On the Use of Ontology for Dynamic Reconfiguring Software Product Line Products

Thyago Tenório
Computing Institute
Federal University of Alagoas (UFAL)
Maceió, Brazil
ttmo@ic.ufal.br

Diego Dermeval
Department of Computer and Systems
Federal University of Campina Grande (UFCG)
Campina Grande, Brazil
diegodermeval@copin.ufcg.edu.br

Ig Ibert Bittencourt
Computing Institute
Federal University of Alagoas (UFAL)
Maceió, Brazil
ig.ibert@ic.ufal.br

***Abstract*—Software Product Line (SPL) is a set of software systems that have a particular set of common features and that satisfy the needs of a particular market segment or mission. The traditional SPLs focus on building software platforms at development time. In contrast, modern systems of emerging domains (e.g., ubiquitous computing, service robotics and autonomic systems) require new settings to perform dynamic reconfiguration. In this context, Dynamic Software Product Line (DSPL) extends the SPL concept to provide an efficient way to deal with software adaptation at runtime. A key artifact in SPL is the feature model. Such model is very important in the specification of SPLs, representing the variability of the software and also supporting the instantiation of applications. However, this model has some limitations regarding its usage in DSPL. In order to effectively provide dynamic reconfiguration of features, it is necessary to represent such model in a formal way thus it can be automatically monitored, retrieved and modified during the execution of a product. Hence, we propose an ontology for feature modeling, regarding its capabilities to handle changes in the feature models, demanding less effort to be reconfigurable at runtime. In order to illustrate the use of the ontology, a set of reconfiguration scenarios in the domain of ubiquitous computing are presented.**

Keywords–Ontology; Software Product Lines; Dynamic Software Product Lines;

I. INTRODUCTION

Software Product Line (SPL) engineering is a paradigm that advocates the reusability of software artifacts and the rapid development of new applications for a particular domain. These objectives are achieved by capturing the commonalities and variabilities between the products from the same domain in variability models (e.g., feature models). Software Product Line engineering methods offer characteristics such as rapid product development, reduced time-to-market, quality improvement, and more affordable development costs [1].

The traditional methods for designing SPL focus on its construction at development time, thus each product configuration is instantiated before a product is delivered to the customer. However, the modern systems of emerging domains such as ubiquitous computing, service robotics, unmanned space and autonomic systems are increasingly requiring new mechanisms capable to reconfigure their variability models at runtime, i.e., without stopping the system's execution. In this context, Dynamic Software Product Lines (DSPL) extend existing Software Product Line engineering approaches to provide ways to handle with software adaptation at runtime [2].

One of the key artifacts used in SPL engineering is the feature model. Such model is widely used in the context of SPLs to capture the common and variable functionalities of products from a same domain. However, its informal representation has several limitations regarding its usage in DSPLs, for instance, it is difficult to automatically monitor, retrieve and modify them at runtime [2].

In order to effectively provide dynamic reconfiguration of products, it is necessary to represent feature models in a formal way, as a result it can be automatically reasoned or queried during the execution of a product. Meanwhile, some studies use ontologies as an effective way to formally represent feature models [3][4]. However, none of the existing studies on ontology-based feature modeling provides explicit elements (e.g., status of the features and product configuration model) capable to allow product reconfiguration at runtime with less effort.

Facing the potential benefits of using ontologies to represent feature models for DSPL purposes and the limitations of existing ontology-based feature modeling approaches regarding dynamic reconfigurations, we propose the OntoSPL ontology. Such ontology presents an alternative way for modeling ontology-based feature models. OntoSPL was conceived with the purpose to be as much flexible as possible, since it specifies a predefined structure of classes and properties and suggests the creation of features model as OWL instances/individuals of such structure. In addition, we present a set of SPARQL queries, in different scenarios, that can be executed to automatically reconfigure SPL products specified in OntoSPL.

The remainder of this paper is organized as follows. Section II describes in details the OntoSPL ontology. Section III presents the OntoSPL for DPSL Products and a set of SPARQL queries for reconfiguring SPL products. Section IV compares our work to related ones. Finally, Section V presents our conclusions and points out future works.

II. ONTOSPL

This section presents the OntoSPL ontology. According to the Guarino's ontologies classification [5], such ontology is a domain one, which aims to describe the main concept of Software Product Line through the feature model diagram. The variability of SPLs is commonly expressed through features represented in this model. A feature is a property of the system which is relevant to some stakeholder and is used to capture similarities and variabilities in software systems.

