

International Journal of Software Engineering and Knowledge Engineering
© World Scientific Publishing Company

Automated Staged Configuration with Semantic Web Technologies*

MARKO BOŠKOVIĆ

*School of Computing and Information Systems, Athabasca University, 1 University Drive
Athabasca, Alberta T9S 3A3, Canada
marko.boskovic@athabascau.ca*

EBRAHIM BAGHERI

*Institute for Information Technology, National Research Council Canada, 46 Dineen Drive,
Fredericton, New Brunswick E3B 9W4, Canada
ebrahim.bagheri@nrc-cnrc.gc.ca*

DRAGAN GAŠEVIĆ

*School of Computing and Information Systems, Athabasca University, 1 University Drive
Athabasca, Alberta T9S 3A3, Canada
dgasevic@acm.org*

BARDIA MOHABBATI

*School of Computing and Information Systems, Athabasca University, 1 University Drive
Athabasca, Alberta T9S 3A3, Canada
and
School of Interactive Arts and Technology, Simon Fraser University, 13450 102 Avenue
Surrey, British Columbia V3T 5X3, Canada
mohabbati@sfu.ca*

NIMA KAVIANI

*Department of Electrical and Computer Engineering, University of British Columbia, 2356
Main Mall
Vancouver, British Columbia V6T 1Z4, Canada
nimak@ece.ubc.ca*

MAREK HATALA

*School of Interactive Arts and Technology, Simon Fraser University, 13450 102 Avenue
Surrey, British Columbia V3T 5X3, Canada
mhatala@sfu.ca*

Received (Day Month Year)
Revised (Day Month Year)
Accepted (Day Month Year)

*The research results present in this paper are at least in part supported by the Alberta Ingenuity through the New Faculty Award program.

2 *M. Bošković, E. Bagheri, D. Gašević, B. Mohabbati, N. Kaviani, M. Hatala*

Abstract. Since the introduction in the early nineties, feature models receive a great attention in industry and academia. Industrial success stories in applying feature models for modeling software product lines, and using them for configuring software-intensive systems motivate academia to discover ways to integrate different feature dependencies into the feature model, and automate verified feature configuration. In this paper we demonstrate how ontologies and Semantic Web technologies facilitate seamless integration of required external services and deployment platform capabilities into the feature model. Furthermore, we also contribute with an algorithm for automating staged configuration using Semantic Web reasoners to discover unfeasible features of the feature model.

Keywords: Ontologies, Semantic Web, Feature Modeling, Software Product Lines Engineering

1. Introduction

In the increasing demand for smaller time to market of high quality software-intensive systems, Software Product Line Engineering (SPLE) [1] has emerged as an industrially proven successful paradigm for software development. SPLE argues development of product lines (software families), sets of software intensive systems with similar functionality [5]. Software product lines typically share the same set of reusable assets and whose commonality and variability is presented in the form of feature models. Feature models consist of feature diagrams, tree like structures of possible final product features, and additional documentation such as policies, feature descriptions, feature integrity constraints such as when one feature requires the selection of another, or when one feature requires absence of another one, features' requirements for some services and platform characteristics etc. [7]. Feature models are used for selecting a product from the product family which meets the stakeholder's requirements. The desired product is being selected by choosing the desired features of that product from the feature model. The process of selecting desired features is called product configuration [7].

Feature configuration is a complex task and requires automation support. Product configuration is typically guided by stakeholder's needs, policies, and available deployment platforms and services. At the present moment, the product configuration is mostly done manually. Manual product configuration requires selection of desired features according to the stakeholder's needs and policies and manual verification of integrity constraints and satisfaction of features' requirements for provision of some services and/or platform characteristics. Industrial experience shows that the number of a product line features can grow up to several thousands [6]. In such large decision space manual verification of integrity constraints and satisfaction of features' requirements is complicated and error prone task. Therefore, automation of verification is of crucial importance for valid product delivery.

In this paper we complement the feature model configuration verification approach of Wang et al. [2] with automated staged configuration. Wang et al. introduce an approach for verification of feature model configuration and integrity constraints. However, their approach does not facilitate the verification of features requirements

for platform characteristics and/or services provision satisfaction. In our approach this is facilitated with staged configuration [7]. Staged configuration is an approach for product configuration where a series of consecutive steps of unfeasible features removal is performed prior to the final product configuration. Each specialization stage yields from an input feature model a new feature model with reduced number of features. In our approach we base this removal on the provided services and/or platform characteristics. Each feature that requires some service and/or platform characteristic which is not provided is being removed from the final feature model. At the end of the staged configuration, the final product is configured by selecting the desired features from the most specialized feature model.

In our approach we additionally exploit the fact that Wang et al. base their configuration verification approach on representing feature model as a Web Ontology Language (OWL) ontology, and demonstrate how features' requirements for services and/or deployment platform characteristics, i.e. hard requirements, can be seamlessly integrated into the feature model. Ontologies are community established conceptualizations used for sharing knowledge about the domain of interest. They are a convenient tool for specification of external services and deployment platform characteristics since they facilitate specification of terminology common to both, service/platform providers and users. The OWL language enables integration of information from different ontologies, and as such serves as a fine tool for specification of features' hard requirements, with community established ontologies such as Delivery Context Ontology [10].

Finally, we provide an algorithm for automatic feature model specialization based on specification of provided services and deployment platform characteristics. The algorithm uses standard description logic reasoning mechanisms for the detection of unfeasible features, i.e., features for which hard requirements are not satisfied, and removes them from the feature model. Additionally, we prove that the algorithm is complete, i.e. that it removes all unfeasible features, and that the yielded specialized feature model is sound, i.e., it can be used as a new feature model.

The remainder of the paper is structured as follows. Section 2 gives an overview of the role and means of feature modeling in software family development. The proposed approach for specification of features' hard constraints specification and staged configuration is described in Section 3. The details about the features' hard constraints specification are given in Section 4. Section 5 gives the algorithm for automatic feature model specialization and proves its completeness and soundness of the specialized feature model. The algorithm is illustrated with an example in Section 6. Related work is given in Section 7. Finally, Section 8 gives the directions in which our future research will go.

4 M. Bošković, E. Bagheri, D. Gašević, B. Mohabbati, N. Kaviani, M. Hatala

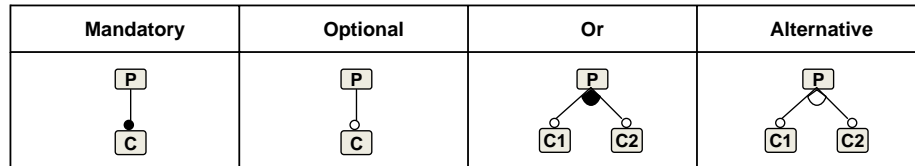


Fig. 1. Feature Relations of a Feature Diagram

2. Feature Models

Software product lines or software factories are sets of software intensive systems with similar functionality [5]. Members of a software family are distinguished between themselves by the features that characterize them. Features are user-visible characteristics of the domain for which the software product line is intended [8]. Commonality and variability of a software product line is captured through feature models. Feature models are used for selecting the desired features, i.e. product configuration.

Each feature models consist of a feature diagram, and other additional information such as integrity constraints, services that features require, rationale etc. A feature diagram is a tree structure with a predefined set of relations between parent and child features. The most common relations between parent and its child features are presented in Figure 1.

Given that the parent feature **P** is selected, the semantics of the relations presented in Table 1 is as follows:

- Mandatory feature **C** is a child feature that must also be selected in the configuration,
- Optional feature **C** is a child feature that must but does not have to be selected in the configuration,
- Or feature group is a group of child features from one or more must be selected in the final configuration,
- Alternative feature group is a group of child features from which only one can be selected in the final configuration.

An illustration of the feature diagram of a client application product line of an international airport Ubiquitous Information System (UIS) is depicted in Figure 2. Members of this product line facilitate provision of news reports to passengers at the airport. In order to be able to receive the news reports, a passenger must request a client application from the server. After the request is provided, the passenger receives the client application. With the aim of being activated, the application first has to be configured with the desired type of news to be received, like sports or weather news, and the manner of rendering those news like text, video, or audio.

