Applying Semantic Web Technology to Feature Modeling

Lamia Abo Zaid
Vrije Universiteit Brussel
Pleinlaan 2
1050 Brussels
+32 2 629 3754
Lamia.Abo.Zaid@vub.ac.be

Frederic Kleinermann
Vrije Universiteit Brussel
Pleinlaan 2
1050 Brussels
+32 2 629 5713
Frederic.Kleinermann@vub.ac.be

Olga De Troyer
Vrije Universiteit Brussel
Pleinlaan 2
1050 Brussels
+32 2 629 3504
Olga.DeTroyer@vub.ac.be

ABSTRACT
Feature models are models used to capture differences and commonalities between software features, enabling the representation of variability within software. There are many variations of feature models and different notations are often used to represent the same information. Currently support for validating or integrating feature models is missing. In this paper, we provide an ontology framework for feature modeling which consists of an ontology that formally provides a specification for feature models. In addition, we provide means to integrate segmented feature models and provide a rule based model consistency check and conflict detection. We use SWRL rules to implement the rules and a DL reasoner to evaluate the rules and infer extra interesting information regarding the variability of the software.

Categories and Subject Descriptors
H.4.2 [**Types of Systems**]: knowledge and information management of feature models.

Keywords
Feature models, software variability, ontologies, OWL, SWRL

1. INTRODUCTION
Introducing and managing variability in software products is a non trivial task. The need for variable software is driven by the increase of software demands, and the large similarity in software delivered to different customers and/or for different platforms. Variable software has a common architecture with a number of reusable assets [1]. By combining these assets products with different flavors can be created. These assets are called *features*.

A feature can be seen as an increment in the program's functionality [1]. Variability in software is specified by defining a set of possible features that distinct products could hold. Some features will be common to all products while others will differ from product to product.

Feature models (also known as feature diagrams) are tree-like structured models used to capture differences and commonalities between software features, enabling the representation of software variability [1]. Furthermore they define the rules and constraints that make up a valid product.

Software systems have grown in terms of the number of features they hold (which could be up to a few thousand), and the complexity of relations and dependencies between these features [2]. Thus the process of efficiently insuring the correctness of the features models representing the system becomes difficult. Furthermore the growth in size calls for distributing the process of feature model creation. Two forms of distribution hold; distribution in terms of functionality (i.e. different models are created for different parts) and distribution due to the fact that different people may be involved in the process. Ensuring that the distributed feature models do not contain conflicting information is not an easy task. To our knowledge there are currently no tools to provide such support. Inconsistent and conflicting feature models lead to buggy software. Incorrect combinations of features would be made; furthermore possible combinations of features could be missed.

Currently features modeling engineering is also facing a number of other challenges, namely:

1. *There exists no real agreed semantics or notation for feature models:* Many variations to the first feature model notation, FODA [3] exist, such as FORM [4] and FeatuRSEB [5]. Different notations are used to represent the same information. Different extensions are added to FODA such as to include cardinality [6] and feature constraints [7]. For more details we refer the reader to [8].
2. *Lack of a formal common semantics for feature models:* This makes it difficult to exchange feature models in practical applications. As a consequence, tool support for feature model has become fragile, making transformations between feature models a problematic issue. We report the work of [9] on feature models, grammars, and propositional formulas as one possibility of formal representation of feature models.
3. *Dependencies between features are poorly represented:* There may be many relations between the features of one single component; moreover, many interactions, dependencies and conflicts may exist between the features of different components. Many of these dependencies and relations are not easily captured by current feature models.
4. *Feature modeling analysis versus variability analysis mapping is required:* a feature relates to variability via its type. In variability analysis, often the term *variation point* is used to refer to a variable feature. The options of that variable feature are referred to as *variants*. A mapping from feature modeling to the terminology of variation points and variants is required to facilitate a common understanding between different stakeholders regarding variability [1].

In this paper we present FMO, an ontology framework to formally represent and define feature models. We have three main

SAC'09, March 8-12, 2009, Honolulu, Hawaii, U.S.A.

contributions. First, we provide an OWL-based ontology to capture and manage feature models. Secondly, we define a set of rules that insure the logical consistency of the feature model ontology. And thirdly, we provide a mapping between the feature terminology which is most commonly used in feature oriented domain analysis and the variation/variant terminology often used when referring to variability. We show that employing an ontology-based technique to capture feature models semantics may solve some of the problems mentioned earlier in this introduction, and allows for easy manipulation and validation of the models.