Feature modeling has been proposed as an approach for

 ISBN: 978-1-61208-367-4

describing variable requirements for Software Product Lines [6]. It is an important activity of the Software Product Line development process, since it is in such phase that the common and variable features of the product family are specified. In this sense, OntoSPL provides an explicit conceptualization of the essential elements involved in such diagram and is described by the following elements: concepts, properties and relations. In the sequel, the ontology is informally described (through the description of feature model elements) and its concepts, properties and relationships are further presented.

A. *Informal description of the ontology*

The ontology is inserted in the SPL Domain Engineering. It describes the concepts involved in feature modeling, as proposed by the Feature-Oriented Domain Analysis (FODA) notation [7]. A feature model provides a graphical tree-like notation that shows the hierarchical organization of features. The root of the tree represents the whole SPL node, all other nodes represent different types of features which are part of a SPL.

Features are organized in feature models and can be one the following types: mandatory, optional, alternative and or-features. The mandatory type must be present in all products derived from a Software Product Line. The optional one may or may not be included into a product derived from a SPL, hence its presence is optional. In the alternative feature, exactly one feature from a set of features must be included in a product. In the or-feature type, one or more features from a set of features can be included in a product from a SPL. Moreover, dependency rules between features may exist and can be of two types: (i) Requires, when one feature requires the existence of another feature (they are interdependent); and (ii) Excludes, when one feature is mutually exclusive to another one (they can not coexist). The Group element indicates a constraint in a set of grouped features.

A feature constraint has also a name and can be classified in Depend (Require), Exclude or Group. The Depend constraint has a name, a set of source features and a set of target features. Such constraint means that if all source features are selected in a product, then, in the same product derived from a SPL, all the target features must be selected too. The Exclude constraint has exactly the same properties of the Depend one. It only has a semantic difference, since if all source features are selected in a product then any target features may not be selected in such product. Finally, the Group constraint has a name, a set of features and a constraint type which indicates a type of the constraint on the group.

B. *Description of the classes, properties and relationships*

In this section, the classes, properties and relationships of the OntoSPL are described. Figure 1 illustrates its hierarchy of classes. Note that the description of the classes on below follows the format ”Class_Name(Class_Attribute_1, Class_Attribute_2,...,Class_Attribute_n)”.

- *SoftwareProductLine* (name, description, FeatureModel): this class represents an arbitrary Software Product Line. It has primitive elements such as: name and description. Moreover, a SPL contains a Feature Model.
- *FeatureModel* (name, Feature, FeatureConstraint): this class describes a Feature Model which represents the hierarchy organization of the features of a SPL. It has a set of features and a set of feature's constraints.

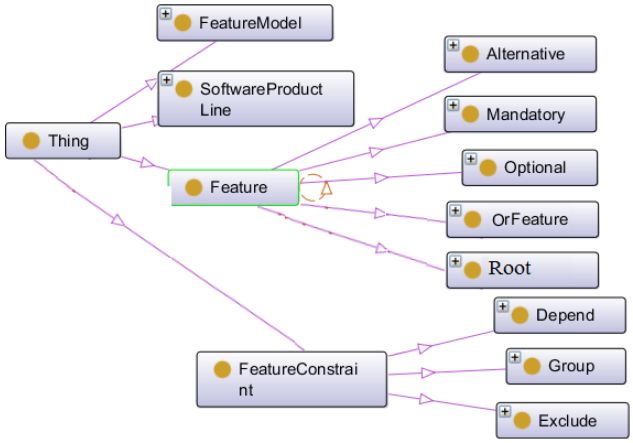

Figure 1. OntoSPL classes hierarchy (OntoViz plugin visualization).

