Validation of Families of Business Processes

Gerd Gröner[1], Christian Wende[2], Marko Bošković[3], Fernando Silva Parreiras[1], Tobias Walter[1], Florian Heidenreich[2], Dragan Gašević[3], and Steffen Staab[1]

[1] WeST Institute, University of Koblenz-Landau, Germany
{groener,parreiras,walter,staab}@uni-koblenz.de
[2] Technische Universität Dresden, Germany
{c.wende,florian.heidenreich}@tu-dresden.de
[3] Athabasca University, Canada
{marko.boskovic,dragang}@athabascau.ca

Abstract. A Software Product Line (SPL) is a set of programs that are developed as a whole and share a set of common features. Product line's variability is typically specified using problem space models (i.e., feature models), solution space models that specify the realization of functionality and mapping models that link problem and solution space artifacts. In this paper, we consider this concept in the scope of families of business processes, whose specificity is that the solution space is defined with business process models. Solution space models are typically specified as model templates, and thus in the rest of the paper we will refer to business process model templates. While the previous research tackled the concepts of families of business processes, there have been very limited research on their validation.

Keywords: business process families, well-formedness constraints, validation, process model variability, configuration.

1 Introduction

The increasing number of software systems with similar required functionality has led software engineers to move from development of single software systems to the development of Software Product Lines (SPLs). A SPL is a set of software systems that share most of the features [1]. Because of the shared commonalities, development of families improves reusability and is more cost effective [2].

A SPL[1] is typically specified with three kinds of models: *problem space models*, *solution space models* and *mapping models* [3]. *Problem space models* define available features of the members of the SPL, as well as their interdependencies. They are typically used by stakeholders for selection of desired features of the product. The set of selected features is called *configuration*. *Solution space models* are comprehensive models that specify the realization of complete SPLs. In this paper, we focus on business process families, i.e., families whose solution space models are business process model templates. Business process model

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

H. Mouratidis and C. Rolland (Eds.): CAiSE 2011, LNCS 6741, pp. 551–565, 2011.
© Springer-Verlag Berlin Heidelberg 2011

templates are specified by business process modeling languages and composed of business process patterns. From solution space models, a particular product, i.e., a business process model is derived by removing or adding parts of it. Finally, *mapping models* define mapping relations between problem space and solution space models. They regulate which parts might be removed from the business process model according to the selected features from the problem space model.

Given such a representation of business process families, we have to guarantee that each process model, that is built according to a feature configuration, does not violate any well-formedness constraints of the business process model template. Due to the size of contemporary business process families, it is time consuming, costly and error-prone to manually validate that each configuration has a well-formed corresponding business process model. For these reasons, an automated approach for the constraint validation is necessary.

To address this problem, in this paper, we propose a classification of interrelationships between elements of business process models and demonstrate how this classification can be used for the validation. The classification is based on an analysis of basic workflow patterns, a set of conceptual basis for process languages. This classification is specified in Description Logics (DL), which we use as means of validating business process templates.

2 Application Context of Business Process Families

A typical SPL consists of three kinds of artifacts, representing its problem space, solution space and mappings between the problem and solution space [3]. We introduce one such SPL, a part of the Electronic Store (e-store) SPL [4].

Fig. 1 depicts a snippet of the business process family of the e-store case study. The representation contains a feature model to represent commonality and variability of the business process family, a business process model template, specified in the Business Process Modeling Notation (BPMN) and mappings between features and elements of the business process model template.

Interdependencies in the feature model are specified with *mandatory* and *optional* parent-child relationships and *alternative* and *or feature groups*. A *mandatory* parent-child relationship specifies that if a parent feature is selected in a certain configuration, its mandatory child feature has to be too (e.g., E-Shop and StoreFront). An *optional* parent-child relationship specifies a possibility of the selection, e.g., StoreFront and WishList. 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, ProductFlagging and AssignmentToPageTypesForDisplay) defines a set of features from which at least one has to be selected.

Feature models also contain interdependencies between features that are not captured by the tree structure of feature diagrams, called cross-tree constraints, 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., EmailWishList and Registration). Excludes is the opposite to includes.

