Modeling Product Line Variants – Semantic Web Approach

Shamim Ripon, Moshiur Mahamud Piash, and Sheikh Md. Alam Hossain

Abstract—Variant management is crucial to successful to product line software (SPL) development. Feature diagram is the most widely used notation to model variants of product line. Feature diagram captures both common and variants features of SPL. Various approaches have been proposed to model the variants in a feature model. However, there is a lack of proper formal semantics in most of these approaches. This paper presents our work-in progress feature modeling approach using semantic web, in particular OWL-DL, where the features and relations in the feature diagram are represented in a concise and formally sound way. This representation is then checked for variant feature consistency using OWL reasoning tool. Such representation facilitates the search and maintenance of feature models and enhances knowledge sharing and transfer within a reusable engineering context.

Index Terms—OWL-DL, software product line, semantic web, variant management.

I. INTRODUCTION

Software product line (SPL) is a software intensive system sharing a common and managed set of features that satisfy the needs of a particular market segment or mission and that are developed from a set of core assets in a prescribed way [1]. Product line technology is a way of improving the software development lifecycle and reuse by providing facilities to reuse the model of the system family. By reusing rather than recreating the work products of the system families, it is possible to increase the productivity and decrease the possible errors significantly.

Domain and application engineering are the two main phases of SPL development [2]. A detailed domain analysis is performed in domain engineering by identifying the commonalities and variabilities of various aspects of the domain. Domain knowledge is captured in a reusable manner. Feature modeling [3] plays an important role for modeling different aspects of family of systems. It models the commonality and variability in a tree structure, and describes the interdependencies of product family features.

There are various reuse mechanism proposed for feature model, such as FODA (Feature Oriented Domain Analysis) [4], FORM (Feature Oriented Reuse Method) [5] and FeatURSEB [6]. However, due to the lack of formal semantic for feature models, no automated tools are available to check the consistency and correctness of feature configuration of a particular product. Various approaches have been suggested to model feature diagram. In our earlier work [7], [8] we model and analyze SPL variants using First-order logic. Although such modeling technique supports sound formal definition, it still lacks the ontological and sharable knowledge description of the features and concepts.

To capture domain knowledge and common vocabularies in any field ontologies have shown itself an acceptable paradigm [9]. It is also necessary to process and exploit knowledge in a computer system. Among the various available approaches for knowledge representation, ontologies appear to be a promising solution due to its ability to make the domain knowledge computer readable and processable. Besides various inference algorithms and tools are available to infer new knowledge from the existing.

Semantic web technology can provide a meaningful and shared ontological description of the domain. Web Ontology Language (OWL) [10] is one of the most expressive languages for specifying, publishing and sharing ontologies. It also provides proper mechanism to exploit the potential of Semantic Web by prescribing how data are defined and related. Importantly, OWL not only facilitates better machine interoperability than that of XML, RDF, RDFS etc. but also has formal semantics and support for defining additional vocabulary. It is therefore evident that semantic web technology, OWL in particular, can be used to represent a particular domain and define the relationship of various features within that domain. Among the various available dialects, this paper uses OWL-DL which is based on Description Logic (DL) [11]. Description logic has already been successfully applies to solving various complex configuration problems [11], [12] as well as checking consistency in UML diagrams [13].

This paper formally models and verifies the variants of SPL using OWL-DL. Semantic web mechanism can integrate meaningful description and semantic information into SPL feature models. This integration will facilitate the search and maintenance of feature models and enhance knowledge sharing and transfer within a reusable engineering context. We use Protégé [14] graphical interface for ontology editing and visual display of both feature models and feature properties. We illustrate the semantic web approach by using a “Hall Booking System” product family.