- *Feature* (name, current_state): this class represents a resource available in the Software Product Line. It may be classified into Mandatory, Optional, Alternative, OrFeature and RootFeature:
  - *Mandatory* (name): this class represents a mandatory resource of the SPL, i.e., it must be present in all products
  - *Optional* (name): this class represents an optional resource of the SPL, i.e., it is optionally present in any product.
  - *Alternative* (name, exclusive, AlternativeFeature): this class represents an alternative resource of the SPL. An alternative resource specifies that two or more resources may not co-exist.
  - *OrFeature* (name, AlternativeFeature): this class represents an or exclusive resource of the SPL. An or exclusive resource specifies that two or more resources may or may not co-exist.
  - *RootFeature* (name): this class represents a root feature. A root feature represents the root of the features tree. It is on top of a feature model.
- *FeatureConstraint* (name): this class represents a constraint in the feature model. It may be classified into Depend, Exclude or Group:
  - *Depend* (name, SourceFeature, TargetFeature): This class represents a constraint of the Depend type. As mentioned above, it has a set of source features and a set of target features.
  - *Exclude* (name, SourceFeature, TargetFeature): this class represents a constraint of the Exclude type. As mentioned above, it has a set of source features and a set of target features.
  - *Group* (name, SetFeatures, typeConstraint): this class represents a constraint of the Group type. It has a set of features and a String typeConstraint which indicates the type of the

 ISBN: 978-1-61208-367-4

constraint. The types can be: (i) zero-or-one feature exactly (0 or 1); (ii) At-least-one feature (1 or more); (iii) Exactly-one feature (1); (iv) Any feature (0 or more); (v) All features (n);

OntoSPL specifies a set of relationships between the ontology classes. The two classes between the parentheses following the property name represents, respectively, the source and target classes of such property.

- *hasRootFeatures* (FeatureModel, RootFeature): specifies that a FeatureModel contains a set of root features (which may not be empty);
- *hasSetOfAlternativeFeatures* (Alternative, Alternative): specifies that an alternative feature must have at least one feature alternative. It is a symmetric property;
- *hasSetOfConstraints* (FeatureModel, FeatureConstraint): specifies that a FeatureModel contains a set of feature's constraints;
- *hasSetOfFeatures* (Group, Feature): specifies that a Group constraint contains a set of features (which may not be empty);
- *hasSourceFeatures* (Depend/Exclude, Feature): specifies that a Depend or Exclude constraint has a set of source features (which must have at least one feature);
- *hasTargetFeatures* (Depend/Exclude, Feature): specifies that a Depend or Exclude constraint has a set of target features (which must have at least one feature);
- *isBasedOn* (SoftwareProductLine, FeatureModel): specifies that a SPL is based on exactly one FeatureModel. It is a functional property;
- *isChildOf* (Feature, Feature): specifies that a feature is child of exactly one another Feature. It is a functional property and it is also the inverse property of isParentOf;
- *isParentOf* (Feature, Feature): specifies that a feature contains a set of children features. It is the inverse property of isChildOf.

C. Axioms of the ontology

The classes and relationships described above express a taxonomy of the OntoSPL ontology. In order to describe it in a detailed and formal way, it must be governed with axioms. All axioms of the OntoSPL are defined in description logics (DL) and the OWL syntax used to represent it is summarized in Table I. The data properties, classes axioms and object properties are, respectively, presented in Tables II, III and IV.

TABLE I. SUMMARY OF DL SYNTAX.

Notations	Explanation
$\top$	Superclass of all OWL classes
A $\sqsubseteq$ B	A is a subclass of B
A $\sqsubseteq \neg$ B	A and B are disjoint class
A $\sqcap$ B	Class intersection
A $\sqcup$ B	Class union
A $\equiv$ B	Class equivalence
$\top \sqsubseteq \forall$ P.A	Range of property is class A
$\exists$ / $\forall$ P.A	allValuesFrom/someValuesFrom restriction that for every instance of this class that has instances of property P, all some of the values of the property are members of the class A

TABLE II. DATA PROPERTIES AXIOMS OF ONTOSPL.

Data Property	Source	Data Type
description	SoftwareProductLine	String
name	SoftwareProductLine; FeatureModel;Feature; FeatureConstraint	String
exclusive	Alternative	String
typeConstraint	Group	String
currentState	Feature	Boolean

TABLE III. CLASSES AXIOMS OF ONTOSPL.