The rest of this paper is organized as follows, in section 2 we give a brief overview of the current state of the art of feature models. In section 3, we discuss our feature model ontology, FMO. In section 4, we demonstrate the use of our FMO ontology with an example and show the use of reasoning to deal with the feature model consistency. In section 5, we highlight some related work. Next, in section 6 we summarize and provide an outlook on our future work.

2. FEATURE MODELS

Feature models describe a hierarchical structure of features with exactly one root node, which is further broken up to its corresponding constructing features. Commonly, there are five types of relations possible in a feature model [3] [9]; table 1 shows their graphical notation and meaning.

In addition to the feature types which show feature relations based on their composition, additional constraints between features may exist. Constraints describe how features interact with each other. They control the way features can be put together to form a product.

Table 1. Feature Type relations modified after [10]

And indicates that all subfeatures must be part of any product of the product line	
Alternative indicates that only one subfeature can be selected in any product in the product line.	
Or indicates that one or more subfeatures can be selected as part of any product in the product line.	
Mandatory indicates that this subfeature is required as part of any product in the product line.	
Optional indicates that this subfeature may or may not be part of a product in the product line.	

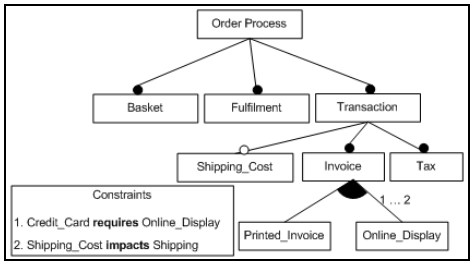

Figure. 1. Order Process Problem, modified after [7]

Figure 1 shows a sample feature model and feature dependencies of the Order Process Problem introduced in [7]. The feature model shows the order process's composing features and how they are composed based on the notation given in table 1. The feature model shows for each feature, its name and type; a feature's contribution to variability is depicted via its type. A feature that contributes to variability is called a *variable feature*. In the above example, *Shipping_Cost* and *Invoice* are variable features. Accompanied with additional feature dependency constraints, a feature model gives information about the features that should be part of a valid software product.

A valid composition of features is called a *configuration* [1] [9]; a valid composition of features results in a *valid product*. A valid product is one that meets all the type restrictions and feature dependencies depicted by the feature model. A *feasible feature model* is one that holds no contradictions or conflicts within the different dependency constraints between features.

3. OUR FEATURE MODELING ONTOLOGY FRAMEWORK

To deal with the challenges mentioned in the introduction, we employ an ontology based approach for the representation of the knowledge contained in feature models. We chose the web ontology language (OWL) [11] to represent our ontology for a number of reasons. First, OWL is the standard (semantic web) language, which will allow making the ontology interoperable among different applications. It will allow easily exchanging different feature models between applications and users. Secondly, OWL has constructs that allows the modeling of both classes and individuals, along with constraints defined on them. This provides a seamless transition from the real-world view of

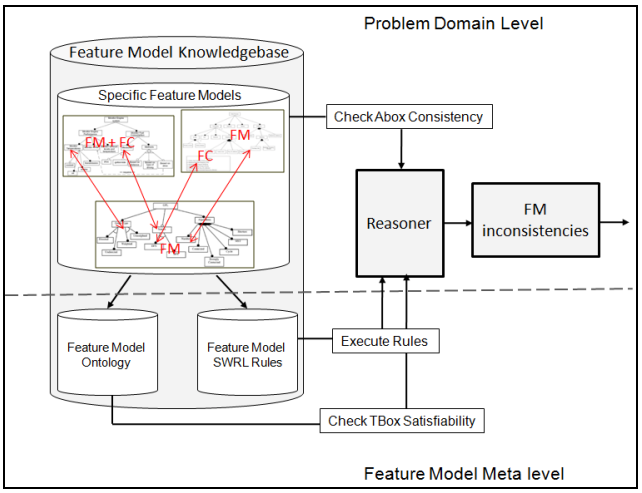

Figure. 2. FMO: feature model ontology framework

the model to the ontological view of the model. Thirdly, OWL (DL) was designed to support DL reasoning on top of the ontology model. This enables using DL reasoners to infer knowledge and using rules implemented in SWRL [12].