The *Context Delivery* feature is a mandatory feature of every member of the

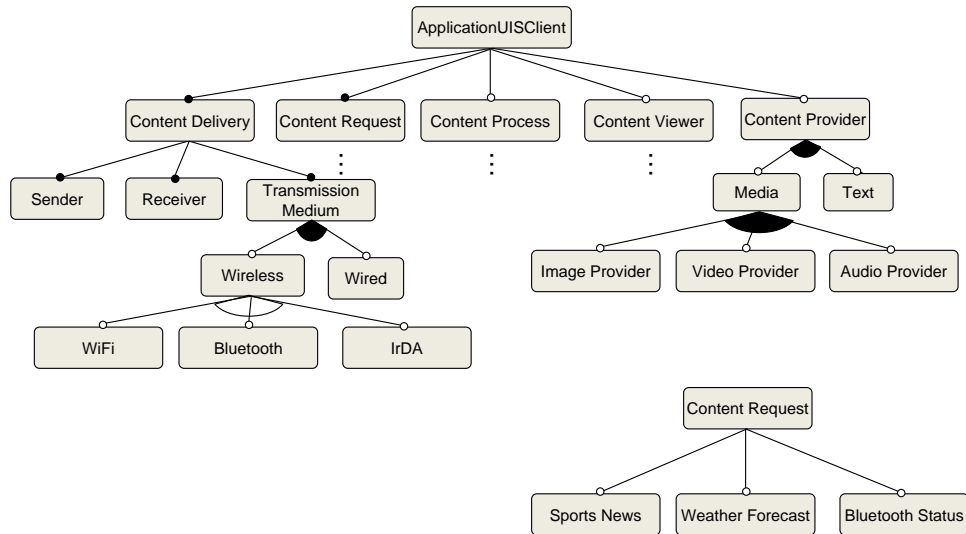


Fig. 2. The Airport Application Feature Model

family. It defines the sender, i.e. server sending news reports, the receiver, device receiving news reports and the transmission medium its mandatory features *Sender*, *Receiver*, and *Transmission Medium*, respectively. The transmission medium can be wired or wireless. In the feature diagram, this is presented as the *or* group of *Transmission Medium* child features consisting of the *Wireless* and *Wired* features. Moreover, wireless transmission can be either over WLAN, Bluetooth, or Infrared, presented as an alternative group of features consisting of *WiFi*, *Bluetooth*, and *IrDA* features, respectively.

The *Content Request* feature encapsulates all kinds of reports that a passenger would like to receive. Therefore, a user can specify receiving sports news report (*Sports News*), weather forecast news report (*Weather Forecast*) or even the status of the communication medium, such as, e.g., Bluetooth (*Bluetooth Status*), in order to detect when it is not working properly.

The news report can be presented to the passenger in different manners, encapsulated in the *Content Provider* feature. The presentation can be in the textual form (*Text*) or in some media (*Media*) form such as image (*Image Provider*), video (*Video Provider*), or audio (*Audio Provider*).

The relationships between features which can not be captured with parent-child feature relations in feature diagrams are specified as integrity constraints. Two integrity constraints are widely recognized:

- (1) **Includes**—the selection of a feature automatically includes the selection of some other feature. For example, in the airport client application, the feature model contains specification of includes integrity constraints between the *Blue-*

6 M. Bošković, E. Bagheri, D. Gašević, B. Mohabbati, N. Kaviani, M. Hatala

tooth Status and *Bluetooth* feature. This implies that the selection of *Bluetooth Status* feature of the Figure 2 feature diagram, automatically includes the selection of *Bluetooth* feature of the communication medium.

- (2) **Excludes**—the selection of a feature automatically eliminates some other feature. In our example feature model, because the wireless communication can be provided only over one wireless medium, there exist two excludes integrity constraints: between the *WiFi* feature and the *Bluetooth Status*, and between the *IrDA* feature and the *Bluetooth Status*. According to these integrity constraints, a selection of the *WiFi* automatically eliminates the *Bluetooth Status* feature from the final configuration, because querying the bluetooth status when the communication is through other wireless device is not possible.

As previously mentioned, features in feature models represent distinguishable characteristics of the software systems, which are of interest to the stakeholders'. However, for features to be able to be implemented might require the existence of some execution platform characteristics and/or external services may be required. For example, in the example case study, the *Bluetooth* transmission medium feature requires a deployment platform containing a Bluetooth communication device. The presentation of the news reports over some media, e.g. video (*Video Provider*), also requires a deployment platform with a device for presentation of that media, e.g. a screen. Or, if a feature model is extended with a feature specifying possibility of password authentication, this feature will require an available encryption service for communication with the news report provider, because sending a password for authentication without encryption enables identity theft. The device characteristics and required services are not presented in the feature diagram, because they are not features of interest to the stakeholders'. They can be specified in the feature model with the additional documentation provided as a part of the feature model. However, these hard requirements have a great impact on the application configuration, because the final configuration can have only the features with satisfied hard requirements. For this reason, there is a need for a way of specifying the hard requirements in the feature model, and using this information for specialization, without influencing the original feature diagram. The next section depicts such an approach.

3. The Seamless Hard Requirements Specification and Staged Configuration

To address the problem of seamless integration of hard requirements, we exploit the capability of OWL to integrate information specified in different ontologies. We propose the approach depicted in Figure 3.

The *Domain Scoping and Family Requirement Analysis* phase of the domain analysis aims at the scoping and analyzing the requirements of the software family members [11]. It identifies functionality which members of the family should provide, analyzes the commonality and variability of future family mem-

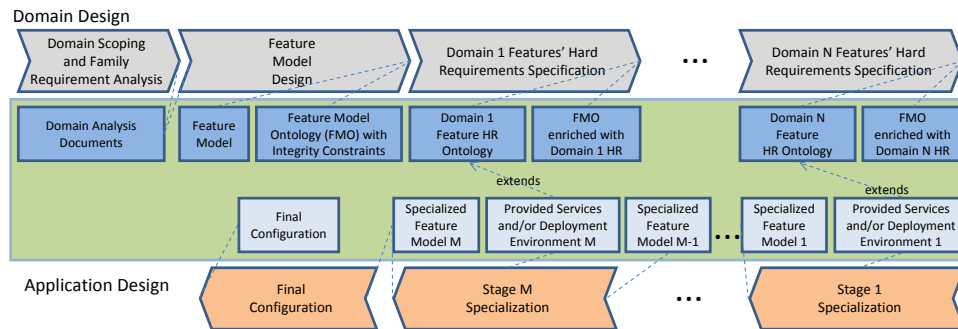


Fig. 3. Integration of features' hard requirements in feature models and staged specialization

bers. Furthermore, each feature is precisely and unambiguously defined and their relations and integrity constraints are defined. Furthermore, feature definitions also encompass specification of their hard requirements. Hard requirements are being scoped and grouped in the domains like devices domain, security domain etc, in order to serve as foundations for later development of hard constraints ontologies. All this information is documented in *Domain Analysis Documents*. The *Domain Analysis Documents* serve as a road map for producing feature models and feature hard requirements ontologies (HRO).

Domain Scoping and Family Requirement Analysis is followed by *Feature Model Design* based on *Domain Analysis Documents*. Feature model design includes the development of product family feature model. Some methods for performing *Domain Scoping and Family Requirement Analysis* and *Feature Model Design* are: Feature Oriented Domain Analysis [8], Organization Domain Modeling (ODM) [12], FeatuRSEB [13], Feature-Oriented Reuse Method (FORM) [14].

When the feature model is developed, in order to use DL reasoners for verification of feature configuration and the extensibility of ontologies expressed in OWL, the feature model is transformed into an OWL *Feature Model Ontology* (FMO).

The final step of the domain design stage is the integration of hard requirements into the FMO. This is carried out in two steps. The first step is the reuse and development of the HR ontology of some particular domain. Previously mentioned hard requirements domain groups documented in *Domain Analysis Documents* serve as a foundation for this activity. After the ontology of hard requirements is developed, the classes representing features in FMO are enriched with additional properties representing the hard requirements of that feature. More on the enrichment of feature representation in the FMO is given in Section 4.