Class	**Axioms**
Alternative	$Alternative \sqsubseteq Feature$ $Alternative \sqsubseteq \neg Optional$ $Alternative \sqsubseteq \neg Mandatory$ $Alternative \sqsubseteq \neg ORFeature$ $Alternative \sqsubseteq \neg Root$
Depend	$Depend \sqsubseteq FeatureConstraint$ $Depend \sqsubseteq \neg Exclude$ $Depend \sqsubseteq \neg Group$
Exclude	$Exclude \sqsubseteq FeatureConstraint$ $Exclude \sqsubseteq \neg Depend$ $Exclude \sqsubseteq \neg Group$
Feature	$Feature \sqsubseteq \top$
Feature Constraint	$FeatureConstraint \sqsubseteq \top$
Feature Model	$FeatureModel \sqsubseteq \top$
Group	$Group \sqsubseteq FeatureConstraint$ $Group \sqsubseteq \neg Depend$ $Group \sqsubseteq \neg Exclude$
Mandatory	$Mandatory \sqsubseteq Feature$ $Mandatory \sqsubseteq \neg Optional$ $Mandatory \sqsubseteq \neg Alternative$ $Mandatory \sqsubseteq \neg ORFeature$ $Mandatory \sqsubseteq \neg Root$
Optional	$Optional \sqsubseteq Feature$ $Optional \sqsubseteq \neg Mandatory$ $Optional \sqsubseteq \neg Alternative$ $Optional \sqsubseteq \neg ORFeature$ $Optional \sqsubseteq \neg Root$
ORFeature	$ORFeature \sqsubseteq Feature$ $ORFeature \sqsubseteq \neg Mandatory$ $ORFeature \sqsubseteq \neg Alternative$ $ORFeature \sqsubseteq \neg Optional$ $ORFeature \sqsubseteq \neg Root$
Root	$Root \sqsubseteq Feature$ $Root \sqsubseteq \neg Mandatory$ $Root \sqsubseteq \neg Alternative$ $Root \sqsubseteq \neg ORFeature$ $Root \sqsubseteq \neg Optional$
Software Product Line	$SoftwareProductLine \sqsubseteq \top$

III. DSPL PRODUCTS RECONFIGURATION

In SPL engineering, the applications are built by reusing domain artifacts and by exploiting the product line variability. To create instances (products) of a Software Product Line, one must choose the features that will be present in the product, following the constraints of the features model.

A SPL product contains a specific subset of features. In the DSPL context, by contrast, the requirements (expressed as feature models) of these products vary at runtime. Thus, these models must be reasoned or queried during the execution of a product, allowing it to be self-reconfigured at runtime, as required by DSPL engineering.

In this sense, OntoSPL may represent a single product based on a SPL feature model specified. It is used as the decision model on where the SPL variations are selected. Hereafter, we present how to represent a product on the ontology. Then, we present a set of dynamic reconfiguration scenarios which were applied to a running example of the

 ISBN: 978-1-61208-367-4

TABLE IV. OBJECT PROPERTIES AXIOMS OF ONTOSPL.

Object Property	Axioms
hasRootFeatures	$\exists$ hasRootFeatures Thing $\sqsubseteq$ FeatureModel
	$\top \sqsubseteq \forall$ hasRootFeatures ($\exists$ hasRootFeatures RootFeature)
hasSetOfAlternativeFeatures	hasSetOfAlternativeFeatures $\equiv$ hasSetOfAlternativeFeatures$^-$
	$\exists$ hasSetOfAlternativeFeatures Thing $\sqsubseteq$ Alternative
	$\top \sqsubseteq \forall$ hasSetOfAlternativeFeatures ($\geq$ 1 hasSetOfAlternativeFeatures Alternative)
hasSetOfConstraints	$\exists$ hasSetOfConstraints Thing $\sqsubseteq$ FeatureModel
	$\top \sqsubseteq \forall$ hasSetOfConstraints ($\forall$ hasSetOfConstraints (Depend $\sqcup$ Exclude $\sqcup$ Group))
hasSetOfFeatures	$\exists$ hasSetOfFeatures Thing $\sqsubseteq$ Group
	$\top \sqsubseteq \forall$ hasSetOfFeatures ($\exists$ hasSetOfFeatures Feature)
hasSourceFeatures	$\exists$ hasSourceFeatures Thing $\sqsubseteq$ Depend
	$\exists$ hasSourceFeatures Thing $\sqsubseteq$ Exclude
	$\top \sqsubseteq \forall$ hasSourceFeatures ($\geq$ 1 hasSourceFeatures Feature)
hasTargetFeatures	$\exists$ hasTargetFeatures Thing $\sqsubseteq$ Depend
	$\exists$ hasTargetFeatures Thing $\sqsubseteq$ Exclude
	$\top \sqsubseteq \forall$ hasTargetFeatures ($\geq$ 1 hasTargetFeatures Feature)
isBasedOn	$\top \sqsubseteq \leq$ 1 isBasedOn Thing
	$\exists$ isBasedOn Thing $\sqsubseteq$ SoftwareProductLine
	$\top \sqsubseteq \forall$ isBasedOn (= isBasedOn FeatureModel)
isChildOf	isChildOf $\equiv$ isParentOf $^-$
	$\top \sqsubseteq \leq$ 1 isChildOf Thing
	$\exists$ isChildOf Thing $\sqsubseteq$ Feature
	$\top \sqsubseteq \forall$ isChildOf (= isChildOf Feature)
isParentOf	isChildOf $\equiv$ isParentOf$^-$
	$\top \sqsubseteq \leq$ 1 isParentOf$^-$ Thing
	$\exists$ isParentOf Thing $\sqsubseteq$ Feature
	$\top \sqsubseteq \forall$ isParentOf Feature