In the rest of the paper, Section II gives a brief overview of the Hall Booking System. In Section III we first illustrate the modeling of feature models and the relationship among the features using OWL-DL. We describe and model six types of feature relations as well as two additional constraints. We then encode the OWL-DL into Protégé tool for graphical editing and consistency checking. We are yet to complete the consistency checking of the feature model. Finally, Section IV concludes the paper and outlines our future plans.
II. HALL BOOKING SYSTEM OVERVIEW

We use Hall Booking System family to illustrate our variability modelling mechanism. The system is used in academic institutions to reserve tutorial rooms and lecture halls, at companies to reserve meeting rooms, and at hotels to reserve rooms and conference facilities, etc. In another sense, the system can be used for either academic or non-academic purposes. Users can manage their own reservation with the system. The main purpose and the core functionality are similar across the Hall Booking System family; however, there are many variants on the basic theme. One of the basic variants is the charging of booking system. Whenever the system is used for academic purposes, no charge is needed for booking halls, whereas there may be a need to charge for booking halls in other areas. In some systems, there are facilities available for seasonal booking as well as multiple bookings.

A part of the features of Hall Booking System is shown in Fig. 1. Extensions of feature diagram described in [3] have been used here. Mandatory features appear in all the members of the family whereas variant features appear in some members of the family. Variant features are also classified as Optional, Alternative and Or features. An example of optional feature is Reservation Charge option. An alternative feature describes one of many features. An example of alternative feature is Reservation Mode which can be either Single or Block. An or-feature describes any of many features. For example a Block Reservation can be made by multiple rooms or multiple times or by both. Variants may depend on other variants. The types of feature analysed in this paper is depicted in Table I.

![Fig. 1. Hall booking system feature diagram](image)

III. MODELING FEATURES USING OWL

Various feature relations are modeled using OWL language constructs. By using OWL-DL we model six types of relations, namely mandatory, optional, alternative, or, optional alternative and optional or. Two additional constraints: requires and excludes are also modeled. Modeling feature models using OWL have several advantages, such as facilitating feature model storing, sharing and distributing and assisting cooperative designing. OWL representation and verification of variants in feature model are the focus in this paper.

First OWL ontology is built for various nodes and edges in the feature model. The ontology is constructed in a number of steps.

**Step 1:** We identify the nodes (concepts and features) present in a feature diagram. Each node in the diagram is modeled as an OWL class. Moreover, we assert that these classes are mutually disjoint. In OWL, all classes are assumed to overlap unless it is otherwise stated that they are disjoint with each other using a disjoint axiom. By default, we assume that features with different names are distinct.

**Step 2:** For each of these nodes in the diagram, we create a Rule class. This Rule class has two kinds of conditions: firstly, a necessary and sufficient (NS, EquivalentClass) condition, using an existential restriction to bind the Rule node to the corresponding feature node in the diagram; and secondly, a number of (possibly 0) necessary (N, subClassOf) constraints later, serving two purposes:
1) To specify how each of its child features is related to this node, capturing the various relations between features.
2) To specify how this feature node is constrained by other features, in the form of requires, excludes, Optional Alternative and Optional Or.

**Step 3:** The root concept and features in a feature diagram are interrelated by various feature relations, represented by different edge types in the diagram. In our OWL model, for each of these edges, we create an object-property. We assert that the range of the property is the respective feature class.

The OWL syntax used in this paper is summarized in Table II. For a parent feature A and for each of its child features B₁, B₂, . . ., Bₙ, the initial modeling produces the following

![TABLE I: FEATURE TYPES](image)

<table>
<thead>
<tr>
<th>Type</th>
<th>Notation</th>
<th>Type</th>
<th>Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mandatory</td>
<td>A</td>
<td>Or</td>
<td>A</td>
</tr>
<tr>
<td>Optional</td>
<td>A</td>
<td>Optional Or</td>
<td>A</td>
</tr>
<tr>
<td>Alternative</td>
<td>A</td>
<td>Optional</td>
<td>A</td>
</tr>
</tbody>
</table>

85
ontology.

\[
A \subseteq T \quad \text{has}A \subseteq \text{ObjectProperty}
\]
\[
\text{ARule} \subseteq T \quad T \subseteq \forall \text{has}A \cdot A
\]
\[
\text{B} \subseteq T \quad \text{has}B \subseteq \text{ObjectProperty}
\]
\[
\text{BRule} \subseteq T \quad T \subseteq \exists \text{has}B \cdot B
\]

Now we are ready to model the feature relations using the ontology. The general definition of each of the six feature relations will be shown, based on the above feature ontology. The ontology will be constructed incrementally to show the modeling of various feature relations as well as additional constraints.

