459–482.

[6] W. M. P. van der Aalst, N. Lohmann, M. L. Rosa, J. Xu, Correctness Ensuring Process Configuration: An Approach Based on Partner Synthesis, in: Business Process Management, 8th Int. Conference, BPM, Vol. 6336 of LNCS, Springer, 2010, pp. 95–111.

[7] M. La Rosa, W. M. P. van der Aalst, M. Dumas, A. H. M. ter Hofstede, Questionnaire-based Variability Modeling for System Configuration, SoSyM 8 (2) (2009) 251–274.

[8] K. Pohl, G. Böckle, F. van der Linden, Software Product Line Engineering - Foundations, Principles and Techniques, Springer, 2005.

[9] K. Czarnecki, M. Antkiewicz, Mapping Features to Models: A Template Approach Based on Superimposed Variants, in: GPCE'05, 2005, pp. 422–437.

[10] W. van der Aalst, A. ter Hofstede, B. Kiepuszewski, A. Barros, Workflow Patterns, in: Distributed and Parallel Databases, 2003.

[11] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, P. Patel-Schneider, The Description Logic Handbook, Cambridge University Press, 2007.

[12] M. Dumas, W. van der Aalst, A. H. M. ter Hofstede, Process-Aware Information Systems: Bridging People and Software through Process Technology, Wiley-Interscience, 2005.

[13] K. Kang, J. Lee, P. Donohoe, Feature-Oriented Product Line Engineering, IEEE software (2002) 58–65.

[14] M. Antkiewicz, K. Czarnecki, FeaturePlugin: Feature Modeling plug-in for Eclipse, in: Proceedings of the 2004 OOPSLA workshop on eclipse technology eXchange, eclipse '04, ACM, 2004, pp. 67–72.

[15] A. Metzger, P. Heymans, K. Pohl, P.-Y. Schobbens, G. Saval, Disambiguating the Documentation of Variability in Software Product Lines: A Separation of Concerns, Formalization and Automated Analysis, in: 15th IEEE Int. Requirements Engineering Conference, RE, IEEE, 2007, pp. 243–253.

[16] OMG, Business Process Model and Notation (BPMN), http://www.omg.org/spec/BPMN/2.0, 2009.

[17] M. Weidlich, A. Polyvyanyy, J. Mendling, M. Weske, Efficient Computation of Causal Behavioural Profiles Using Structural