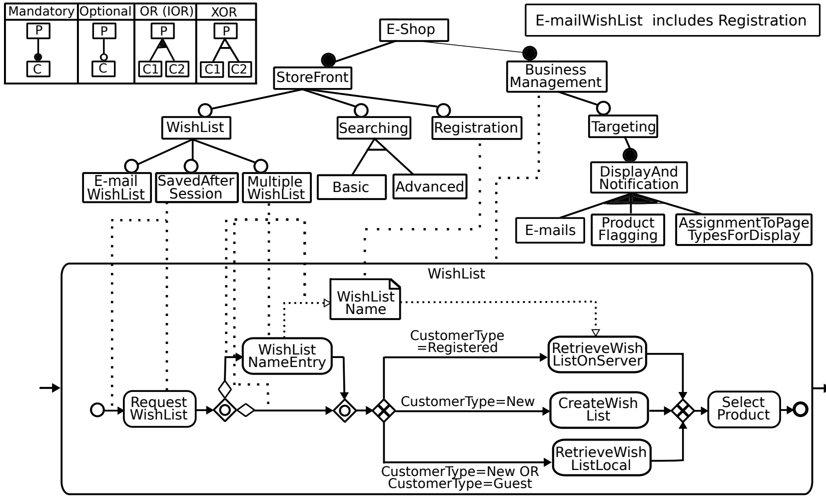

Fig. 1. A Part of the E-Store [4] Software Product Line

Business process model templates are specified in a business process modeling language. In Fig. 1, we use BPMN to specify a business process model template. Such a template typically consists of process patterns like subprocesses (WishList), activities (e.g., WishListNameEntry), gateways (diamonds), conditional sequence flows and data objects (WishListName). These process patterns impose well-formedness constraints like grouping of activities.

Finally, mappings connect the features with elements of the solution space model that implement the business logic of particular configurations. For example, every configuration that contains the Registration feature, contains the WishListName data object. On the contrary, every configuration that does not contain the feature MultipleWishList leads to a business process model where the corresponding mapped activity WishListNameEntry is missing. All elements of the business process model template that are not mapped to any feature are contained in every business process model.

The problem in this context is given by the constraints of both modeling spaces in combination with mappings between the feature model and the business process model template. Given a particular feature configuration and a mapping, we have to ensure, that the corresponding business process model satisfies the well-formedness constraints of the template.

3 Solution Space Model Dependencies and Validation

In this paper, Description Logic (DL) [5] is 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. Discussions of expansiveness of DL over some other options, although an important research topic, is outside of the scope of this paper.

3.1 Modeling with Description Logics

DL is a decidable subset of first-order logic (FOL). A DL-based knowledge base is established by a set of terminological axioms (TBox) and assertions (ABox). The TBox is used to specify classes, which denote sets of individuals and properties defining binary relations between individuals. The main syntactic constructs are depicted in Table 1, supplemented by the corresponding (FOL) expressions.

Table 1. Constructs and Notations in DL and FOL Syntax

Construct Name	DL Syntax	FOL Syntax
atomic class, atomic object property	C, R	$C(x)$, $R(x,y)$
subclass relation	$C \sqsubseteq D$	$\forall x.C(x) \rightarrow D(x)$
union class expression	$C_1 \sqcup \ldots \sqcup C_n$	$C_1(x) \vee \ldots \vee C_n(x)$
intersection class expression	$C_1 \sqcap \ldots \sqcap C_n$	$C_1(x) \wedge \ldots \wedge C_n(x)$
complement class expression	$\neg C$	$\neg C(x)$
universal quantification	$\forall P.C$	$\forall y.(P(x,y) \rightarrow C(y))$
existential quantification	$\exists P.C$	$\exists y.(P(x,y) \wedge C(y))$
object subproperty	$R \sqsubseteq S$	$\forall x,y.R(x,y) \rightarrow S(x,y)$

3.2 Representing Models of Business Process Families

In this section, we provide formal definitions of the three considered modeling spaces: (i) problem space, (ii) solution space and (iii) mapping space that combines features from the problem space with entities in the solution space.

Definition 1. *A Feature Model $\Phi = \langle \mathcal{F}, \mathcal{F}_P, \mathcal{F}_M, \mathcal{F}_{IOR}, \mathcal{F}_{XOR}, \mathcal{F}_{cr} \rangle$ is a tree structure that consists of features $\mathcal{F}$. $\mathcal{F}_p \subset \mathcal{F} \times \mathcal{F}$ is a set of parent and child feature pairs, $\mathcal{F}_M \subset \mathcal{F}$ is a set of mandatory features with their parents. All other features are optional. $\mathcal{F}_{IOR} \subset \mathcal{P}(\mathcal{F}) \times \mathcal{F}$ and $\mathcal{F}_{XOR} \subset \mathcal{P}(\mathcal{F}) \times \mathcal{F}$ are sets of pairs of child features and its common parent feature. The child features are either inclusive or exclusive features. Finally, $\mathcal{F}_{cr} \subset \mathcal{F} \times \mathcal{F}$ is a set of cross-tree constrained feature pairs that are either in an* includes *or* excludes *relation.*

$$E-Shop \equiv \exists\ hasFeature.StoreFront \sqcap \exists hasFeature.BusinessManagement \quad (1)$$

$$StoreFront \sqsubseteq \exists\ parent.E-Shop \quad (2)$$

$$Searching \equiv (\exists\ hasFeature.Basic \sqcup\ \exists\ hasFeature.Advanced) \sqcap \neg(\exists\ hasFeature.Basic \sqcap \exists\ hasFeature.Advanced) \quad (3)$$

$$EmailWishList \sqsubseteq \exists includes.Registration \quad (4)$$

Axiom 1 defines StoreFront and BusinessManagement as mandatory child features of E-Shop. There is no such axiom for optional child features. Except of the root feature, each feature has a parent feature. Axiom 2 defines E-Shop as the

parent feature of StoreFront. For inclusive and exclusive child features, a class union is used. For instance, Axiom 3 defines Basic and Advanced as exclusive child features of the parent feature Searching. The axiom ensures that only one feature is selected (cf. [6]). Axiom 4 depicts a cross-tree constraint where the feature EmailWishList includes Registration.