### A. Mandatory

A mandatory feature is included if its parent feature is included. For each of the mandatory features \(B_1, B_2, \ldots, B_n\) of a parent feature \(A\), we use one \(N\) constraints in \(\text{ARule}\) to model it. It is a someValuesFrom restriction on \(\text{has}B_i\), stating that each instance of the rule class must have some instance of \(B_i\) class for \(\text{has}B_i\). The following ontology fragment shows the modeling of mandatory feature set and parent feature \(A\).

\[
\text{ARule} \subseteq \exists \text{has}B_i \cdot B_i \quad \text{for} \ i \leq n
\]

It can be seen from Fig. 1 that the root node has a mandatory child feature Reservation Mode, which is itself a non-leaf node. We create two new classes (parent and child) for these two non-leaf nodes.

### B. Optional

An optional feature may or may not be included in a diagram, if its parent is included. For each of the optional features \(B_1, B_2, \ldots, B_n\) of a parent feature \(A\), no additional statements are required to model this relationship. In that case they are denoted by.

\[
B_i \subseteq T \quad \text{has}B_i \subseteq \text{ObjectProperty}
\]
\[
\text{BRule} \subseteq T \quad \exists \text{has}B_i \cdot B_i
\]

“Reservation Charge” is an optional feature of Hall Booking and it may, or may not be included in a configuration of Hall Booking System. As it is a non-leaf node, we create one new class \(\text{ReservCharge}\) for Reservation Charge and one object-property \(\text{hasReservCharge}\). The ontology is augmented as follows. Note that no new restriction on HallBooking is added.

\[
\text{ReservCharge} \subseteq T \quad \exists \text{hasReservCharge} \cdot \text{ReservCharge}
\]

### C. Alternative

One and only one feature from a set of alternative features can be included, if their parent feature is included in a configuration. Hence, for a set of alternative features \(B_i\) and \(B_j\) are two children of parent feature \(A\), we use disjunction of someValuesFrom restrictions over \(\text{has}B_i\) and \(\text{has}B_j\) to ensure that some feature will be included. We use the complement of distributed disjunction of the conjunction of two someValuesFrom restrictions to ensure that only one feature can be included. The or symbol \(\lor\) represents distributed disjunction.

\[
\text{ARule} \subseteq ((\exists \text{has}B_i \cdot B_i) \lor (\exists \text{has}B_j \cdot B_j))
\]

for \(1 \leq i \leq n\) and \(1 \leq j \leq n\)

\[
\text{ARule} \subseteq \neg ((\exists \text{has}B_i \cdot B_i) \land (\exists \text{has}B_j \cdot B_j))
\]

for \(1 \leq i \leq n\) and \(1 \leq j \leq n\)

Fig. 1 shows that features Block and Single are alternative features of Reservation mode. We model this relation as follows.

\[
\text{ReservMode} \subseteq T \quad \exists \text{hasReservMode} \subseteq \text{ObjectProperty}
\]
\[
\text{ReservModeRule} \subseteq T \quad \text{ReservModeRule} \subseteq \exists \text{hasReservMode} \cdot \text{ReservMode}
\]

<table>
<thead>
<tr>
<th>Notations</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\top)</td>
<td>Superclass of all OWL classes</td>
</tr>
<tr>
<td>(\subseteq)</td>
<td>A is a subclass of (B)</td>
</tr>
<tr>
<td>(\land)</td>
<td>(A) and (B) are disjoint class</td>
</tr>
<tr>
<td>(\lor)</td>
<td>Class intersection</td>
</tr>
<tr>
<td>(\equiv)</td>
<td>Class union</td>
</tr>
<tr>
<td>(\equiv)</td>
<td>Class equivalence</td>
</tr>
<tr>
<td>(\forall)</td>
<td>Range of property is class (A)</td>
</tr>
<tr>
<td>(\exists)</td>
<td>allValuesFrom/someValuesFrom restriction, giving the class 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)</td>
</tr>
</tbody>
</table>