ubiquitous domain published in the literature, a simplified smart hotel [8].

A. DPSL products configuration

OntoSPL supports the instantiation of products based on the SPL in order to facilitate the reconfiguration of the product when it is necessary. In this sense, the property $current\_state$ of the Feature class indicates whether the feature belongs or not to a particular product. This property presents the following range of values: $\{"eliminated" : string, "selected" : string\}$. Such a property can only receive the values: selected, case the feature must be in the product, or eliminated, case the feature must not be in the product.

Hence, one can reason in the ontology to perform dynamic reconfiguration in an arbitrary product. After defining the features that may be present in the product to be created, there is only necessary to set the property $current\_state$ for each feature instantiated in a product.

For instance, Figure 2 depicts the feature model of the Simplified Smart Hotel (extracted from [8]). Its mandatory features are represented by a small filled circle above the feature name (e.g., *Automated Illumination*). Optional features are represented by a small circle not filled (e.g., *Piped Music*, *Security* and *Alarm*). Alternative features share the same parent feature and are graphically represented by a not filled arc below the parent feature; such arc means that one and only one of the child features must be chosen (e.g., *Silent Alarm*,*Siren* and *Visual Alarm*). Finally, the or-features (e.g., *Infrared Sensor* and *Volumetric Sensor*) are represented by a filled arc, in a similar way to alternative features.

As shown in Figure 2, the gray features indicate which are the selected features for a product configuration. The current configuration of the simplified smart hotel includes the *Piped Music*, *Security*, *In Room Detection*, *Volumetric Sensor*, *Alarm*, *Silent Alarm*, *Automated Illumination features* and *Lighting by Occupancy* features. However, the white features represent potential variants of a product configuration [8].

B. Dynamic reconfiguration scenarios

Once a product is specified in the OntoSPL ontology, it can be reconfigured dynamically according to different scenarios. This section describes scenarios directly related to changes in the type of a feature and also changes related to product configuration.

In this sense, to specify this product in the OntoSPL ontology, one must set the *current_state* property of these features to the "selected" value and "eliminated" for the others features. For instance, in our running example, the gray features of Figure 2 assume the "selected" value in the OntoSPL, whereas the white features assume the "elimated" value.

In the sequel, we present three scenarios (specified in SPARQL 1.1 [9]) which can be executed for changing SPL products at runtime. Note that these scenarios are presented by applying them to our running example, but they have generic purposes.

1) Changing an optional feature to mandatory feature: Changing an optional feature to a mandatory one demands a simple change scenario in the requirements of a particular feature. Such change does not have a great impact in the feature model, since it is not necessary to create/remove features on the feature model.

Let's suppose, for an arbitrary reason, that an Smart Hotel requires security requirements. In this context, there is the need to make the "Security" feature a mandatory one. This way, the query on Figure 3 could be used to perform such reconfiguration.

This query updates the property *type* of the feature being changed. Note that, the optional type is deleted while the mandatory type is inserted. As consequence, the feature will be mandatory in the product.

2) Selecting an optional feature in a product: This is one of the most common changing scenarios in SPLs. Usually, a product is firstly generated according to customer's needs at

