Feature Model Debugging based on Description Logic Reasoning

Mahdi Noorian[1], Alireza Ensan[1], Ebrahim Bagheri[1,2], Harold Boley[3], and Yevgen Biletskiy[1]

University of New Brunswick, Fredericton, Canada[1]
Athabasca University, Edmonton, Canada[2]
National Research Council Canada[3]
m.noorian@unb.ca, alireza.ensan@unb.ca, ebagheri@athabascau.ca, harold.boley@nrc-cnrc.gc.ca, biletski@unb.ca

***Abstract*—Software product line engineering refers to the concept of sharing commonalities and variabilities of a set of software products in a target domain of interest. Feature models are one of the prominent representation formalisms for software product lines. Given the fact that feature models cover all possible applications and products of a target domain, it is possible that the artifacts are not necessarily and always consistent. Therefore, identifying and resolving inconsistencies in feature models is a significant task; especially, due to the fact that a large number of possible products and complex interactions between the software product line features need to be checked. To address these challenges, in this paper, we propose a framework with an automated tool to find and fix the inconsistencies of feature models based on Description Logic (DL) reasoning. The basic idea of our approach is to first transform and represent a feature model using Description Logics. The second step is to identify the possible inconsistencies of the feature model using DL reasoning and then recommend appropriate solutions to a domain analyst for resolving existing inconsistencies.**

I. Introduction

Software Product Line (SPL) engineering is a paradigm that models families of software products with similar characteristics [12]. These characteristics, known as features, can characterize the specifications of a given domain. In fact, SPLs employ features to model all of the possible configurations and applications of a domain. The use of these features that allow for the sharing of commonalities among features of a domain strengthens the capability of SPLs to be used in re-use-based software development [15]. SPLs are usually represented using Feature Models (FM) that can indicate variability and commonality of product lines in a graphical representation. FMs have a tree structure where each feature is a node of the tree and its edges are the possible variabilities of the features.

Within the software development lifecycle of SPL, the process of developing a feature model for a given domain is referred to as the Domain Engineering phase, whereas the process of selecting the desirable features from the designed feature model for developing a new applications is called the Application Engineering phase. In both steps, inconsistent definitions and designs can be encountered that need to be automatically detected and semi-automatically resolved which requires the development and customization of AI techniques specifically for this purpose.

The task of inconsistency identification and resolution within a feature model is the main concern of this paper. Several approaches have been proposed to verify the validity of a feature model and its products; but only very few techniques exist that address the issue of inconsistency resolution within feature models. Furthermore, there is no automated tool available that is able to perform all of the tasks of inconsistency checking, debugging and resolution within feature models; so this fact shows the necessity for the development of a comprehensive automated AI-centric tool for this purpose. To this end, we propose a framework that describes the whole process of FM design and product generation based on Description Logics. It can check the validity of a given model, and all of its products; also it gives resolutions for invalid products or inconsistent models.

In real world applications such as MS Office, millions of features exist in the feature model. To generate new applications from this huge number of features, a comprehensive and reliable method is needed because feature model validation in this case would need too much time to be properly performed by a human analyst. So if a domain analyst is responsible for validating a large product line, he would need an automated tool that would help him/her by suggesting the best configuration and resolution strategies in an efficient amount of time. For this purpose, the strength of DL reasoning can be useful to guide domain analyst within the process of domain verification.

In this work, we propose a formal framework for performing inconsistency checking, validation and resolution based on DL reasoning for SPL. Using DL reasoning techniques which are implemented within Pellet [16] and also based on Reiter's algorithm [13], we can find the solutions for resolving inconsistencies within a feature model. The main contributions of this work can be summarized as follows:

- We show the importance of using Description Logic variants especially OWL-DL in the area of software product lines;
- We propose a comprehensive framework for inconsistency checking by employing the Pellet reasoning engine. Also, we utilize Reiter's algorithm to propose solutions for resolving the identified inconsistencies.

- Based on our framework, we develop a tool called AUFM which supports the design of FMs in both domain and product levels. Hence, it identifies and resolves inconsistencies in the FM and its related products. Indeed, developing such a tool in the area of software product line can be helpful for domain analysts who are dealing with feature model validation and verification.
- We ground the problem of inconsistency detection and resolution on sound Description Logic representation and reasoning mechanisms that would allow us to ensure that the proposed revision solutions are at least syntactically acceptable.

The remainder of this paper is as follows: The next section reviews some preliminaries. Then, our approach for inconsistency resolution in feature models is described in Section III. In Section IV, we support our approach by analyzing a sample feature model. Subsequently, we evaluate the proposed approach with a case study in Section V. After that, in Section VI we discuss related work in this area. Finally, Section VII is devoted to conclusions and some future work.

II. Preliminaries

A. Feature Models

Features are important distinguishing aspects, qualities, or characteristics of a family of systems [11]. They are widely used for depicting the shared structure and behavior of a set of similar systems. To form a product family, all the features of a set of similar/related systems are composed into a feature model. A feature model represents the possible configuration space of all the products of a system product family in terms of its features. Feature models can be represented both formally and graphically; however, the graphical notation depicted through a tree-like structure is more favored due to its visual appeal and easier understanding.