The application design phase consists of consecutive steps of feature model specialization concluded with the final configuration of the product. The application designer must understand the requirements and services and/or the deployment en-

vironment that the stakeholder can provide. The provided services and/or deployment environment must be structured in such way that they form specialization stages. In each specialization stage, the application developer specifies the description of the stakeholder's provided services and/or execution platform characteristics of interest to that stage. These provided services and/or execution platform characteristics are integrated in the HROs. These extended HRO ontologies are used in algorithm for for automated feature model specialization. The specialization algorithm, described in Section 5 takes as the input (specialized) feature model and the extended ontology of features' hard requirements, and yields the new specialized feature model. The specialized feature model is formed by removing unfeasible features, from the input feature model. Unfeasible features are features whose hard requirements are not satisfied with the provided stakeholder's services and/or deployment environment characteristics. The whole process ends with the configuration of the final product from the most specialized feature model.

Finally, it should be noticed that specialization stages in staged configuration can but do not have to correspond to the domains specified in the product line design. One domain can be used in several stages, and one specialization stage can use several ontologies. However, in the case that a specialization stage uses several ontologies, the specialization stage must be separated to several sub stages where each sub stage corresponds to the employed ontology.

4. Feature and Hard Constraints Modeling in OWL

Wang et al. [2] base the verification of feature configurations on TBox reasoning. The TBox reasoning is enabled by representing both, feature model and feature configuration as ontologies with the same concepts. The verification of configuration is then provided by checking the consistency of ontologies representing feature model and feature representation. Such approach additionally enables usage of DL reasoners for debugging the feature configuration, because the DL reasoners show inconsistencies which should be removed in order configuration to be consistent. This is particularly important in the final phase of the staged configuration when the application is finally configured. However, during the specialization steps, the configuration checking is not used, due to the fact that specialization stages only remove the unfeasible features.

In order to complement this configuration verification with staged configuration, in this section we demonstrate how we integrate the features' hard requirements into the FMO. Table 1 shows part of the description logic syntax used in this paper for the description of OWL ontologies.

4.1. Feature Modeling in OWL

Wang et al. model each node in the feature model with two classes in the FMO [2]: a "feature" class and a "rule" class. "Feature" classes are mutually disjoint and they represent a feature of the feature model. The "rule" class is used for the specification

Table 1. Summary of used DL(OWL) syntax

Notation	Semantics
\top	superclass of all OWL classes
\perp	an empty set
$C_1 \sqsubseteq C_2$	C_1 is a subset of C_2
$R_1 \sqsubseteq R_2$	R_1 is a sub property of R_2
$C_1 \sqsubseteq \neg C_2$	Classes C_1 and C_2 are disjoint
$C_1 \sqcap C_2$	Class intersection
$C_1 \sqcup C_2$	Class union
$C_1 \equiv C_2$	Class equivalence
$\forall P.C$	An anonymous class whose every instance has instances of property P and all values of property P of instances are elements of class C
$\exists P.C$	An anonymous class whose every instance has instances of property P and at least one value of property P of every instance is in C
$\geq nP$	An anonymous class whose every instance has at least n instances of property P
$\geq 1P \sqsubseteq C$	The domain of property P is class C
$\top \sqsubseteq \forall P.C$	The range of property P is class C

of a feature relation with other features. “Rule” classes are related to corresponding “feature” classes with the existentially restricted property. Let us assume that $\mathbf{Feat}_1 \dots \mathbf{Feat}_n$ are features of the feature model. They are represented in the FMO as:

$$\begin{aligned}
 F_{Feat_i} &\sqsubseteq \top \\
 FRule_{Feat_i} &\sqsubseteq \top \\
 hasF_{Feat_i} &\sqsubseteq ObjectProperty \\
 \top &\sqsubseteq \exists hasF_{Feat_i}.F_{Feat_i} \\
 FRule_{Feat_i} &\equiv \exists hasF_{Feat_i}.F_{Feat_i}, \text{ for } 1 \leq i \leq n \\
 F_{Feat_j} &\sqsubseteq \neg F_{Feat_k}, \text{ for } 1 \leq j, k \leq n \wedge i \neq j
 \end{aligned}$$

For example, from the feature model of the airport application, the *Bluetooth* feature is presented in the FMO ontology as:

$$\begin{aligned}
 F_{Bluetooth} &\sqsubseteq \top \\
 FRule_{Bluetooth} &\sqsubseteq \top \\
 hasF_{Bluetooth} &\sqsubseteq ObjectProperty \\
 \top &\sqsubseteq \exists hasF_{Bluetooth}.F_{Bluetooth} \\
 FRule_{Bluetooth} &\equiv \exists hasF_{Bluetooth}.F_{Bluetooth}
 \end{aligned}$$

10 *M. Bošković, E. Bagheri, D. Gašević, B. Mohabbati, N. Kaviani, M. Hatala*

Mandatory child features are features which are included in the configuration if their parent features are included. Mandatory child features $Feat_{cm_1}, \dots, Feat_{cm_m}$ of the parent feature $Feat_{pm}$ are represented in the FMO as:

$$\begin{aligned} FRule_{Feat_{pm}} &\sqsubseteq \exists has F_{Feat_{cm_1}} . F_{Feat_{cm_1}} \\ \dots \\ FRule_{Feat_{pm}} &\sqsubseteq \exists has F_{Feat_{cm_m}} . F_{Feat_{cm_m}} \end{aligned}$$

Such features are the *Sender*, *Receiver*, and *Transmission Medium* child features of the *Content Delivery* feature in Figure 2. From these features are generated the following entries in the corresponding FMO:

$$\begin{aligned} FRule_{ContentDelivery} &\sqsubseteq \exists has F_{Sender} . F_{Sender} \\ FRule_{ContentDelivery} &\sqsubseteq \exists has F_{Receiver} . F_{Receiver} \\ FRule_{ContentDelivery} &\sqsubseteq \exists has F_{TransmissionMedium} . F_{TransmissionMedium} \end{aligned}$$

Optional child features may or may not be included in the final configuration and for that reason their parent feature’s ontology “rule” class does not contain any restriction with respect to that feature. Such feature is the *Sports News*, *Weather Forecast*, and *Bluetooth Status* of the feature of the *Content Request* feature.

Alternative features group is a group of features from which only one feature can be included in the final configuration. For that reason, from the alternative feature group $Feat_{ca_1}, \dots, Feat_{ca_a}$ of the parent feature $Feat_{pa}$ is generated in FMO:

$$\begin{aligned} FRule_{Feat_{pa}} &\sqsubseteq \bigsqcup_{i=1}^a \exists has F_{Feat_{ca_i}} . F_{Feat_{ca_i}} \\ FRule &\sqsubseteq \neg \bigsqcup_{i=1}^a \bigsqcup_{j=i+1}^a (has F_{Feat_{ca_i}} . F_{Feat_{ca_i}} \sqcap has F_{Feat_{ca_j}} . F_{Feat_{ca_j}}) \end{aligned}$$

The first of the above two DL axioms ensures that some feature will be selected. The lower one is the negation of conjunction of any two features, used to restrict the selection to only one feature. In the Figure 2, the alternative feature group are child features *WiFi*, *Bluetooth*, and *IrDA* of the parent feature *Wireless*. The *WiFi*, and *IrDA* features are represented in the same way as the already described *Bluetooth* feature. The *Wireless* feature is then represented in the Figure 2 FMO as:

$$\begin{aligned} F_{Wireless} &\sqsubseteq \top \\ FRule_{Wireless} &\sqsubseteq \top \\ has F_{Wireless} &\sqsubseteq ObjectProperty \\ \top &\sqsubseteq \exists has F_{Wireless} . F_{Wireless} \\ FRule_{Wireless} &\equiv \exists has F_{Wireless} . F_{Wireless} \\ FRule_{Wireless} &\sqsubseteq \end{aligned}$$

$$\begin{aligned}
 FRule_{Wireless} \sqsubseteq & ((\exists hasF_{WiFi}.F_{WiFi}) \sqcup (\exists hasF_{Bluetooth}.F_{Bluetooth}) \sqcup (\exists hasF_{IrDA}.F_{IrDA})) \\
 & \sqsubseteq \neg(((\exists hasF_{WiFi}.F_{WiFi}) \sqcap (\exists hasF_{Bluetooth}.F_{Bluetooth})) \\
 & \quad ((\exists hasF_{WiFi}.F_{WiFi}) \sqcap (\exists hasF_{IrDA}.F_{IrDA})) \\
 & \quad ((\exists hasF_{Bluetooth}.F_{Bluetooth}) \sqcap (\exists hasF_{IrDA}.F_{IrDA})))
 \end{aligned}$$