---

86

**TABLE II: SUMMARY OF OWL SYNTAX**

<table>
<thead>
<tr>
<th>Notations</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\top)</td>
<td>Superclass of all OWL classes</td>
</tr>
<tr>
<td>(\subseteq)</td>
<td>A is a subclass of (B)</td>
</tr>
<tr>
<td>(\land)</td>
<td>(A) and (B) are disjoint class</td>
</tr>
<tr>
<td>(\lor)</td>
<td>Class intersection</td>
</tr>
<tr>
<td>(\equiv)</td>
<td>Class union</td>
</tr>
<tr>
<td>(\equiv)</td>
<td>Class equivalence</td>
</tr>
<tr>
<td>(\forall)</td>
<td>Range of property is class (A)</td>
</tr>
<tr>
<td>(\exists)</td>
<td>allValuesFrom/someValuesFrom restriction, giving the class 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)</td>
</tr>
</tbody>
</table>
The last two restrictions ensure that one and only one feature from the set of alternative features can be included.

D. Or

At least one from a set of available features is included, if the parent feature is included. For a set of Or features \( B_1, B_2, \ldots, B_n \) of a parent feature \( A \), we need to use a disjunction of someValuesFrom restrictions to model this relation.

\[ \text{ARule} \equiv \bigvee (\exists \text{hasBi} \cdot \text{Bi}), \text{ for } 1 \leq i \leq n \]

It may be noticed that the definition of Or is very similar to that of Alternative, with the omission of the negation of distributed disjunction to allow for multiple or features to be included. In Fig. 1, the feature Reservation Charge has a number of Or features. We use the following constructs to model it.

\[ \text{ReservationCharge} \equiv \top \]
\[ \text{haveReserveCharge} \equiv \text{ObjectProperty} \]
\[ \text{ReservationChargeRule} \equiv \top \]
\[ \text{ReserveChargeRule} \equiv \exists \text{hasReserveCharge} \cdot \text{ReserveCharge} \]
\[ \text{Deposit} \equiv \top \]
\[ \text{hasDeposit} \equiv \text{ObjectProperty} \]
\[ \text{DepositRule} \equiv \top \]
\[ \text{ReserveChargeRule} \equiv \exists \text{hasDeposit} \cdot \text{Deposit} \]
... 
\[ \text{ReserveChargeRule} \equiv (\exists \text{hasDeposit} \cdot \text{Deposit}) \bigvee (\exists \text{hasTax} \cdot \text{Tax}) \bigvee (\exists \text{hasBasic_Charge} \cdot \text{Basic_Charge}) \bigvee (\exists \text{hasDiscount} \cdot \text{Discount}) \]

E. Optional Or

One or more optional feature may be included when parent feature is included. There is no Optional Or feature in our brief case study but a detailed feature model consists of such features. As shown in Table 2, \( A \) is parent feature of child features \( B \) and \( C \). The OWL-DL representation is defined as follow:

\[ B \equiv \top \]
\[ \text{hasB} \equiv \text{ObjectProperty} \]
\[ \text{BRule} \equiv \exists \text{hasB} \cdot \text{B} \]
\[ C \equiv \top \]
\[ \text{hasC} \equiv \text{ObjectProperty} \]
\[ \text{CRule} \equiv \exists \text{hasC} \cdot \text{C} \]
\[ \text{ARule} \equiv (\exists \text{hasB} \cdot \text{B}) \bigvee (\exists \text{hasC} \cdot \text{C}) \]

F. Optional Alternative

One feature from a set of alternative features may or may not be included if parent is included. As in Table I, \( A \) is parent feature of child features \( B \) and \( C \). We can write the OWL-DL representation as follow:

\[ A \equiv \top \]
\[ \text{hasA} \equiv \text{ObjectProperty} \]
\[ \text{ARule} \equiv \top \]
\[ \text{ARule} \equiv \exists \text{hasA} \cdot \text{A} \]
\[ B \equiv \top \]
\[ \text{hasB} \equiv \text{ObjectProperty} \]
\[ \text{BRule} \equiv \exists \text{hasB} \cdot \text{B} \]
\[ C \equiv \top \]
\[ \text{hasC} \equiv \text{ObjectProperty} \]