Our goal is not to provide a taxonomy for feature models but rather to provide a knowledge representation mechanism for feature models. We do so via providing an OWL representation for the knowledge contained in feature models, and representing all possible feature model semantics. Furthermore, we use SWRL rules [12] to insure the consistency of the feature model and to detect contradicting or conflicting knowledge in the model. Figure

2 shows the main components of the feature modeling ontology framework. The feature model ontology defines the vocabulary to represent the knowledge within feature models. The Feature model logical satisfiability and correctness is captured by a set of rules defining cases of error. The *feature model knowledgebase* holds three types of knowledge namely: the F*eature Model Ontology*, the feature model *SWRL Rules,* and the feature model *instances*. We will discuss each into more detail.

3.1 Feature Model Ontology

An ontology is a conceptualization of a domain. An ontology expresses knowledge of a certain domain of discourse in terms of classes, properties and restrictions. We have chosen the iterative engineering approach described in [13] to model our ontology. We model the feature model constructs as classes. To capture as much as possible all possible semantics represented by feature models, we have conducted a study of the state of the art of feature models to explore the similarities and differences in feature modeling methods. The basic source of our feature model concepts come from FODA representation. We have also added concepts from FORM [4] and FeatuRSEB [5]. We have added semantics for cardinality and feature attributes following the recommendations of [6].

One of the major benefits of having a feature model ontology is that it allows representing features, as well as both feature relations and features constraints in one model. Furthermore we provide feature constraints that allow integrating multiple distributed or segmented feature models. The integration aims at creating one feature model representing the system while detecting conflicts and contradictions. These constraints represent another form of relation between features, and link together different segments of features. We call these constraints *feature to feature constraints* (FTFC). It is the case that features interact together and thus influence the selection of other features within a valid composition. The integration of segmented models must insure the consistency and correctness of the overall model and thus the correctness of configuration decisions based on it.

a) Feature Model Ontology Class Constructs

Following a top down approach to define the key classes within the feature models model, we have defined the following class constructs (shown in figure 3):

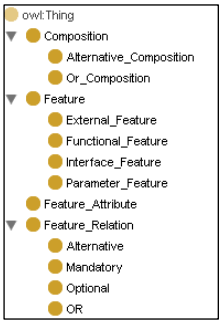

Figure. 3. FMO class hierarchy

- *Feature:* This is the main ontology construct; features can be of type: *external*, *functional*, *interface* or *parameter*.
- *Composition:* represents alternative/or relations in a feature model.
- *Feature Attribute*: defines a variable associated with the feature, the value of the variable is specified during the composition of the product
- *Feature Relation*: represents the possible types: *Mandatory*, *Or*, *Optional*, or *Alternative* for a feature.

b) Feature Model Ontology Properties

The relations between the classes in the ontology are defined via a set of properties; we define the following main properties:

- *Feature_to_Feature_Constraint (FTFC):* property that has Feature class both as domain and range. Providing FTFC as part of the model using formal vocabulary to represent relations between features removes ambiguity. The following are subproperties of the FTFC: *Excludes*, *Extends*, *Impacts*, *Implies*, *Includes*, *Incompatible*, *Requires*, *Uses*, *Same*.

It must be noted that *Incompatible* and *Excludes* are defined as symmetric properties. *Extends*, *requires* and *includes* are defined as transitive properties. Furthermore, for the sake of logical consistency of the feature model, some properties are mutual exclusive such as: (*Requires*, *Excludes*), (*Requires*, *Incompatible*), (*Uses*, *Excludes*), (*Extends*, *Incompatible*).

- *Feature Value constraint:* Represents the value attached to the feature if one exists. It has three disjoint subproperties: *Equal_to*, *Greater_than*, *Less_than*. An example in the order process problem is: *Tax Greater_than 0*
- *Attribute value constraint:* Represents the value attached to the feature attribute, if one exists. It has three disjoint subproperties: *Equal*, *Greater*, *Lesser*. An example in the order process problem: the shipping_cost has an attribute *initial_cost equal 2.5*
- *Compositional properties:* show how the features are composed as parts of other features in the original feature model tree-like notation. Properties which represent this are: C*omposition_of* and its inverse property *Part_of;* they show the belonging of a feature to a composition. *Is_Composed_of* and its inverse *Decomposition_of,* show the simple part of relation between features.

c) *Same/Different features*

In our conceptualization, to explicitly say that two features are the same, we use the *Same* construct, which is defined as a property of *Feature*. *Same* construct is actually the *owl:sameAs* construct in OWL. We use it to state that two features are the same.