 ISBN: 978-1-61208-367-4

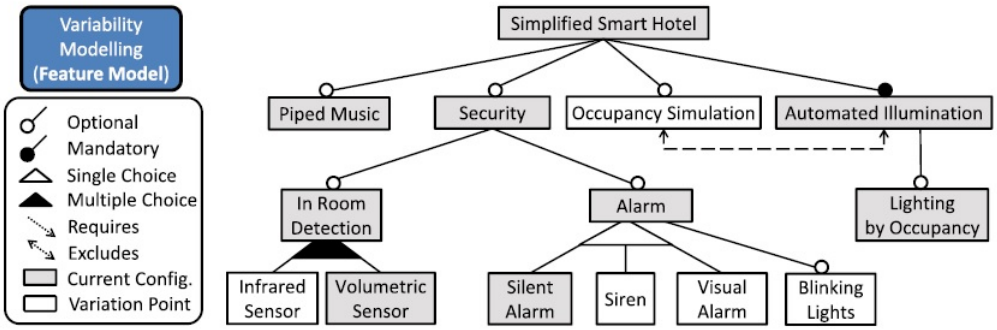

Figure 2. Feature model of the simplified smart hotel (extracted from [8]).

```
PREFIX spl:<http://nees.com.br/ontologies/SPL.owl#>
PREFIX rdf:<http://w3.org/1999/02/22-rdf-syntax-ns#>
DELETE {?x rdf:type spl:Optional.}
INSERT {?x rdf:type spl:Mandatory.}
WHERE{?x spl:name "Security".}
```

Figure 3. Changing an Optional feature to Mandatory feature.

development time and includes several features and, in such moment, a optional feature would not be selected to be in the product. However, in a later moment, it is possible that a client demands the inclusion of an optional feature that was not previously addressed by the configured product.

In the DSPL context, it is important to specify a mechanism which can reconfigure the product to reflect the current requirements of the client. For instance, in our running example, despite the hotel having alarm, the feature *Bliking lights* was not originally selected to be in the Smart Hotel configuration. However, in a changing scenario, the client would like to include it in his product.

To achieve such change, it is necessary to change the property that indicates that the feature is present or not in the system in the ontology of the product at runtime. The update of this property can be realized by the SPARQL query on Figure 4.

```
PREFIX spl:<http://nees.com.br/ontologies/SPL.owl#>
DELETE {
?x spl:current_state "eliminated".}
INSERT {
?x spl:current_state "selected".}
WHERE{?x spl:name "BlikingLights"}
```

Figure 4. Selecting An Optional Feature in a Product.

As can be seen in the query, after setting the status of the *Bliking Lights* feature as ”Selected”, such feature is included in the product.

3) Changing an Alternative Feature: Usually, it is necessary to select alternative ways to realize a product requirement. The alternative type of features specifies a design space of variations on which a product can use. A product configuration requires the selection of one of the variants on such kind of features, but it is possible that a client would be not satisfied with the variant selected and wants to reconfigure a product with another variant.

For instance, in our running example, the selected variant of alarm is the silent type. However, one could require to change from the *Silent Alarm* feature to the *Siren* one.

Figure 5 specifies a SPARQL query which makes such variant change. This query is similar to the one presented in Subsection III-B2, however, it is not only a selection of a new feature, but rather the substitution of one feature by another. Thus, there is the need to remove the existing feature and then, add the new feature.

```
PREFIX spl:<http://nees.com.br/ontologies/SPL.owl#>
DELETE{
?x pSPL:current_state "selected".
?y pSPL:current_state "eliminated"
}
INSERT{
?x pSPL:current_state "eliminated".
?y pSPL:current_state "selected"
}
WHERE{
?y spl:name "Siren".
?x spl:name "SilentAlarm"
}
```

Figure 5. Changing an Alternative Feature.

As defined in the query, the *Silent Alarm* would be eliminated from the hotel and the *Siren* would be enabled in the product. After executing this query, the selected type of *Alarm* is the *Siren* feature.

IV. Related Works

Using ontologies in the development of SPLs has been addressed by several studies in the literature. In fact, it was found some studies with the particular aim of using ontology for representing feature models [3][4]. Furthermore, we have

 ISBN: 978-1-61208-367-4

found only one study which makes use of ontology-based feature modeling in the design of DSPLs [10].

Wang et al. [3] presents a technique to design ontology-based feature models, by representing feature models as OWL classes and properties. Moreover, they use OWL reasoning engines to check for inconsistences of feature configurations automatically. However, since the features in such modeling are represented as OWL classes and properties, every change in the feature model requires a structural modification in the ontology.

OntoSPL ontology presents an alternative way for modeling ontology-based feature models. It proposes a predefined structure of classes and properties and suggests the creation of features model as OWL instances/individuals of such structure. This alternative way for feature modeling is more suitable than the one of [3] for dynamic reconfigurations of features.