Or feature group, as defined in Section 2, mandates selection of at least one feature from that group, when the parent feature is chosen. To facilitate verification of such selection, a *or feature group* of the parent feature $Feat_{op}$, consisting of $Feat_{co_1}, \dots, Feat_{co_o}$ is in FMO represented as:

$$FRule_{Feat_{op}} \sqsubseteq \bigsqcup_{i=1}^o (\exists hasF_{Feat_{co_i}}.F_{Feat_{co_i}})$$

In the Figure 2, such features are *Wireless* and *Wired* features of the *Transmission Medium* parent feature, and they are represented in the corresponding FMO ontology as:

$$FRule_{TransmissionMedium} \sqsubseteq (\exists hasF_{Wireless}.F_{Wireless}) \sqcup (\exists hasF_{Wired}.F_{Wired})$$

Beside feature diagram parent child relations, FMO also facilitates integrity constraints verification. The **includes integrity constraint** is very similar to mandatory child features of the parent feature, and are, for this reason, presented in the same way. For $Feat_{ic_{f_1}}, \dots, Feat_{ic_{f_i}}$ features which are included in the feature $Feat_{if}$, the corresponding FMO contains the following axioms:

$$\begin{aligned}
 FRule_{Feat_{if}} \sqsubseteq & \exists hasF_{Feat_{ic_{f_1}}}.F_{Feat_{ic_{f_1}}} \\
 \dots & \\
 FRule_{Feat_{if}} \sqsubseteq & \exists hasF_{Feat_{ic_{f_i}}}.F_{Feat_{ic_{f_i}}}
 \end{aligned}$$

An includes integrity constraint of the Figure 2, as previously mentioned, exists between the *Bluetooth Status* feature and the *Bluetooth* feature. This integrity constraint is represented in the FMO as:

$$FRule_{BluetoothStatus} \sqsubseteq \exists hasF_{Bluetooth}.F_{Bluetooth}$$

Finally, the **excludes integrity constraint**, is used for restriction selection of some features in the same configuration. For a given feature $Feat_{ef}$, and the set of features $Feat_{ef_{c_1}}, \dots, Feat_{ef_{c_e}}$, the corresponding FMO contains the following entry:

12 *M. Bošković, E. Bagheri, D. Gašević, B. Mohabbati, N. Kaviani, M. Hatala*

$$\begin{aligned} FRule_{Feat_{ef}} &\sqsubseteq \neg(\exists has F_{Feat_{efc_1}}. F_{Feat_{efc_1}}) \\ \dots \\ FRule_{Feat_{ef}} &\sqsubseteq \neg(\exists has F_{Feat_{efc_e}}. F_{Feat_{efc_e}}) \end{aligned}$$

In our example it is stated in Section 2, that excludes integrity constraints exist between the *Bluetooth Status* feature and *WiFi* and *IrDA* features. This is represented in the FMO as:

$$\begin{aligned} FRule_{BluetoothStatus} &\sqsubseteq \neg(\exists has F_{WiFi}. F_{WiFi}) \\ FRule_{BluetoothStatus} &\sqsubseteq \neg(\exists has F_{IrDA}. F_{IrDA}) \end{aligned}$$

4.2. Feature Configuration Specification in OWL

To provide inconsistencies debugging, Wang et al. base their approach for product configuration verification on TBox reasoning, i.e., they specify the feature configuration as classes and use TBox reasoning on ontology inconsistencies to point out the conflicting features in configuration.

Let us assume that there exist a feature model consisting of $Feat_1, \dots, Feat_n$ features. And let $Feat_1$ be a root feature of the feature model. Let the $Feat_2, \dots, Feat_s$ be a set of selected features, and $Feat_{s+1}, \dots, Feat_n$ be a set of the features not included in the final configuration, then the feature configuration is presented in OWL as a class C .

$$\begin{aligned} C &\sqsubseteq FRule_{Feat_1} \\ C &\equiv \prod_{i=2}^s (has F_{Feat_i}. F_{Feat_i}) \prod_{j=s+1}^n (\neg has F_{Feat_j}. F_{Feat_j}) \end{aligned}$$

To illustrate the approach, let us assume that among other, features *Content Delivery*, *Sender*, *Receiver*, *Transmission Medium*, and *Wired* of the Figure 2 are selected, while *Wireless*, *WiFi*, *Bluetooth*, and *IrDA* are not selected. The rest of the feature model is not of importance for this illustration. Such configuration is presented in FMO as follows.

$$\begin{aligned} C &\sqsubseteq FRule_{ApplicationUISClient} \\ C &\equiv (\exists has F_{ContentDelivery}. F_{ContentDelivery}) \sqcap (\exists has F_{Sender}. F_{Sender}) \\ &\sqcap (\exists has F_{Receiver}. F_{Receiver}) \sqcap (\exists has F_{TransmissionMedium}. F_{TransmissionMedium}) \\ &\sqcap (\exists has F_{Wired}. F_{Wired}) \sqcap \dots \\ &\sqcap (\neg \exists has F_{Wireless}. F_{Wireless}) \sqcap (\neg \exists has F_{WiFi}. F_{WiFi}) \\ &\sqcap (\neg \exists has F_{Bluetooth}. F_{Bluetooth}) \sqcap (\neg \exists has F_{IrDA}. F_{IrDA}) \sqcap \dots \end{aligned}$$

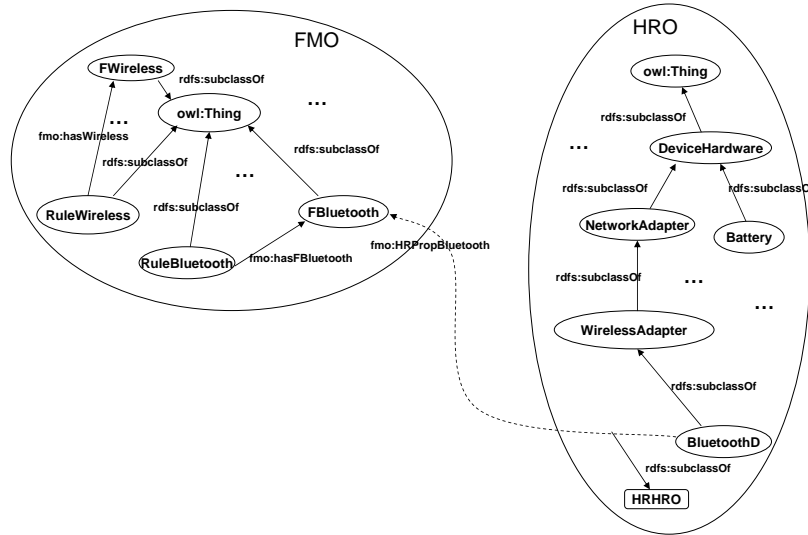


Fig. 4. Feature Model Ontology and a Hard Requirements Ontology

4.3. Features' Hard Requirements Specification

The seamless integration of features' hard requirements is based on addition of properties describing their hard requirements to "feature" classes of FMO. The extension is illustrated in Figure 4.