The solution space is defined by a business process model template. In Def. 2, we give generic definitions for solution space models. We focus on elements that are mapped to features, i.e., these elements realize a certain feature. Obviously, the granularity of mapping solution space models (e.g., classes, attributes) depends on the applications. In the reset of this paper, all elements that are subclasses of the BPMN class Element can be mapped to features.

Definition 2. *A Solution Space Model $\Omega = \langle \mathcal{S}, \mathcal{T} \rangle$ consists of entities $\mathcal{S}$ that could be mapped to features and entities that are not mapped and do not directly realize features ($\mathcal{T}$). The sets $\mathcal{S}$ and $\mathcal{T}$ are disjoint.*

Concrete process models found in the solution space are automatically transformed to a knowledge base Σ_Ω. The transformation creates classes for constructs of the BPMN metamodel, like the classes Activity and SequenceEdge, independently of the concrete BPMN model template. All elements of the BPMN model template are modeled as subclasses of Element (i.e., elements of $\mathcal{S}$).

$$Vertex,\ Activity,\ SequenceEdge \sqsubseteq Element \quad (5)$$

$$RequestWishList \sqsubseteq Activity \quad (6)$$

$$E1 \sqsubseteq SequenceEdge \quad (7)$$

$$RequestWishList \sqsubseteq \exists outgoingEdge.E1 \quad (8)$$

Vertex, Activity and SequenceEdge are subclasses of Element (Axiom 5). RequestWishList is a concrete activity, defined by a subclass axiom too (Axiom 6). Likewise, an edge E1 in the process model is represented by a subclass axiom (Axiom 7). The relations from activities to edges are described using existential restrictions on the object property outgoingEdge (Axiom 8).

Mappings (Def. 3) connect features $\mathcal{F}$ and the elements $\mathcal{S}$ of the solution space model Ω. A feature can be mapped to multiple elements and likewise an element in the solution space might realize multiple features.

Definition 3. *For a feature model $\Phi = \langle \mathcal{F}, \mathcal{F}_P, \mathcal{F}_M, \mathcal{F}_{IOR}, \mathcal{F}_{XOR}, \mathcal{F}_{cr} \rangle$ and a solution space model $\Omega = \langle \mathcal{S}, \mathcal{T} \rangle$, a Mapping Model M is a relation $M \subseteq \mathcal{F} \times \mathcal{S}$, that is defined as $\mathcal{F} \times \mathcal{S} := \{(f, s) : f \in \mathcal{F} \wedge s \in \mathcal{S}\}$.*

Finally, we transform the mapping models into a TBox Σ_M. For each mapped element $E \in \mathcal{S}$ in Ω, we introduce a class Map_E that is a subclass of Map. The object property feature is used to describe the mappings from elements to features. A mapping from an element E to a feature F is represented by an axiom $Map_E \sqsubseteq \exists feature.F$ (Axiom 9). The mapping classes Map_E are introduced to separate the feature mapping from the solution space. In order to ease the validation later on, we define the property $hasFeature$ as a subproperty of the property $feature$ form the mapping model (Axiom 10).

$$Map_{RequestWishList} \sqsubseteq \exists\ feature\ .\ SavedAfterSession \tag{9}$$
$$hasFeature \sqsubseteq feature \tag{10}$$

3.3 Well-Formedness in the Business Process Model Templates

Besides solution space models, we have to represent syntactic and structural well-formedness constraints of the process models. They are imposed by the BPMN language and by basic workflow patterns, as described in [7].

We focus in our work on structural well-formedness for two reasons. Firstly, for the class of structural models, structural constraints coincide with behavioral constraints (see the work on behavioral profiles that are derived from process structure trees [8]). Secondly, there are techniques to derive structured models for a broad class of unstructured models [9].

Elements of the business process model template that are not mapped to any feature occur in each process model. In contrast, the appearance of a mapped element in a process model depends on the feature configuration. For the sake of a more compact representation, we introduce auxiliary constructs. In Axiom 11, each element with at least one mapping to a mapping class is defined by the class *MappedElement*. Axioms 12, 13 and 14 define (composite) properties.

$$MappedElement \sqsubseteq Element \sqcap \exists mapped.Map \tag{11}$$
$$incomingEdge\ \circ\ source \sqsubseteq predecessor \tag{12}$$
$$outgoingEdge\ \circ\ target \sqsubseteq successor \tag{13}$$
$$successor\ \circ\ predecessor \sqsubseteq sibling \tag{14}$$

We identify three types of constraints. (i) An element might *require* another element. (ii) An element always appear in *conjunction* with another element. (iii) Elements of a process model might *exclude* each other. There is no influence on elements in inclusive or-branches by feature mappings, since well-formedness violations only occur due to missing elements in the process model.