3.2 Feature Model Ontology Rules

The need to introduce rules in our ontology is driven by the fact that we need a mechanism to detect the inconsistencies in a single feature model, as well as in a set of integrated feature models. By inconsistency we refer to the term defined by [14], "*A state in which two or more overlapping elements of different software models make assertions about aspects of the system they describe which are not jointly satisfiable*". Some contradictions cannot be represented in OWL DL, such as mutual exclusive properties. Therefore, this type of inconsistency cannot be detected by DL reasoners which try to identify sources of inconsistencies by searching for contradictory facts. Therefore,, we need to define the rules that represent all these invalid states in order to assure that the underlying feature model is logically consistent. By logical consistency we mean that the facts in the ontology are not contradictory or violate rules governing the real case problem. We refer to this type of consistency as *Variability Model Consistency*.

For the sake of inconsistency detection, we have added to our ontology a class named *Inconsistency*, which has one property called *Problem*. *Problem* has Inconsistency as both a domain and range. The situations that cause inconsistencies are defined by a set of SWRL rules. A rule has a body (antecedent) defining an inconsistent situation and a head (consequent) that marks the individuals causing this inconsistent situation. Marking is done by

asserting them to have a *problem* relation between them. The rules assign inconsistent individuals to the Inconsistency class via the problem property. We capture two types of inconsistency problems. The first type of inconsistency emerges from using two properties that are mutually exclusive for the same features. An example rule is: `Requires(?x, ?y) ∧ Excludes(?x, ?y) → problem(?x, ?y)`. The rule captures a problem when a feature *x requires* feature *y* and yet in another part of the model *x excludes y*. A second type of inconsistency is when a two-way direction of using an asymmetric property is detected. An example rule is: `Requires(?x, ?y) ∧ Requires(?y, ?x) → problem(?x, ?y)`. This rule captures a problem when two features *x*, *y* have a two-way *requires* relation, as this is an indication of a cycle in the feature model.

We also use rules to provide information of particular importance for stakeholders interested in variability opportunities. We define SWRL rules that capture all the possibilities to classify features as either a (prospective) *Variation Point* or a (prospective) *Variant*. An example rule is: `Composition(?x) ∧ Composition_Of(?x, ?y) ∧ Decomposition_of(?y, ?z) ∧ type(?y, OR_F) → Variation_Point(?z)`. This rule defines a variation point as a feature that has a child which is part of a composition and has type *OR*. While the rule: `Composition(?x) ∧ Composition_Of(?x, ?y) ∧ Decomposition_of(?y, ?z) ∧ type(?y, OR_F) → Variant(?y)` defines that child as a variant.

3.3 Ontology implementation issues

The Meta model of our feature models ontology (as shown by figure 2 section 3) is fixed and thus there will be no changes in the TBox of the knowledgebase. Rather the changes will occur in the ABox where we constantly add /delete or update existing knowledge while creating and maintaining feature models. Our feature models ontology was implemented using protégé OWL [15] and using Pellet [16] as a DL reasoner.

4. APPLICATION TO AN EXAMPLE

In this section we provide an example that demonstrates how our framework can be used to first model and then integrate segmented feature modes and feature constraint information while detecting model inconsistency. Figure 4 shows an example of feature models and feature dependencies of an Order Process Problem. The order process problem has three consistent segments namely: Order Process Segment, Fulfilment Segment, and Payment Segment. When putting together the three segments there is a clear inconsistency between the features and the constraints of the integrated model (shown in bold in figure 4).

We represent knowledge in figure 4 with the appropriate semantics from the FM ontology. This yields that features are mapped to individuals and the relations between features are defined via their properties. Furthermore constraints between the features represent semantic links for the integration, such as the *uses* relation between *Shipping* and *Shipping_Cost*. We also link Features that are *semantically* the same using the *same* property. Such as *Fulfilment* in figure 4.a which is *semantically* the same feature as *Order_Fulfilment*, in figure 4.b.

Using Pellet for reasoning on the ontology and evaluating the SWRL rules described earlier we obtained the following results: from constraints (4.b.2, 4.c.1) in figure 4 there is a *problem* relation between *Shipping* and *Pay_on_Delivery*. Similarly constraints (4.b.4, 4.a.4) results in a *problem* relation between *Shipping* and *Credit_Card*. The reasoner then infers *Shipping*, *Pay_on_Delivery*, and *Credit_Card* as members of the *Inconsistency* class. Furthermore, by evaluating the SWRL rules, variation points and variants are inferred. *Order_ Fulfilment*, *Order_Payment*, *Invoice, Shipping*, *Payment*, and *Fulfilment* are identified as Variation Points. *Credit_Card*, *Electronic_Delivery*, *Frequent_Flyer, Pay_On_Delivery, Shipping*, *Online_Display*, *Package_Slip*, *Pay_By_Bill* *Package_Tracking_Number*, *Printed_Invoice*, and *Service_Delivery* are identified as Variants.