At the left hand side of Figure 4 depicts the part of FMO describing the **Wireless** and **Bluetooth** features of the Figure 2. At the right hand side is a part of the extended DCO ontology for modeling the hardware devices in the domain of ubiquitous computing, presented in [9]. The part of the ontology presented in Figure 4, specifies that the **BluetoothD** (bluetooth device) is a type of **DeviceHardware** (hardware device), more particularly it is a **WirelessAdapter** kind of **NetworkAdapter**.

The ontology for modeling hard requirements in Figure 4 is presented in DL as:

$$\begin{aligned}
 Device &\sqsubseteq \top \\
 DeviceHardware &\sqsubseteq \top \\
 \geq 1hasDeviceHardware &\sqsubseteq Device \\
 \top &\sqsubseteq \forall hasDeviceHardware.Devicehardware \\
 NetworkAdapter &\sqsubseteq DeviceHardware \\
 \top &\sqsubseteq \forall hasNetworkAdapter.NetworkAdapter
 \end{aligned}$$

14 *M. Bošković, E. Bagheri, D. Gašević, B. Mohabbati, N. Kaviani, M. Hatala*

$WirelessAdapter \sqsubseteq NetworkAdapter$
 $WiFi \sqsubseteq WirelessAdapter$
 $\geq 1hasWiFi \sqsubseteq Device$
 $\geq 1hasBluetooth \sqsubseteq Device$
 $\top \sqsubseteq \forall hasWiFi.WiFi$
 $\top \sqsubseteq \forall hasBluetooth.Bluetooth$
...

The hard requirements of features are integrated in the feature models as properties of features. In order to be able to make a distinction between the hard requirements of different ontologies, each hard requirements ontology HRO contains a property which is a super property of all hard requirements defining properties. Therefore, the ontology HRO contains a property:

$ObjectProperty \sqsubseteq HR_{HRO}$

Properties defining hard requirements of features, are subtypes of the HR_{HRO} property. Each relation property has as a range the class of the HRO ontology modeling some particular service or deployment environment characteristic which can be required. The domain of this property is the union of all “feature” classes of the FMO having this service or deployment environment characteristic as hard constraint. Therefore, let us assume that $Feat_{hc_1}, \dots, Feat_{hc_m}$ have a hard requirement defined with the class HRC of the HRO . Then, this hard requirement is defined as follows.

$HRprop_{HRC} \sqsubseteq HR_{HRO}$
 $\top \sqsubseteq \forall HRprop_{HRC}.HRC$
 $\leq 1HRprop_{HRC} \sqsubseteq \bigsqcup_{i=1}^m F_{Feat_{hc_i}}$

In our example, we want to state that in order to be possible to choose the **Bluetooth** communication, the deployment environment has to have the deployment environment with the bluetooth communication device. This is specified in the FMO as the $HRprop_{BluetoothD}$ having the $BluetoothD$ from the HCO as the range and the $Bluetooth$ “feature” class as the domain. For example, the $Bluetooth$ feature’s requirement for a device with a Bluetooth wireless device is specified as.

$HRprop_{BluetoothD} \sqsubseteq HR_{HRO}$
 $\top \sqsubseteq \forall HRprop_{BluetoothD}.BluetoothD$
 $\geq 1HRProp_{HRC} \sqsubseteq Bluetooth$

5. Automated Specialization

In order to automate the process of specialization and provide users means to have verified feature configuration, we accompany our approach with an algorithm for automatic specialization according to provided services and/or deployment environment characteristics. The entry of the automatic configuration is the (specialized) feature model, and the set of provided services and/or deployment environment characteristics specified as instances of HRO ontology classes. The outcome of this procedure is the subset of the entry feature model which contain features unfeasible due to the dissatisfaction of their hard requirements. The algorithm for the automatic specialization is defined in the following.

Definition 1. (Set of unfeasible features $Feat_{uf}$) We let $UFeat_{Feat_{uf}}$ for the feature $Feat_{uf}$ be the set of unfeasible features when hard requirements of the feature $Feat$ are not satisfied, which is inductively defined as follows:

- (1) $Feat_{uf} \in UFeat_{Feat_{uf}}$,
- (2) Let $Feat_{ic}$ be a feature in the feature model and $Feat_{iic_1}, \dots, Feat_{iic_{icf}}$ features having includes integrity constraint with $Feat_{ic}$, and let $Feat_{ic} \in UFeat_{Feat_{uf}}$. Then are also $Feat_{iic_1}, \dots, Feat_{iic_{icf}} \in UFeat_{Feat_{uf}}$,
- (3) Let feature $Feat_{p_{pf}}$ be a parent feature of the feature $Feat_{cf}$, and let $Feat_{p_{pf}} \in UFeat_{Feat_{uf}}$. Then then also $Feat_{cf} \in UFeat_{Feat_{uf}}$,
- (4) Let feature $Feat_m$ be a mandatory feature of the feature $Feat_p$. And let $Feat_m \in UFeat_{Feat_{uf}}$. Then also $Feat_p \in UFeat_{Feat_{uf}}$.
- (5) Let $Feat_{or_1}, \dots, Feat_{or_n}$ form a whole **or** or **alternative** feature group of the parent feature $Feat_p$. And let $Feat_{or_1}, \dots, Feat_{or_n} \in UFeat_{Feat_{uf}}$. Then also $Feat_p \in UFeat_{Feat_{uf}}$.

Let us consider this definition in the context of the feature diagram presented in Figure 2. Let us assume that the provided platform does not have any of the wireless communication devices (Bluetooth, WiFi, IrDA). And let us assume that in previous specialization stages features $WiFi$ and $IrDA$ have been removed from the feature diagram. Then $UFeat_{Bluetooth} = \{Bluetooth, BluetoothStatus, Wireless\}$. According to the item (1) of the definition the $Bluetooth$ feature is included in the set of unfeasible features caused by unfeasibility of it. Item (2) of the definition puts the feature $BluetoothStatus$ into this set. Because of the removal of $WiFi$ and $IrDA$ features in some of the previous specialization stages, the only left feature of the or group is the $Bluetooth$ feature. Having this feature in the $UFeat_{Bluetooth}$ set, according to the item (5) implies that the feature $Wireless$ is also an element of the $UFeat_{Bluetooth}$ set. Now let us additionally assume that the provided platform also does not have the wired communication device, and that in some previous specialization stages the $Wired$ feature has been removed. This would imply that (item (5)), the feature $TransmissionMedium$ is also an element of the $UFeat_{Bluetooth}$. Furthermore, item (4) makes the $ContentDelivery$ feature an element of this set. It means that in the case that there is no communication medium, the content

delivery would not be possible. Having the the *Content Delivery* feature, according to rule (3) of the definition, makes features *Sender* and *Receiver* elements of the $UFeat_{Bluetooth}$ set, which is reasonable, because if there is no content delivery, than having a sender and a receiver does not have sense. Finally, according to rule (4) again, the ***ApplicationUISClient***, the root feature is also member of the $UFeat_{Bluetooth}$ which makes the whole application unfeasible. This is commonsensical, because if there is no content delivery than there is no need for the application at all.

In order to integrate the provided services and hardware characteristics into the knowledge base used later for reasoning, we extend the previous HR ontology with the class which models provided services and/or device capabilities. Extension is the addition of instances specifying the provided services and hardware characteristics. Furthermore, in order to unite all platform characteristics and available services as one concept that is used in TBox reasoning, we also add a new class to the ontology which is a union of classes defining the provided services and hardware characteristics. Let cs_1, \dots, cs_{ps} be a set of provided services and/or device capabilities. This is, we add to the HR ontology the following class:

$$PSEC \equiv \bigsqcup_{i=1}^{ps} C(cs_i)$$

Finally, the knowledge base of our system KB is the union of the feature model ontology and the hard requirements ontology of that specialization (sub) stage:

$$KB = FMO \cup HRO$$

The algorithm for **Automated Specialization (ASpec)** is specified as follows. The input to this algorithm is $ISFM$, a (specialized) feature which can be either the complete feature model when it is in the first specialization, or the outcome of previous specialization stage. SFM is the specialized feature model which will be the outcome of this algorithm. Initially, SFM takes the value of $ISFM$ (Δ) and with this algorithm, all its unfeasible features are removed from it.