We introduce three classes *Required*, *Conjunct* and *Exclude* to capture elements according to the different constraints. For instance, if element E is classified as a required element, we expect that E is subsumed by the (super-) class *Required*. Additionally, a property *isRequired* is introduced to get for an element its required element. The corresponding element E' that requires E is obtained by a derivable axiom like $E \sqsubseteq \exists isRequired.E'$. Likewise, we can use the property *sibling* (Axiom 14) to get the sibling activities.

Sequence. A sequence describes a series of activities that are connected by sequence edges. In a BPMN model, there are no dangling edges allowed, i.e., if there is a mapping to an activity, a violation might occur, if the activity is missing. Let us consider the mapping of the optional feature $MultipleWishList$ to the activity $WishListNameEntry$. In this case, we have to guarantee that sequence edges require their corresponding source and target vertex. Axiom 15 defines each mapped activity with an incoming or outgoing sequence edge as *required*. The corresponding properties *incomingEdge* and *outgoingEdge* are defined as subproperties of the introduced property *isRequired* (Axiom 16).

$$MappedElement \sqcap (\exists\ incomingEdge.SequenceEdge \sqcup \exists\ outgoingEdge.SequenceEdge) \sqsubseteq Required \quad (15)$$

$$incomingEdge,\ outgoingEdge \sqsubseteq isRequired \quad (16)$$

Container Element. BPMN models facilitate grouping and decomposition of activities. Axiom 17 describes that container elements such as groups and subprocesses are supposed to be required by their elements. Groups in a BPMN model use the property *activities* to define which activities are members of this group, while subprocesses use the properties *vertices* and *sequenceEdges*. These properties are defined as subproperties of *isRequired* in order to have a connection to the element's counterparts (Axiom 18).

$$MappedElement \sqcap (\exists\ activites.Element \sqcup \exists\ vertices.Element \sqcup \exists\ sequenceEdges.Element) \sqsubseteq Required \quad (17)$$

$$activities,\ vertices,\ sequenceEdges \sqsubseteq isRequired \quad (18)$$

Conditional Flow. Activities with outgoing conditional sequence edges impose the existence of at least one unconditional outgoing sequence edge. This constraint might be violated in the case an unconditional sequence edge is mapped and could be possibly removed in a certain configuration. For instance, consider the mapping of the optional feature $MultipleWishList$ to the conditional outgoing edges of the activity $RequestWishList$ in Fig. 1. In this case, a feature configuration could result in a process model without any outgoing unconditional edge. In order to avoid this, we describe in Axiom 19 that a mapped unconditional edge which is the only outgoing edge is required.

$$MappedElement \sqcap UnConditionalEdge \sqcap \exists\ source.\ (Activity \sqcap \exists_{\leq 1} outgoingEdge.UnConditionalEdge) \sqsubseteq Required \quad (19)$$

$$source\ \circ\ outgoingEdge \sqsubseteq isRequired \quad (20)$$

Opening Gateway. An opening gateway is used to express the divergence of a control flow. While the number of outgoing branches is arbitrary in general, it is required that there are at least two outgoing sequence edges. Otherwise the gateway does not represent any divergence. Axiom 21 defines a mapped sequence edge that is one of at most two outgoing edges as a required element. Axiom 22 defines the property composition of *source* and *outgoingEdge* as subproperty of *isRequired* in order to find the counterparts.

$$MappedElement \sqcap SequenceEdge \sqcap \exists source.(OpeningGateway \sqcap \exists_{\leq 2} outgoingEdge.SequenceEdge) \sqsubseteq Required \quad (21)$$

$$source\ \circ\ outgoingEdge \sqsubseteq isRequired \quad (22)$$

Closing Gateway. The convergence of a control flow in a process is described by closing gateways. Like for opening gateways, a closing gateway needs at least two incoming sequence edges. Axioms 23 and 24 are similar to the previous Axioms just for closing gateway.

$$MappedElement \sqcap SequenceEdge \sqcap \exists target.(ClosingGateway \sqcap \exists_{\leq 2} incomingEdge.SequenceEdge) \sqsubseteq Required \quad (23)$$

$$target \circ incomingEdge \sqsubseteq isRequired \quad (24)$$

Exclusive Branching. Activities that appear in exclusive branches are supposed to be exclusive, i.e., it is not allowed to execute activities from alternative branches in a process execution. Axiom 25 defines each mapped element between XOR-gateways as a subclass of Exclude.

$$MappedElement \sqcap \exists successor.XORgateway \sqcap \exists predecessor.XORgateway \sqsubseteq Exclude \quad (25)$$

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. In Axiom 26, we define elements between AND-gateways as subclasses of the class *Conjunct*.

$$MappedElement \sqcap \exists successor.ANDgateway \sqcap \exists predecessor.ANDgateway \sqsubseteq Conjunct \quad (26)$$

4 Validation Using Description Logics

Our aim is to ensure that the well-formedness constraints are satisfied in all process models that can be derived from a process model template. The constraints are represented by logical formulas, DL expressions in our case. We use the expression f_Φ to represent a constraint on features and f_Ω to represent a constraint on elements of the business process. From a logical point of view, checking whether the constraint given by f_Φ ensures that the constraints represented by f_Ω hold, can be realized by checking whether the implication $f_\Phi \Rightarrow f_\Omega$ holds for each interpretation, i.e., $f_\Phi \Rightarrow f_\Omega$ is a tautology.