In order to change a feature model using Wang's ontology [3], for instance, adding a new feature, it would be necessary to change the structure of the ontology and then it would be also necessary to generate the ontology mapping code again. Thus, applying such changes in an application would require to stop the execution of the system. On the other hand, using the OntoSPL, changes are performed at the instances level. This characteristic allows to change instances at runtime, i.e., it is not necessary to generate the ontology mapping code again and hence, it would not be necessary to stop the application.

The work by Zaid et al. [4] also presents an ontology to represent feature models based on OWL instances which is similar to our ontology. However, it is focused on the automatic consistency verification of feature models and it was not conceived to support dynamic reconfiguration of features. Thus, it does not consider important issues regarding changes at runtime, for example, properties related to the status of the feature.

Regarding the use of ontology in the development of dynamic Software Product Lines, Kaviani et al. [10] use ontology to annotate feature models covering non-functional requirements modeling in the context of ubiquitous environments. However, it also represents feature models as proposed in [3], thus the same limitations regarding the impact of changes in the feature model are also applicable to it. Moreover, it is noteworthy that the dynamic reconfiguration effort using the OntoSPL ontology is lower, since it is only necessary to change product configurations through SPARQL queries without needing to generate code to manipulate a product.

V. Conclusion and Future works

In this paper, we presented the OntoSPL ontology, which is used to specify feature models in a formal way with the special aim to support automatic reconfigurations of products in the context of Dynamic Software Product Line. To illustrate how to reconfigure DSPL products using such ontology, we also specified three SPARQL queries that were applied to an existent running example in the literature.

This study can be considered as a first step towards selecting a suitable way for formalizing feature models to be used in the context of DSPLs. Future works should include the conduction of a controlled experiment in different contexts to evaluate the effectiveness of OntoSPL in comparison with other ontologies regarding its capabilities (e.g., time to realize some change, flexibility and so on) for performing reconfiguration at runtime. Moreover, we intend to incorporate some consistency checking mechanism in OntoSPL to validate product reconfigurations. We also intend to define a DSPL process based on the OntoSPL.

Acknowledgments

This work has been supported by the Brazilian institutions: Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) and Coordenacão de Aperfeicoamento de Pessoal de Nível Superior (CAPES).

References

[1] K. Pohl, G. Bockle, and F. Van Der Linden, Software product line engineering. Springer, 2005, vol. 10.

[2] M. Hinchey, S. Park, and K. Schmid, "Building dynamic software product lines," Computer, vol. 45, no. 10, 2012, pp. 22–26.

[3] H. H. Wang, Y. F. Li, J. Sun, H. Zhang, and J. Pan, "Verifying feature models using owl," Web Semantics: Science, Services and Agents on the World Wide Web, vol. 5, no. 2, 2007, pp. 117–129.

[4] L. A. Zaid, F. Kleinermann, and O. De Troyer, "Applying semantic web technology to feature modeling," in Proceedings of the 2009 ACM symposium on Applied Computing. ACM, 2009, pp. 1252–1256.

[5] N. Guarino, Formal Ontology in Information Systems: Proceedings of the 1st International Conference June 6-8, 1998, Trento, Italy, 1st ed. Amsterdam, The Netherlands, The Netherlands: IOS Press, 1998.

[6] K. Czarnecki, C. H. Peter Kim, and K. T. Kalleberg, "Feature models are views on ontologies," in Proceedings of the 10th International on Software Product Line Conference, ser. SPLC '06. Washington, DC, USA: IEEE Computer Society, 2006, pp. 41–51.

[7] K. Kang, S. Cohen, J. Hess, W. Nowak, and S. Peterson, Feature-Oriented Domain Analysis (FODA) Feasibility Study, 1990.

[8] C. Cetina, P. Giner, J. Fons, and V. Pelechano, "Prototyping dynamic software product lines to evaluate run-time reconfigurations," Science of Computer Programming, vol. 78, no. 12, 2013, pp. 2399 – 2413, special Section on International Software Product Line Conference 2010 and Fundamentals of Software Engineering (selected papers of {FSEN} 2011).

[9] W3C, "Sparql 1.1 query language," Available in http://www.w3.org/TR/2012/PR-sparql11-query-20121108/, retrieved: October, 2014.

[10] N. Kaviani, B. Mohabbati, D. Gasevic, and 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 International Semantic Web Conference (ISWC), Karlsruhe, Germany, 2008.

 ISBN: 978-1-61208-367-4