The specialization algorithm is performed for each feature of the $ISFM$ (\clubsuit). In the case that it is still not shown that the feature is unfeasible, i.e., it is still the element of SFM , the algorithm checks if it has the hard requirements of the HRO at all (\diamond). It is performed by checking whether this feature is the subset of the domain of the HR_{HRO} property. Not being in the domain of the property which is the union of all properties modeling hard constraints, implies that it does not have them. If it does, then it is checked if all hard requirements are satisfied (\spadesuit). This check is performed by asking if the range of the union of all relations presenting hard constraints of that feature is a subset of the class representing all provided services and/or deployment environment capabilities. In the case it is not, then the

feature is unfeasible and the unfeasible feature set of that feature is removed from the feature model (∞).

Algorithm *ASpec*(*ISFM*)

input *ISFM*

output Feature model specialization *SFM*

begin algorithm

/* \triangle *SFM* is initialized */

$SFM \leftarrow ISFM$

/* \clubsuit All features in *ISFM* are analyzed one by one */

for each ($Feat \in ISFM$)

/* \diamond Hard requirements of *HRO* are checked */

if $KB \models F_{Feat} \sqsubseteq \geq 1HR_{HRO}$ **then**

/* \spadesuit Check to see whether all hard requirements are satisfied */

if ($KB \models \geq 1HR_{Feat} \sqsubseteq F_{Feat}$) and ($KB \not\models \top \sqsubseteq \forall(HR_{Feat} \sqcap HR_{HRO}).PSEC$) **then**

/* ∞ unfeasible features are removed */

$SFM \leftarrow SFM/UFeat_{Feat}$;

end if

end if

end for each

return *SFM*;

end algorithm

The outcome of this algorithm is a feature model which can be used in the next specialization step. Such specialized feature model should be sound and complete as defined in the following.

Definition 2. (Soundness)

The specialized feature model is sound iff it is consistent with the integrity constraints and it maintains the hierarchical structure specified in the complete feature model of the domain.

Definition 3. (Completeness)

The specialized feature model is complete if all configurations which can be derived from it satisfy hard requirements.

The specialized feature model created with the *ASpec* algorithm is sound and complete, as we show in the following.

Theorem 1. The feature model specializations developed by the *ASpec* is sound.

Proof. In order for a specialized feature model not to be sound, it should:

- (1) contain a child feature without a parent feature,
- (2) contain a feature with includes integrity constraint with a removed feature.

18 M. Bošković, E. Bagheri, D. Gašević, B. Mohabbati, N. Kaviani, M. Hatala

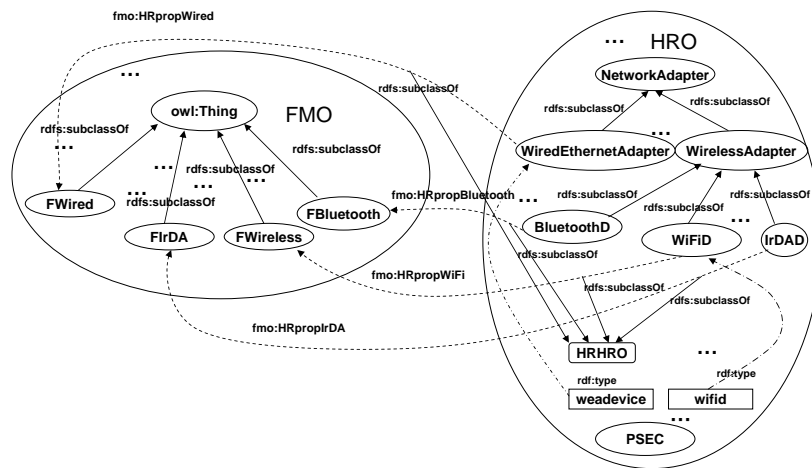


Fig. 5. An example of the provided services and capability characteristics configuration

Items 2 and 3 of Definition 1 prohibits such cases. \square

Theorem 2. A feature model configuration developed by *ASpec* is complete.

Proof. Let us assume that we have an incomplete specialized feature model. An incomplete specialized feature model is the one which has an unfeasible feature. Through the analysis of feature relationships and integrity constraints it can be concluded that a feature is unfeasible if:

- (1) the hard requirements of the feature are not provided,
- (2) it has an includes integrity constraint with an unfeasible feature,
- (3) its parent feature is unfeasible,
- (4) at least one of its mandatory child features is unfeasible,
- (5) all of its child features belonging to one or group are unfeasible.

Each of these cases have the corresponding item with the same number in the definition of $UFeat_{Feat}$ which restricts their existence in the final configuration. \square

6. An Illustrative Example

For the purpose of illustration, we use the feature model depicted in Figure 2 feature model as an input for our proposed algorithm. Now let us assume that to the “feature” classes of transport mediums are additional properties for the definition of hard requirements as illustrated in Figure 5.

Specification of all transmission medium hard requirements is performed similarly to the the case of the *Bluetooth* feature at the end of Section 4. Therefore, the rest of the features’ hard requirements for the example feature model is specified as

follow:

$$\begin{aligned} HRprop_{WiFi} &\sqsubseteq HR_{HRO} \\ \top &\sqsubseteq \forall HRprop_{WiFi}.WiFiDD \\ &\geq 1HRProp_{HRC} \sqsubseteq WiFi \end{aligned}$$

$$\begin{aligned} HRprop_{IrDA} &\sqsubseteq HR_{HRO} \\ \top &\sqsubseteq \forall HRprop_{IrDA}.IrDAD \\ &\geq 1HRProp_{HRC} \sqsubseteq IrDA \end{aligned}$$

$$\begin{aligned} HRprop_{Wired} &\sqsubseteq HR_{HRO} \\ \top &\sqsubseteq \forall HRprop_{Wired}.WiredEthernetAdapter \\ &\geq 1HRProp_{HRC} \sqsubseteq Wired \end{aligned}$$

Let us assume that the provided execution platform has capabilities of communication over **WiFi** and over **Ethernet**. This information is specified in the hard requirements ontology by adding the *weadevice* instance of the *WiredEthernetAdapter* class, and the *wifidevice* instance of the *WiFiD* class. Furthermore, the capability of the provided execution platform is defined through the *PSEC* class, which is in this case defined as a union of *WiredEthernetAdapter* and *WiFiD* classes.

$$PSEC \equiv WiredEthernetAdapter \sqcup WiFiD$$

The execution of the **ASpec** algorithm starts with receiving input feature model. For the purpose of this example, we will assume that *ISFM* is supposed to be configured for breath first traversal.

Features *ApplicationUISClient*, *Content Delivery*, *Content Request*, *Content Viewer*, *Content Provider*, *Sender*, *Receiver*, *Transmission Medium*, *Sports News*, *Weather Forecast*, *Bluetooth Status*, *Media* and *Text*, when checked for the hard requirements satisfaction in () all stay in the *SFM* set because they do not have the hard requirements and do not pass the () condition. When the value of the *Feat* variable in the () loop is the *Wired* feature, the control flow passes the () condition, because the *Wired* feature contains the specification of the hard requirements. After satisfying the () condition, the control flow enters into checking whether all hard requirements are satisfied. The concept HR_{Feat} in this condition check is a set of all hard requirements of the *Feat* feature. If this class is not a subclass or equal to the *PSEC* class, then all hard requirements are not satisfied and this feature is unfeasible. In the case of the *Wired* feature, $HR_{Wired} \equiv WiredEthernetAdapter$ and it is a subclass of the *PSEC* class. Therefore, all hard requirements are satisfied. In the further execution of the **ASpec** algorithm for this example, features *Image Provider*, *Audio Provider*, and *Video Provider* do not have hard requirements and