In order to test the implication $f_\Phi \Rightarrow f_\Omega$ for all mapped elements, we have to tackle the following problems. (i) While the constraints in the problem space are directly represented by the axioms, the constraints of the process model template are implicitly given by the element's dependencies (cf. Sect. 3). Hence, we have to *classify and derive* constraints of the mapped elements. (ii) We have to guarantee that the same vocabulary and the same expression structure is used in f_Φ and f_Ω. This is realized by *building constraint expressions.* (iii) Finally, we have to check in DL whether the *implication* holds.

In Sect. 3.3, we specify implicit constraints of business processes and define axioms in order to allow a dynamic classification of mapped and constrained elements. These elements are categorized as subclasses of the classes Required, Conjunct and Exclude. Defined properties (sibling and isRequired) as well as their inverse properties help to get their counterparts. E.g., for the conjunct element E, the counterpart (E') can be found by using the subsumption $E \sqsubseteq \exists sibling.E'$.

Building Constraint Expressions. For the mapped elements, we build class expressions f_Φ and f_Ω. To check the implication (subsumption in DL ($f_\Phi \sqsubseteq f_\Omega$)),

we describe f_Φ and f_Ω by complex class expressions in DL. Additionally, we introduce a class expression f_Ψ which will be used in the next step (Def. 4).

Depending on the constraint type (*Conjunct*, *Required* or *Exclude*) of the element E, the expression f_Ω is built either as intersection, implication or exclusive class expression. However, instead of the element E, we use the corresponding mapping class Map_E (cf. Axiom 9) of the mapping model and the property *feature* that is a superproperty of *hasFeature* (cf. Axiom 10). This guarantees the alignment of classes and properties between Σ_Φ and Σ_Ω.

For the feature model, f_Φ is the intersection of parents and cross-tree constraints of all mapped features ($\mathcal{F}$) of E. The absence of optional child features in the parent definition of the feature model (Axioms 1-3) directly meets the need of the feature representation in f_Φ, since for an optional mapped feature, we can not guarantee the appearance of the corresponding element in each business process model. The set $\mathcal{F}$ of mapped features F is obtained from the mapping knowledge base Σ_M by axioms like $Map_E \sqsubseteq \exists feature.F$. Cross-tree constraints are captured by the expression $cr(F)$. To allow a subsumption checking of the expressions in $cr(F)$, we define the properties *includes* and *excludes* as subproperties of *feature*. The functions *elements* and *element* are abbreviations. The element(s) is/are either the element that require another element or sibling elements, they can be found by using the introduced properties *isRequired* and *sibling*.

Definition 4. *The final knowledge base Σ is constructed from the problem, solution and mapping space knowledge bases, i.e., $\Sigma := \Sigma_\Phi \cup \Sigma_\Omega \cup \Sigma_M$. Moreover, for each mapped and constrained element, an axiom $f_\Psi \equiv \neg f_\Phi \sqcup f_\Omega$ is added to Σ, where f_Φ and f_Ω are defined as follows:*

- *for each element $E \sqsubseteq Conjunct$ and $\mathcal{S} := elements(sibling, E)$:* $f_\Omega \equiv \bigsqcap_{E' \in \mathcal{S}} Map_{E'}$ *and* $f_\Phi \equiv \bigsqcap_{F \in \mathcal{F}} Parent(F) \sqcap cr(F)$
- *for each element $E \sqsubseteq Required$ and $E' := element(isRequired, E)$:* $f_\Omega \equiv \neg Map_{E'} \sqcup Map_E$ *and* $f_\Phi \equiv \bigsqcap_{F \in \mathcal{F}} Parent(F) \sqcap cr(F)$
- *for each element $E \sqsubseteq Exclude$ and $\mathcal{S} := elements(sibling, E)$:* $f_\Omega \equiv \bigsqcup_{E' \in \mathcal{S}} Map_{E'} \sqcap \neg \bigsqcup (Map_{E''} \sqcap Map_{E'''})$ *for* $(E'', E''' \in \mathcal{S})$ *and* $f_\Phi \equiv \bigsqcap_{F \in \mathcal{F}} Parent(F) \sqcap cr(F)$

Implication Checking. We reduce subsumption checking $f_\Phi \sqsubseteq f_\Omega$ to a classification problem by introducing f_Ψ. According to Def. 4, we add for each subsumption checking problem the corresponding axiom $f_\Psi \equiv \neg f_\Phi \sqcup f_\Omega$. Finally, we check for each class expression f_Ψ whether it is equivalent with the top class ($f_\Psi \equiv \top$). In this case, the solution space constraints (f_Ω) are satisfied, otherwise there is a violation. Moreover, f_Ω is a subclass of the corresponding mapping class Map_E. This directly indicates the element that violates the constraint.