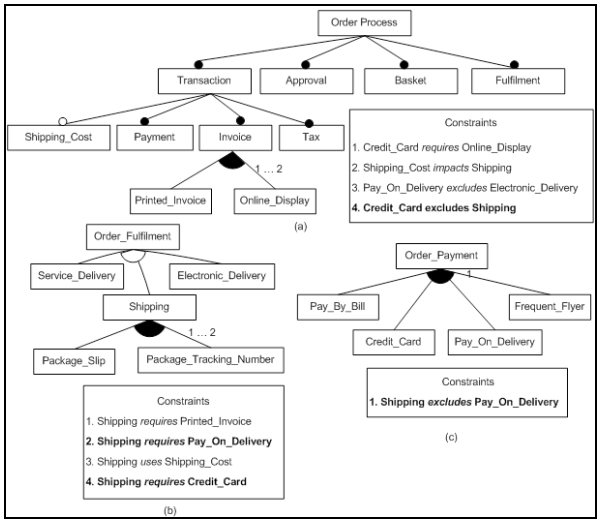

Figure. 4. Order process problem, modified after [7] a) order process segment b) fulfilment segment c) payment segment

The above example illustrates the efficiency of our feature modeling framework. However, we need to apply it to a real world case.

5. RELATED WORK

Related work falls into two main categories, the first being representation of feature models, the second about analyzing feature models to find feasible configurations and detecting dead or problematic features. We first cite work from the former category. In [10], an OWL-based approach was used to represent and verify feature models. OWL constraints are used to model feature relations and constraints defined by the feature model. Given a certain feature composition their approach can detect whether it is valid or not. Furthermore it can also present the axioms that caused it to be invalid. Opposed to our approach, the authors focus more on using OWL and DL reasoning as a technique for verifying the configuration after transforming the problem to an OWL representation. In [17], an OWL ontology for feature modeling is provided. The ontology provides aid in application oriented tailoring. The ontology classifies features based on several categories depending on the underlying business model. The basis is 'action', which represents the business operation. Dependencies between features are identified based on their action requirements. Although very related to our work, the work presented in [17] takes a business prospective to represent feature models which we think limits the use of their technique to business applications which are highly function oriented.

Research in the latter category includes the following work: In [9], the authors attempt to use a Logic based Truth

Maintenance System (LTMS) and Boolean Satisfiability Problem Solver (SAT solver) to propagate constraints. LTMS also provides automatic selections for a possible configuration, and provide justification for automatically selected/deselected features. In [18] feature models are transformed into a Constraint Satisfaction Problem where a Constraint Solver is used to determine the feasible configurations of a feature model. In [19], the authors use Higher Order Logic (HOL) to formulate feature models: Prototype Verification System (PVS), a HOL solver, is then used to find feasible configurations

Although in these techniques configurations (i.e. possible products) are automatically found, debugging in case of a design error is a hard task. Neglecting the fact that a contradiction in the model may be blocking feasible or expected feature combinations is a major drawback for such feature analysis techniques [20].

6. DISCUSSION AND FUTURE WORK

In this paper we have presented a framework for representing, integrating and validating feature models by using OWL and SWRL. We have also applied it to a small example that demonstrates its use. Although OWL was initially proposed for the semantic web, its expressive power and formal semantics made it usable in many other different domains. We provided an ontology for feature models that captures a large category of the semantics in existing feature modeling techniques. We have also added feature-based integration semantics to our ontology enabling integration of distributed feature models. Furthermore we have formulated a list of SWRL rules that define conflicts or inconsistencies in the model as well as rules that infer information regarding variability.

For our future work towards a complete framework to model and manage feature models, we aim to enrich the ontology by applying more examples.

From a usability point of view, it is not only important to detect inconsistencies but it is also very important to be able to explain to users the reason(s) for the inconsistency. Currently, Pellet provides some support for debugging possibilities, but in a very non-user-friendly format. Feature modeling is an accumulative process with many changes; therefore we plan to explore more the possibilities of reasoning with changing ontologies. In our case changes only happens in the Abox which should not complicate the process.