20 M. Bošković, E. Bagheri, D. Gašević, B. Mohabbati, N. Kaviani, M. Hatala

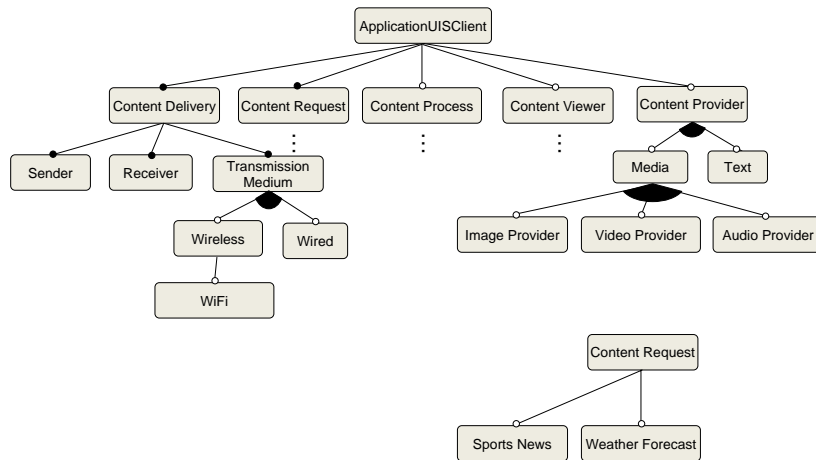


Fig. 6. Specialized feature model after the execution of the ASpec algorithm for the environment characteristics presented in Figure 5

similarly to features traversed before the *Wired* feature stay in the *SFM* set. Feature *WiFi* similarly to the *Wired* feature has all hard requirements satisfied and stays in the *SFM* set. However, this is not the case with the *Bluetooth* feature. The $HR_{Bluetooth} \equiv BluetoothD$ and it is not a subclass of *PSEC*. For this reason, the *Bluetooth* feature is not feasible. Additionally, unfeasibility of the *Bluetooth* feature implies that the *Bluetooth Status* is also unfeasible, because of the includes integrity constraint between the *Bluetooth Status* and *Bluetooth* features. For this reason according to Definition 1, the *Bluetooth Status* feature is also removed from the *SFM* set. Finally, the hard requirements of the *IrDA* feature are also not satisfied because the execution environment does not contain an infrared communication device. The final specialized feature model is presented in Figure 6.

7. Related Work and Discussion

Real industrial feature models with very large number of features motivates research in the direction of to splitting feature configuration in stages and verification of specialized feature models and feature configurations.

Facing the problem of complex feature verification, several approaches have been proposed. Mannion [15] proposed the usage of first order logic for representation of feature models and verification of feature configurations. However, representation of feature models as propositional formulas was not convenient for use by practitioners. Practitioners were more fond of graphical representation of feature models as feature diagrams. Czarnecky et al. [7] made use of context free grammars for representing the semantics of feature models. However, relying only on context free grammars for feature models does not facilitate specification of integrity constraints.

For representation of integrity constraints with context free grammars, Czarnecky and Kim [17] employ Object Constraint Language (OCL). To bridge the gap between the use of context free grammars and propositional calculus. Batory [16] bridges the gap between specification of feature models as grammars and propositional formulas. Herewith, he exploits the full potential of both representation of feature models with grammars.

Approaches based on propositional formulas (and grammars) can use Logic Truth Maintenance Systems (LTMS) for verification and debugging, similar to Description Logic reasoners in the approach presented in feature model [16] [18] [19]. The most widely used methods in LTMS are Constraint Satisfaction Problem (CSP) solvers [20], Binary Decision Diagrams (BDD) [22], and SATisfiability problem (SAT) [16]. CSP solvers are used in such a way that verify the feature model by examining whether the configuration falls into the set where all constraints of the feature model are satisfied. However, the the shortcoming of this approach is not suitable for feature models with a larger amount of features (>25 [20]). BDDs are data structures representing a boolean function, used in this case for representing the possible configuration space. Contrary to CSP solvers BDDs are time efficient, but suffer a very inefficient space complexity. SAT solvers are a very promising approach for verification, since they become more and more efficient [21]. However, for industrial feature models (>1000), this approach is still not applicable [21]

In order to reduce the complexity of a product configuration from the large feature models, the product configuration needs to be separated into consecutive steps of sub feature model configurations. Work of Czarnecky et al. in staged [7] and multi-level [23] configuration served as the initial steps in this direction. In order to define these processes in a precise way, and to specify specialization steps in the way that they produce sound and complete specialized feature models, Classen et al. [24] introduce a formal semantic for the multi-level staged configuration. One more recently introduced approach for formal specification of multi-level staged configuration is MUSCLE (MUlti-step Software Configuration proBLEm solver) has been introduced by White et al. [25].

Approaches based on propositional formulas do not facilitate seamless integration of features' hard requirements into the feature model. In approaches based only on propositional formulas, services and/or deployment platform requirements are represented as features in feature model. Features' hard requirements are then defined as includes integrity constraints between the feature and services and/or integrity constraints that it requires. The information about required services and platform characteristics might significantly increase already large feature model representation, increasing the complexity, with the information not of interest to the stakeholders. Approaches based on grammars and propositional formulas overcome this shortcoming by decoupling representation, based on grammars, from the verification mechanism based on propositional formulas. In such approaches features' hard requirements can be integrated into propositional formulas without changing the feature model representation based on grammars. However, in both of the men-

tioned approaches, i.e. based only on propositional formulas and based on grammars and formulas, there is no support for standardized specification of external services and/or execution platform characteristics. This can cause significant problems in combining external services due to architectural mismatches, i.e., mismatches in assumptions the developer of a reusable part, makes about the structure of the system it is to be part of [28].

In this paper, we have proposed a staged configuration process based on Semantic Web technologies including OWL ontologies and DL reasoners. We have demonstrated how capabilities of OWL can be used for seamless integration of features' hard requirements in the way that the additional complexity has not been introduced to the initial feature model. Furthermore, this approach reuses standardized domain description ontologies, herewith preventing the architectural mismatches by using the ontologies as common shared integration standards [29]. Additionally, DL reasoners can be used for verification and debugging of product configurations. However, standard DL reasoners can only verify whether the given feature configuration, including the integrity constraints is valid. For example, selection of *Bluetooth Status* feature in the feature model does not enforce an inclusion of the *Bluetooth* feature, due to the DL's descriptive and not imperative nature. Furthermore, there is often a need for definition of so called soft requirements such as for example desire for high security and low performance. Soft requirements are not captured with the approach presented in this paper. Nevertheless, we have made some steps in this direction by using the Fuzzy Datalog for specification and reasoning on feature models with soft requirements [31], and by introducing a S-AHP [30] a Stratified Analytic Hierarchy Process [32]. The approach presented in this paper, serves as a good basis for the verification of hard requirements. The descriptive nature of DL is a very good tool for product configuration verifications through the use of concept abduction [27] and justification [26].

8. Conclusions and Future Work

In this paper, we have presented an approach for leveraging Semantic Web technologies in the process of configuring feature models, a commonly-used technique for managing variability and commonality in software product line engineering. Here, we started from the previous work [Wang et al, 2007], which introduced a method for the representation of and reasoning over feature models with Description Logic. We expanded that work to enable for representing non-functional requirements (e.g., deployment platform) that each feature might have. The values of those requirements are specified in terms of specialized ontologies defining different non-functional domains. In our experiments, we used the Delivery Context Ontology, an effort of the W3C consortium. Finally, we have developed an algorithm that drives the process of staged configuration of feature models. Proving the soundness and completeness of the algorithm, we proposed a novel automated method for guiding the process of staged configuration.

By automating the staged configuration built on the description logic consistency checking reasoning service, we equipped software developers with a product configuration tool that considers both hard functional requirements (i.e., software features) and different kinds of non-functional properties (e.g., deployment platform). Our approach innovates the current configuration practice by allowing software engineers to consider definitions of non-functional properties more systematically through the use of ontologies. Such ontologies also improve the sharing of intentions between software engineers and stakeholders. Another important benefit of our proposal is that the process of staged configuration can be organized in a more disciplined way, where each stage considers one domain (i.e., ontology) of non-functional properties till the final configuration of the product is reached. Comparing to current approaches for staged configuration, our approach seamlessly integrates standardized specifications of features' requirements for external services and execution platform capabilities to avoid the problems of architectural mismatches.