The validation effort is determined by the number of mappings and the number of elements that are involved in one of the constraints (Required, Conjunct and Exclusive). There are two steps where reasoning is applied. (i) We classify the knowledge base in order to find the constrained elements. To build the class

expressions f_Ω (for each mapping one expression), we use class subsumption to get the counterparts. For this purpose, we have to iterate over each element that is a subclass of Required, Conjunct and Exclusive, but without any further classification. (ii) The second step is a further knowledge base classification, in order to find those expressions f_Ψ that are not equivalent to the top class $\top$. Class subsumption and classification are both standard reasoning services that are quite tractable in practice. The DL expressivity is $\mathcal{SHOIN}$.

5 Correctness of the Validation

This section demonstrates the correct capturing of the constraints in DL by the implication $f_\Phi \Rightarrow f_\Omega$. We start with an consideration of the constraint coverage by these expressions. Afterwards, we show that well-formedness of the business process model template can be concluded from the implication checking.

Constraint Coverage. In our case, we know that both models are correct on its own. Hence, a violation of the well-formedness constraints can only be caused by the mappings. Our aim is to guarantee the well-formedness of each process model from the business process model template, for each valid feature configuration. We consider different cases how an element E might be involved in a feature mapping. In case the element is not mapped, there might be no violation, since the constraint types only contains mapped elements. E remains in each process model and is not involved in any constraint with another element.

If E is mapped to at least one feature F, it depends whether E is involved in a well-formedness constraint. In case E is not involved, there cannot be any violation of a well-formedness constraint, due to the same reasons as for unmapped elements. More difficult is the case when E is involved in one of the constraints. The constraint expression f_Ω (Def. 4) encodes the corresponding constraint of E with its counterpart elements, e.g., sibling elements. The intention of f_Ω is, that the encoded constraint has to be satisfied in all business process models. Hence, we have to check whether f_Ω holds for each feature configuration.

Again, we know that without any mapping there is no violation in the business process model template. Therefore, we know that only the constraints of the mapped feature F might lead to a violation of f_Ω. The expression f_Φ captures these constraints of all mapped features F of E. We build f_Φ as a conjunction on all these features (cf. Def. 4) to capture the case that there are multiple mappings of one element E. The constraints of the features are directly given by the definition of the parent features and the cross-tree constraints of F.

The alignment is solved by a design decision of the mapping model (cf. Sect. 3.3). The property $feature$ maps an element E to a feature F, by an axiom $Map_E \sqsubseteq \exists feature.F$. The property $hasFeature$ is defined as a subproperty of $feature$ from the mapping space (Axiom 10). Hence, all class expressions from the problem space using the property $hasFeature$ are subsumed by expressions where this property is replaced by its superproperty $feature$. In Def. 4, we use Map_E instead of elements E in the expression f_Ω. Hence, f_Φ and also f_Ω

only contain classes of the feature model and $hasFeature$ and $feature$ are in a subproperty relation. The expression f_Ω is composed of the mapped elements (Map_E of Σ_M) according to the logical meaning of their constraint classification ($Required$, $Conjunct$ or $Exclude$).

Formula Representation in DL. In the validation, we check whether f_Ψ ($f_\Psi \equiv \neg f_\Phi \sqcup f_\Omega$) is equivalent with $\top$, which means to test whether the subsumption $f_\Phi \sqsubseteq f_\Omega$ holds for each interpretation (tautology). Due to the alignment, we can compare f_Φ and f_Ω by DL reasoning. Finally, we have to demonstrate that this subsumption ensures that the solution space constraints are satisfied for each allowed feature configuration (Lemma 1).

Lemma 1 (Correctness of the Validation). *For mappings from an element E to a set of features $\mathcal{F}$, f_Φ are the constraints of $\mathcal{F}$ and f_Ω the constraints of E. If f_Φ is subsumed by f_Ω then the well-formedness constraints of all elements E hold.*

Proof. Looking to the different types of constraints in both spaces, we basically deal with *implication*, *and*, *or* and *xor*. Hence, we have to consider all possible combinations in both spaces and check whether $f_\Phi \Rightarrow f_\Omega$ is a tautology. This kind of logical problem is in the nature of propositional logic. Hence in Def. 4, we define the DL expressions f_Φ and f_Ω 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 class expressions like $\exists hasFeature.F$ containing features and the properties $feature$ and $hasFeature$ from Σ_Φ and Σ_M. It is easy to see in Def. 4 that f_Ω is built as a DL expression representing either a conjunctive, exclusive or implicative combination. In f_Φ, we conjunctively connect the parent features and the cross-tree constraints that are already represented in this modeling style.