As a second stage, an easy to use user interface to query the features model ontology is required to allow users to query about features and their relations within the ontology.

Our final goal is to provide a tool that will allow collaborative analysis and modeling of feature models while pinpointing conflicts and inconsistencies on the spot allowing for accurate decisions and error free configurations. Furthermore our tool should allow for evolutionary development and act as a repository for feature models.

7. ACKNOWLEDGEMENT

This research is sponsored by ISRIB through the VariBru project.

8. REFERENCES

[1] Bosch,J.: *Design & Use of Software Architectures: Adopting and Evolving a Product-Line Approach*, Addison-Wesley, 2000.

[2] Bosch J.: Software Product Families in Nokia. In: *9th International Conference SPLC 2005* (2005).

[3] Kang, K., Cohen, S., Hess, J., Novak, W., Peterson, A.: Feature-oriented domain analysis (FODA) feasibility study. *Technical Report CMU/SEI-90-TR-021*, *Software Engineering Institute*, Carnegie-Mellon University (1990)

[4] Kang, K., Kim, S., Lee, J., Kim, K., Shin, E., Huh, M.: FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures. In: *J. Annals of Software Engineering. vol. 5,* pp. 143-168 (1998)

[5] Griss, M., Favaro, J., d'Alessandro, M.: Integrating Feature Modeling with the RSEB. In: *Fifth International Conference on Software Reuse*, pages 76–85 (1998)

[6] Czarnecki, K., Kim, C. H. P.: Cardinality-Based Feature Modeling and Constraints: A Progress Report. In: *OOPSLA'05 International Workshop on Software Factories* (2005)

[7] Ye, H.; Liu, H.: Approach to modelling feature variability and dependencies in software product lines. In: *Software, IEE Proceedings -Volume 152, Issue 3*, Page(s): 101 – 109, (2005)

[8] Schobbens, P.-Y., Heymans, P., Trigaux, J.-C., Bontemps, Y.: Generic Semantics of Feature Diagrams. In*: Computer Networks, volume 51, issue 2*, pp. 456-479, 2007

[9] Batory, D.: Feature models, grammars, and propositional formulas. In: *Obbink, H., Pohl, K. (eds.) SPLC 2005. LNCS, vol. 3714* (2005)

[10] Wang, H., Li, Y., Sun, J., Zhang, H., Pan, J.: Verifying Feature Models using OWL. In: *Journal of Web Semantics: Science, Services and Agents on the World Wide Web. Volume 5, Issue 2*, Pages 117-129, Elsevier, June 2007

[11] OWL, http://www.w3.org/TR/owl-features/

[12] Horrocks, I., Patel-Schneider, P. F., Boley, H., Tabet, S., Grosof, B., Dean, M.: SWRL: A Semantic Web Rule Language Combining OWL and RuleML, http://www.w3.org/Submission/SWRL

[13] Noy, N. F., McGuinness, D. L.: Ontology Development 101: A Guide to Creating Your First Ontology. Stanford Knowledge Systems Laboratory *Technical Report KSL-01-05 and Stanford Medical Informatics*, Technical Report SMI-2001-0880 (2001)

[14] Spanoudakis, G., Zisman, A.: Inconsistency management in software engineering: Survey and open research issues. In: *Handbook of Software Engineering and Knowledge Engineering*, 1, pp. 329-380, 2001.

[15] Stanford Protégé OWL, http://protege.stanford.edu/overview/protege-owl.html

[16] Pellet DL Reasoner, http://pellet.owldl.com/

[17] Peng, X., Zhao, W., Xue, Y., Wu, Y.: Ontology-Based Feature Modeling and Application-Oriented Tailoring. In: *ICSR 2006:* 87-100

[18] Benavides, D., Trinidad, P., Ruiz-Cortés, A.: Automated Reasoning on Feature Models. In: *17th Conference on Advanced Information Systems Engineering (CAiSE'05)*

[19] Mikoláš, J., Kiniry, J.: Reasoning about Feature Models in Higher-Order Logic. In*: 11th International Software Product Lines Conference (SPLC 2007)*.

[20] Batory, D., Benavides, D., Ruiz-Cortés, A.: Automated Analyses of Feature Models: Challenges Ahead. In: *Communications of the ACM* (Special Section on Software Product Lines) (2006)