In the future work, we will investigate the integration of soft requirements (e.g., preferences for higher security or lower costs) into the automated staged configuration process. Our intention is to integrate the elements of fuzzy logic into our current description logic-based feature models and the algorithm for staged configuration process. We plan to evaluate and adapt the proposed method for configuring families of business process models and service-oriented architectures.

References

- [1] K. Pohl, G. Böckle, F.J. van der Linden. *Software Product Line Engineering Foundations, Principles and Techniques*. Springer, 2005.
- [2] H. H.Wang, Y. F. Li, J. Sun, H. Zhang, and J. Pan. Verifying feature models using OWL. *Web Semant.* 5, 2 (Jun. 2007), 117-129.
- [3] I. Horrocks, P. F. Patel-Schneider, F. van Harmelen. From SHIQ and RDF to OWL: the making of a Web Ontology Language, *Web Semantics: Science, Services and Agents on the World Wide Web*, Volume 1, Issue 1, December 2003, pp. 7-26.
- [4] G. Guizzardi, *Ontological Foundations for Structural Conceptual Models*, PhD Thesis, CTIT, Centre for Telematics and Information Technology, Enschede, 2005.
- [5] D. L. Parnas. On the Design and Development of Program Families. *IEEE Transactions on Software Engineering* 2, 1 (Jan. 1976), 1-9.
- [6] M. Steger, C. Tischer, B. Boss, A. Mller, O. Pertler, W. Stolz, S. Ferber. Introducing PLA at Bosch Gasoline Systems: Experiences and Practices. In *Proceedings of the International Conference on Software Product Lines (SPLC)*, 2004.
- [7] K. Czarnecky, S. Helsen, U. Eisenecker. Staged Configuration Using Feature Models. *Proceedings of the 2004 Software Product Line Conference, LNCS 3145*, pp. 266-283.
- [8] K. Kang, S. Cohen, J. Hess, W. Novak, and A. Peterson, "Feature-Oriented Domain Analysis (FODA) Feasibility Study" Tech. Report CMU/SET-90-TR-021, SE I, Nov. 1990.
- [9] N. Kaviani, B. Mohabbati, D. Gasevic, M. Finke. Semantic Annotations of Feature Models for Dynamic Product Configuration in Ubiquitous Environments. In *Proceedings of the 4th International Workshop on Semantic Web Enabled Software Engineering*, in collaboration with ISWC 2008.
- [10] W3C. Delivery

24 *M. Bošković, E. Bagheri, D. Gašević, B. Mohabbati, N. Kaviani, M. Hatala*

Context Ontology. Available at: <http://www.w3.org/TR/dcontology/>. Accessed in September 2009.

- [11] M. Asadi, B. Mohabbati, N. Kaviani, D. Gasevic, M. Boskovic, and M. Hatala, “Model-Driven Development of Families of Service-Oriented Architectures”, Proceedings of the 1st Workshop on Feature Oriented Software Development FOSD2009, October 6, 2009, Denver, Colorado, USA, pp.95—102
- [12] M. A. Simos, Organization domain modeling (ODM): formalizing the core domain modeling life cycle. SIGSOFT Softw. Eng. Notes 20, SI (Aug. 1995), 196-205.
- [13] M. L. Griss, J. Favaro, and M. d. Alessandro. Integrating Feature Modeling with the RSEB. In Proceedings of the 5th International Conference on Software Reuse (ICSR). IEEE Computer Society, Washington, DC, 76—85. 1998.
- [14] K.C. Kang, S. Kim, J. Lee, K. Kim, E. Shin, and M. Huh. 1998. FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures. Annals of Software Engineering 5 (Jan. 1998), 143-168.
- [15] M. Mannion. Using First-Order Logic for Product Line Model Validation. In Proceedings of the Second International Conference on Software Product Lines (August 19 - 22, 2002). G. J. Chastek, Ed. Lecture Notes In Computer Science, vol. 2379. Springer-Verlag, London, 176-187. 2002.
- [16] Batory, D.: Feature models, Grammars, and Propositional formulas. In H. Obbink and K. Pohl (Eds.): Proceedings of the 9th International Conference on Software Product Lines, SPLC 2005, LNCS 3714, Springer, pp. 7–20. 2005.
- [17] K. Czarnecki and C. H. P. Kim. Cardinality-Based Feature Modeling and Constraints: A Progress Report. In Proceedings of the OOPSLA05 International Workshop on Software Factories. 2005.
- [18] P. Schobbens, P. Heymans, and J. Trigaux. Feature Diagrams: A Survey and a Formal Semantics. In Proceedings of the 14th IEEE international Requirements Engineering Conference (RE06). IEEE Computer Society, pp. 136-145. 2006.
- [19] M. Janota, and J. Kiniry. Reasoning about Feature Models in Higher-Order Logic. In Proceedings of the 11th international Software Product Line Conference IEEE Computer Society, pp. 13-22. 2007.
- [20] D. Benavides, P. Trinidad, and A. Ruiz-Corts. Automated Reasoning on Feature Models. In Proceedings of the Conference on Advanced Information Systems Engineering 2005 (CAISE2005). LNCS 3520, Springer. pp. 491—503. 2005.
- [21] D. Benavides, S. Segura, P. Trinidad, and A. Ruiz-Corts. A First Step Towards a Framework for the Automated Analysis of Feature Models. In Managing Variability for Software Product Lines: Working With Variability Mechanisms, pages 4553, 2006.
- [22] M. Mendonca, A. Wasowski, K. Czarnecki, D. Cowan. Efficient Compilation Techniques for Large Scale Feature Models. In Proceedings of the 7th International Conference on Generative Programming and Component Engineering, ACM Press, pp. 1322. 2008.
- [23] K. Czarnecky, S. Helsen, and U. Eisenecker. Staged Configuration through Specialization and Multi-level Configuration of Feature Models. Software Process Improvement and Practice, 10(2), John Wiley & Sons. pp 143—169. 2005.
- [24] A. Classen, A. Hubaux, and P. Heymans. A Formal Semantics for Multilevel Staged Configuration. In Proceedings of the Third Workshop on Variability Modelling of Software-intensive Systems, pages 5160, January 2009.
- [25] J. White, D. Benavides, B. Dougherty, D. C. Schmidt. Automated Reasoning for Multi-step Software Product-line Configuration Problems. In Proceedings of the 13th Software Product-lines Conference (SPLC2009). Volume 1. 2009.
- [26] M. Horridge, B. Parsia, and U. Sattler. Laconic and Precise Justifications in OWL.

- In Proceedings of the 7th International Conference on the Semantic Web. A. Sheth, S. Staab, M. Dean, M. Paolucci, D. Maynard, T. Finin, and K. Thirunarayan, Eds. LNCS, vol. 5318. Springer-Verlag, Berlin, Heidelberg, pp. 323-338. 2008.
- [27] S. Colucci, T. Di Noia, E. Di Sciascio, F. Donini, and M. Mongiello. Concept Abduction and Contraction in Description Logics. In Proceedings of the 16th International Workshop on Description Logics (DL'03), volume 81 of CEUR Workshop Proceedings. 2003.
 - [28] D. Garlan, R. Allen, and J. Ockerbloom. Architectural Mismatch: Why Reuse Is So Hard. *IEEE Software* 12, 6 (November 1995), 17-26. 1995.
 - [29] D. Garlan, R. Allen, J. and Ockerbloom. Architectural Mismatch: Why Reuse Is Still So Hard. *IEEE Software* 26, 4 (July 2009), 66-69. 2009.
 - [30] E. Bagheri, and D. Gasevic. How to Select the Best Software Features. Submitted to *Software*. IEEE Computer Society. 2009.
 - [31] E. Bagheri, T. Di Noia, D. Gasevic, A. Ragone. Formalizing Interactive Staged Feature Model Configuration. Submitted to the *Journal of Software Maintenance and Evolution: Research and Practice*. Wiley. 2009.
 - [32] T. L. Saaty, *The Analytic Hierarchy Process*, McGraw-Hill, 1980.