G. Requires

A feature may depend on some other features; hence its presence in a feature configuration requires the appearance of the others. For a given feature \( A \) and a set of features \( B_1, B_2, \ldots, B_n \) that \( A \) requires, besides the NS condition that binds \( \text{ARule} \) to \( A \), we make sure that each of the \( B_i \) features appears in a configuration if \( A \) is present.

\[ \text{ARule} \equiv \exists \text{hasBi} \cdot \text{Bi}, \text{ for } 1 \leq i \leq n \]

In Fig 1, if ‘Fax’ feature requires ‘Printed Paper’, then its OWL representation is as follows.

\[ \text{FaxRule} \equiv \top \]
\[ \text{Fax} \equiv \exists \text{hasPrinted_Paper} \cdot \text{Printed_Paper} \]

H. Excludes

The presence of a feature may be inhibited by that of some other features. We say the appearance of a feature in a configuration excludes the appearance of some other features. For a given feature \( A \) and a set of features \( B_1, B_2, \ldots, B_n \) that \( A \) excludes, we make sure, using the negation of someValuesFrom restriction on hasBi property, that \( \text{ARule} \) does not have any \( B_i \) feature.

\[ \text{ARule} \equiv \neg (\exists \text{hasBi} \cdot \text{Bi}) \text{ for } 1 \leq i \leq n \]

We encoded the OWL-DL models of the Hall Booking System features into Protégé tool. Protégé gives an immediate feedback to the encoded logic by showing the corresponding feature graph ensuring the preliminary correctness of the syntax. Fig. 2 shows the feature graph created in Protégé.

Fig. 2. Protégé screenshot

IV. CONCLUSION

Successful development of software system families requires appropriate organisation and management of the products involved. A significant characteristic of developing system families is the management of variabilities, which is a
crucial success factor of system family approach. This paper presents our work-in-progress modeling and verification of product line variants using semantic web technology. OWL-DL is used to represent feature models and configuration in a concise and unambiguous way. Features are represented as OWL classes and relations as properties. OWL ontologies provide a suitable platform for the development of semantically aware software product line allowing the knowledge within the feature model to be shared among the reusable features of the SPL. We are yet to check the consistency of feature model as well as configurations. We are currently using RACER [15] tool to automatically check the feature model inconsistency.

REFERENCES


Shamim Ripon is an assistant Professor in the Department of Computer Science and Engineering, East West University, Dhaka, Bangladesh where he leads Software Engineering and Formal Method Research Group. Previously, he was a Research Associate in the Department of Computing Science, University of York, UK and Research Fellow in the Department of Computing Science, University of Glasgow, UK. He also served as a Lecturer in Khulna University, Bangladesh. He is a member of IAENG. Dr. Ripon holds a B.Sc. in Computer Science and Engineering from Khulna University, MSc in Computer Science from National University of Singapore and PhD in Computer Science from University of Southampton, UK. His research interests focus on the Requirement Engineering, Software Product Line, Semantic Web, Natural Language Processing. His current research examines the formal representation and verification of knowledge based requirement specification.

Moshiur Mahamud Piash is a BSc final year student in the Department of Computer Science and Engineering, East West University, Dhaka, Bangladesh where he is involved in the Software Engineering and Formal Method Research Group. Mr. Piash is currently working in the product line based requirement verification using semantic web. His is also interested in Web services and Intelligent Agent.

Sheikh Md. Alam Hossain is a BSc final year student in the Department of Computer Science and Engineering, East West University, Dhaka, Bangladesh where he is involved in the Software Engineering and Formal Method Research Group. Mr. Alam is currently working in the product line based requirement verification using semantic web. His research interests include Web services and Knowledge Based Software Engineering.