Decomposition, in: Petri Nets, Vol. 6128 of LNCS, Springer, 2010, pp. 63–83.
[18] A. Polyvyanyy, L. García-Bañuelos, M. Dumas, Structuring Acyclic Process Models, in: BPM, Vol. 6336 of LNCS, Springer, 2010, pp. 276–293.
[19] N. Russel, A. ter Hofstede, W. van der Aalst, N. Mulyar, Workflow Control-Flow Patterns: A Revised View, Tech. rep., BPM Center Report BPM-06-22, BPMcenter.org (2006).
[20] P. Wohed, W. van der Aalst, M. Dumas, A. ter Hofstede, N. Russell, On the Suitability of BPMN for Business Process Modelling, in: Proceedings of the 4th International Conference on Business Process Management, 2006.
[21] M. Weidlich, J. Mendling, M. Weske, Efficient Consistency Measurement Based on Behavioural Profiles of Process Models, IEEE Transactions on Software Engineering 99. doi:http://doi.ieeecomputersociety.org/10.1109/TSE.2010.96.
[22] B. Mohabbati, M. Hatala, D. Gašević, M. Asadi, M. Bošković, Development and Configuration of Service-Oriented Systems Families, in: Proceedings of the 26th ACM Symposium on Applied Computing, 2011.
[23] H. Wang, Y. Li, J. Sun, H. Zhang, J. Pan, Verifying Feature Models using OWL, J of Web Semantics 5 (2) (2007) 117–129.
[24] F. Heidenreich, J. Kopcsek, C. Wende, FeatureMapper: Mapping Features to Models, in: ICSE'08 Companion, ACM, 2008, pp. 943–944.
[25] Y. Ren, G. Gröner, J. Lemcke, T. Rahmani, A. Friesen, Y. Zhao, J. Z. Pan, S. Staab, Validating Process Refinement with Ontologies, in: Description Logics, Vol. 477 of CEUR Workshop Proceedings, CEUR-WS.org, 2009.
[26] S. Q. Lau, Domain Analysis of E-Commerce Systems Using Feature-Based Model Templates, Master's thesis, University of Waterloo, Waterloo (2006).
[27] E. Marcos, Software Engineering Research versus Software Development, SIGSOFT Software Engineering Notes 30 (4) (2005) 1–7.
[28] M. La Rosa, M. Dumas, A. H. ter Hofstede, J. Mendling, Configurable Multi-perspective Business Process Models, Information Systems 36 (2) (2011) 313 – 340.
[29] A. Schnieders, F. Puhlmann, Variability Mechanisms in E-Business Process Families, in: BIS 2006: 9th Int Conf. on Business Information Systems, 2006, pp. 583–601.
[30] N. Boffoli, M. Cimitile, F. M. Maggi, Managing Business Process Flexibility and Reuse through Business Process Lines, in: ICSOFT (2), Proc. of the 4th Int. Conf. on Software and Data Techn., 2009, pp. 61–68.
[31] M. La Rosa, J. Lux, S. Seidel, M. Dumas, A. H. M. ter Hofstede, Questionnaire-Driven Configuration of Reference Process Models, in: Advanced Information Systems Engineering, 19th Int. Conference, CAiSE, Vol. 4495 of LNCS, Springer, 2007, pp. 424–438.
[32] M. La Rosa, F. Gottschalk, M. Dumas, W. van der Aalst, Linking Domain Models and Process Models for Reference Model Configuration, in: Business Process Management Workshops, Vol. 4928 of LNCS, Springer, 2008, pp. 417–430.
[33] S. Buhne, K. Lauenroth, K. Pohl, Modelling Requirements Variability across Product Lines, in: Proceedings of the 13th IEEE International Conference on Requirements Engineering, IEEE Computer Society, 2005, pp. 41–52. doi:10.1109/RE.2005.45.
[34] I. Reinhartz-Berger, P. Soffer, A. Sturm, Extending the Adaptability of Reference Models, IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans 40 (5) (2010) 1045–1056.
[35] I. Reinhartz-Berger, P. Soffer, A. Sturm, Organisational reference models: supporting an adequate design of local business processes, International Journal of Business Process Integration and Management 4 (2) (2009) 134–149.
[36] S. Sadiq, M. Orlowska, W. Sadiq, Specification and Validation of Process Constraints for Flexible Workflows, Information Systems 30 (5) (2005) 349–378.
[37] M. Weidlich, J. Mendling, M. Weske, A Foundational Approach for Managing Process Variability, in: Advanced Information Systems Engineering - 23rd Int. Conference, CAiSE, Vol. 6741 of LNCS, Springer, 2011, pp. 267–282.
[38] I. Weber, J. Hoffmann, J. Mendling, Beyond Soundness: On the Verification of Semantic Business Process Models, Distributed and Parallel Databases 27 (3) (2010) 271–343.
[39] A. Hallerbach, T. Bauer, M. Reichert, Capturing Variability in Business Process Models: the Provop Approach, Journal of Software Maintenance 22 (6-7) (2010) 519–546.
[40] M. Reichert, S. Rinderle, U. Kreher, P. Dadam, Adaptive Process Management with ADEPT2, in: 21st Int. Conference on Data Engineering, ICDE, IEEE Computer Society, 2005, pp. 1113–1114.
[41] B. Weber, S. Rinderle, M. Reichert, Change Patterns and Change Support Features in Process-Aware Information Systems, in: Advanced Information Systems Engineering, 19th Int. Conference, CAiSE, Vol. 4495 of LNCS, Springer, 2007, pp. 574–588.
[42] B. Weber, M. Reichert, S. Rinderle-Ma, Change Patterns and Change Support Features - Enhancing Flexibility in Process-Aware Information Systems, Data Knowl. Eng. 66 (3) (2008) 438–466.
[43] K. Czarnecki, K. Pietroszek, Verifying Feature-Based Model Templates Against Well-Formedness OCL Constraints, in: GPCE'06, ACM, 2006, pp. 211–220.
[44] S. Thaker, D. Batory, D. Kitchin, W. Cook, Safe Composition of Product Lines, in: GPCE'07, ACM, 2007, pp. 95–104.
[45] M. Janota, G. Botterweck, Formal Approach to Integrating Feature and Architecture Models, in: FASE, Vol. 4961 of LNCS, 2008, pp. 31–45.
[46] T. Van Der Storm, Generic Feature-based Software Composition, in: 6th Int. Conference on Software Composition, Vol. 4829 of LNCS, 2007, pp. 66–80.
[47] P.-Y. Schobbens, P. Heymans, J.-C. Trigaux, Y. Bontemps, Generic Semantics of Feature Diagrams, Computer Networks 51 (2) (2007) 456–479.
[48] G. Gröner, C. Wende, M. Bošković, F. S. Parreiras, T. Walter, F. Heidenreich, D. Gašević, S. Staab, Validation of Families of Business Processes, in: 23rd Int. Conference on Advanced Information Systems Engineering (CAiSE), Vol. 6741 of LNCS, Springer, 2011, pp. 551–565.
[49] J. White, D. Schmidt, D. Benavides, P.Trinidad, A. Ruiz-Cortés, Automated Diagnosis of Product-Line Configuration Errors in Feature Models, in: SPLC 2008, IEEE Computer Society, 2008, pp. 225–234.