6 Proof-of-Concept and Discussion

The evaluation of our approach has been conducted by providing a proof-of-concept which has been developed by integrating the FeatureMapper [10] and the transformation of the control flow parts of BPMN to DL, as described in [11].

Setting. We applied the validation to the case study that was introduced in Sect. 2 and is part of the e-store SPL [4]. The feature model consists of 287 features, 2 top features, 192 of the features are leaf features and all others are parent features. There are 21 cross-tree references, including mandatory and optional as well as OR-grouped features. The process model contains 84 activities.

In the settings, we validated feature models with 154 features and with the entire feature model (287 features). In both cases, we build either 22 or 48 mappings. The average validation time using the Pellet reasoner is 2970 ms for 154 features with 22 mappings and 4430 ms for 287 features with 48 mappings. The time for the transformation to DL is less that the validation time. This is based on the fact that we use the DL-oriented feature model of [6] and we only transform the relevant control flow informations of BPMN to DL.

Validation Exemplified. We demonstrate the validation of one mapping for an easy example from the case study excerpt of Fig. 1. We assume a mapping from the subprocess $WishList$ to the mandatory feature $BusinessManagement$. The mapping is represented in Σ by an axiom $Map_{WishList} \sqsubseteq \exists feature.BusinessManagement$. We expect no constraint violations since the feature $BusinessManagement$ is mandatory. Concerning the validation, f_Φ is build using the parent definition, i.e., $f_\Phi \equiv \exists hasFeature.StoreFront \sqcap \exists hasFeature.BusinessManagement$. The expression f_Ω is build using the class $Map_{WishList}$ from the mapping model ($f_\Omega \equiv \exists feature.BusinessManagement$). For the subsumption checking of f_Φ by f_Ω, we can replace the property $feature$ by $hasFeature$. It is easy to see that due to the negation of f_Φ, f_Ψ is equivalent to the top class $\top$: $f_\Psi \equiv \forall hasFeature.\neg StoreFront \sqcup \forall hasFeature.\neg BusinessManagement \sqcup \exists hasFeature.BusinessManagement$. In case a particular mapping causes a violation, the user finds the corresponding constraint expression f_Ψ classified as not equal to the top class ($\top$).

Lessons Learned. In Sect. 3.3, we already distinguished the focus of our work on well-formedness constraints to the work on behavioral profiles of [12]. After a deeper comparison of both formalisms, we find two interesting aspects that directly impose further research challenges. Firstly, behavioral profiles are efficient to compare process behavior and behavior consistency. It might be a promising step to extend our well-formedness constraint validation towards a behavior constraint validation, while we still offer the same feature-oriented configuration view in combination with the mappings. Secondly, in this context, we see potential on using DL for the validation. The main challenge from the DL modeling perspective is to handle the possibility of concurrent executions of activities. This problem seems to be in line with the descriptive modeling style of DL to capture this kind of execution potentiality.

7 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. [13], Boffoli et al. [14], La Rosa et al. [15] and van der Aalst et al. [16]. Schnieders et al. [13] 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. [14] and La Rosa et al. [15] also distinguish between business process models and problem space 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.

More similar to our objective is the approach for process configuration from van der Aalst et al. [16]. 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.

Weidlich et al. [8,12] derive behavior profiles to describe the essential behavior in terms of activity relations like exclusivity, interleaving and ordering of activities. Weber et al. [17] 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.

In the context of SPLs several approaches have been introduced, in order to ensure the well-formedness of solution space models. Czarnecki et al. [18] 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. [19] 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. [20] and van der Storm [21] introduce approaches to validate the correctness of mappings between feature and component models. They use propositional logics too.

8 Conclusion

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

Our proposal validates business process models w.r.t. their well-formedness constraints; mappings to problem space models; and dependencies in the problem space 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 patterns. 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 patterns used in this paper and control flow support analyzes presented in the relevant literature [23].

We evaluated our work with the largest publicly-available case study, for which we were able to find all the three types of models. While this case study has a realistic size, we would like to have a benchmarking framework which will allow for simulating larger solution space and mapping models. This is similar to what has been already proposed for feature models [24], but now to be enriched for the generation of business process model templates of different characteristics. We plan to organize a user study where the proposed approach will be evaluated by asking software modelers to complete some tasks by applying our tooling. A further plan is to extend our validation formalism towards behavioral constraints in the business process model template.

Acknowledgements. This work has been supported by the EU Project MOST (ICT-FP7-2008 216691).

References

1. Pohl, K., Böckle, G., van der Linden, F.J.: Software Product Line Engineering: Foundations, Principles and Techniques. Springer, Heidelberg (2005)
2. McGregor, J., Muthig, D., Yoshimura, K., Jensen, P.: Successful Software Product Line Practices. IEEE Software 27(3), 16–21 (2010)
3. Czarnecki, K., Eisenecker, U.W.: Generative Programming: Methods, Tools, and Applications. ACM Press, New York (2000)
4. Lau, S.Q.: Domain Analysis of E-Commerce Systems Using Feature-Based Model Templates. Master's thesis, University of Waterloo, Waterloo (2006)
5. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P.: The Description Logic Handbook. Cambridge University Press, Cambridge (2007)
6. Wang, H., Li, Y., Sun, J., Zhang, H., Pan, J.: Verifying Feature Models using OWL. J. of Web Semantics 5(2), 117–129 (2007)
7. van der Aalst, W., ter Hofstede, A., Kiepuszewski, B., Barros, A.: Workflow Patterns. In: Distributed and Parallel Databases (2003)
8. Weidlich, M., Polyvyanyy, A., Mendling, J., Weske, M.: Efficient Computation of Causal Behavioural Profiles Using Structural Decomposition. In: Lilius, J., Penczek, W. (eds.) PETRI NETS 2010. LNCS, vol. 6128, pp. 63–83. Springer, Heidelberg (2010)
9. Polyvyanyy, A., García-Bañuelos, L., Dumas, M.: Structuring Acyclic Process Models. In: Hull, R., Mendling, J., Tai, S. (eds.) BPM 2010. LNCS, vol. 6336, pp. 276–293. Springer, Heidelberg (2010)
10. Heidenreich, F., Kopcsek, J., Wende, C.: FeatureMapper: Mapping Features to Models. In: ICSE 2008 Companion, pp. 943–944. ACM, New York (2008)
11. Ren, Y., Gröner, G., Lemcke, J., Rahmani, T., Friesen, A., Zhao, Y., Pan, J.Z., Staab, S.: Validating Process Refinement with Ontologies. In: Description Logics. CEUR Workshop Proceedings, CEUR-WS.org, vol. 477 (2009)
12. Weidlich, M., Mendling, J., Weske, M.: Efficient Consistency Measurement Based on Behavioural Profiles of Process Models. IEEE Transactions on Software Engineering 99 (2010)
13. Schnieders, A., Puhlmann, F.: Variability Mechanisms in E-Business Process Families. In: BIS 2006: 9th Int Conf. on Business Information Systems, pp. 583–601 (2006)

14. Boffoli, N., Cimitile, M., Maggi, F.M.: Managing Business Process Flexibility and Reuse through Business Process Lines. In: Cordeiro, J., Ranchordas, A., Shishkov, B. (eds.) ICSOFT 2009. Communications in Computer and Information Science, vol. 50, pp. 61–68. Springer, Heidelberg (2011)
15. Rosa, M.L., van der Aalst, W.M.P., Dumas, M., ter Hofstede, A.H.M.: Questionnaire-based Variability Modeling for System Configuration. SoSyM 8(2), 251–274 (2009)
16. van der Aalst, W.M.P., Dumas, M., Gottschalk, F., ter Hofstede, A.H.M., Rosa, M.L., Mendling, J.: Preserving Correctness during Business Process Model Configuration. Formal Asp. Comput. 22(3-4), 459–482 (2010)
17. Weber, I., Hoffmann, J., Mendling, J.: Beyond Soundness: On the Verification of Semantic Business Process Models. Distributed and Parallel Databases 27(3), 271–343 (2010)
18. Czarnecki, K., Pietroszek, K.: Verifying Feature-Based Model Templates Against Well-Formedness OCL Constraints. In: GPCE 2006, pp. 211–220. ACM, New York (2006)
19. Thaker, S., Batory, D., Kitchin, D., Cook, W.: Safe Composition of Product Lines. In: GPCE 2007, pp. 95–104. ACM, New York (2007)
20. Janota, M., Botterweck, G.: Formal Approach to Integrating Feature and Architecture Models. In: Fiadeiro, J.L., Inverardi, P. (eds.) FASE 2008. LNCS, vol. 4961, pp. 31–45. Springer, Heidelberg (2008)
21. van der Storm, T.: Generic Feature-Based Software Composition. In: Lumpe, M., Vanderperren, W. (eds.) SC 2007. LNCS, vol. 4829, pp. 66–80. Springer, Heidelberg (2007)
22. Mohabbati, B., Hatala, M., Gašević, D., Asadi, M., Bošković, M.: Development and Configuration of Service-Oriented Systems Families. In: Proceedings of the 26th ACM Symposium on Applied Computing (2011)
23. Wohed, P., van der Aalst, W.M.P., Dumas, M., ter Hofstede, A.H.M., Russell, N.: On the suitability of BPMN for business process modelling. In: Dustdar, S., Fiadeiro, J.L., Sheth, A.P. (eds.) BPM 2006. LNCS, vol. 4102, pp. 161–176. Springer, Heidelberg (2006)
24. White, J., Schmidt, D., Benavides, D., Trinidad, P., Ruiz-Cortés, A.: Automated Diagnosis of Product-Line Configuration Errors in Feature Models. In: SPLC 2008, pp. 225–234. IEEE Computer Society, Los Alamitos (2008)