In a FM, features are hierarchically organized by *Structural Constraints* which can be typically classified as: 1) *Mandatory*: a feature must be included in the description of its parent feature; 2) *Optional*: a feature may or may not be included in its parent description given the situation; 3) *Alternative feature group*: one and only one of features from the feature group can be included in the parent description; 4) *Or feature group*: one or more features from a feature group can be included in the description of the parent feature. In some case, the tree structure of feature models falls short at fully representing the complete set of mutual interdependencies of features; thus, additional constraints are often added to feature models and are referred to as *Integrity Constraints*. The two most widely used integrity constraints are: *Includes* - the presence of a given feature (set of features) requires the inclusion of another feature (set of features); and *Excludes* - the presence of a given (set of) feature(s) requires the elimination of another (set of) feature. In the following sections, the term 'constraint' refers to both integrity and structural constraints unless it is specifically mentioned.

Inconsistent Feature Model (IFM): An IFM is a FM that is vulnerable to generate invalid products with regard to structural and integrity constraints [18], [19]. In other words, an IFM violates some constraint (both integrity and structural) simultaneously. As discussed in [18], inconsistency in the FM can happen in two levels including domain and product configuration level when some defined constraints are violated.

B. Description Logics (DL)

Description Logic as a subset of First Order Logic is a knowledge representation formalism that can help to effectively perform reasoning over a knowledge base. A knowledge base modeled using Description Logics could be defined as $\Psi = (\mathcal{T}, \mathcal{A})$, where $\mathcal{T}$ denotes TBox and comprises of a set of general inclusion axioms and $\mathcal{A}$ stands for ABox and comprises of a set of instance assertions. This kind of knowledge representation contains a set of all concept names (C_N), role name (R_N) and individuals (I_N). The semantic of DL-knowledge base is defined by an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, .^{\mathcal{I}})$ where $\Delta^{\mathcal{I}}$ is a non-empty set of individuals and $.^{\mathcal{I}}$ is a function which maps each $C \in C_N$ to $C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, each $R \in R_N$ to $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ and each $a \in I_N$ to an $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$. An interpretation $\mathcal{I}$ satisfies a TBox axiom $C \sqsubseteq D$ iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. An interpretation $\mathcal{I}$ is a model of a TBox $\mathcal{T}$ iff it satisfies all of its axioms. Furthermore, an interpretation $\mathcal{I}$ is a model of knowledge base Ψ if it satisfies every TBox axiom and ABox assertion of Ψ. A concept C is *unsatisfiable* with regards to TBox $\mathcal{T}$ iff $C^{\mathcal{I}} = \varnothing$ for all models $\mathcal{I}$ of $\mathcal{T}$. In addition, a TBox $\mathcal{T}$ is called *incoherent* iff there is an unsatisfiable concept in $\mathcal{T}$. For detailed introduction to description logic, interested readers can refer to [1].

C. DL-knowledge base Diagnosis and Debugging

In the area of knowledge representation and specially the semantic Web, knowledge base quality assurance is a prominent task. In the literature, various model-based diagnosis methods have been proposed for the purpose of developing a consistent model of a DL knowledge base. Schlobach et al [14] have utilized Reiter's algorithm for identifying the underlying reasons of inconsistencies in a DL knowledge base, which is based on generating the *conflict sets* and the related *minimal hitting sets*. A set is called conflict set when it includes such elements that causes incoherencies. A hitting set is further defined based on the concept of a conflict set such that it contains at least one element from the collection of the conflict sets. A hitting set is *minimal* iff it does not have any subset which is a hitting set. In order to implement Reiter's algorithm, the concept of Minimal Unsatisfiability-Preserving Sub-TBoxes (MUPS) is used, which refers to minimal conflict sets in the TBox. MUPS of a TBox for unsatisfiable concept A are the subsets of the TBox where A is satisfiable. MUPS can be used to compute the Minimal Incoherence- Preserving Sub-TBox (MIPS), which explains the incoherence of a TBox [3]. Consider the TBox $\mathcal{T}$ in Table I [14].

The unsatisfiable concepts are A_1, A_3, A_6, A_7, e.g., MUPS for A_3 is $\{ax_3, ax_4, ax_5\}$. Also MIPS for TBox $\mathcal{T}$ is

TABLE I
AN INCOHERENT TBOX $\mathcal{T}$

$ax_1 : A_1 \sqsubseteq \neg A \sqcap A_2 \sqcap A_3$	$ax_2 : A_2 \sqsubseteq A \sqcap A_4$
$ax_3 : A_3 \sqsubseteq A_4 \sqcap A5$	$ax_4 : A_4 \sqsubseteq \forall s.B \sqsubseteq C$
$ax_5 : A_5 \sqsubseteq \exists s.\neg B$	$ax_6 : A_6 \sqsubseteq A_1 \sqcup \exists r.(A_3 \sqsubseteq \neg C \sqcap A_4)$
$ax_7 : A_7 \sqsubseteq \forall s.B \sqcap C$	

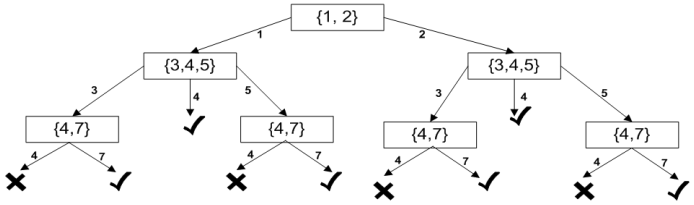

Fig. 1. Reiter's hitting set tree for MIPS in TBox $\mathcal{T}$.

$\{\{ax_1, ax_2\}, \{ax_3, ax_4, ax_5\}, \{ax_4, ax_7\}\}$. Fig.1 represents the Reiter's hitting set tree for MIPS in the above sample incoherent TBox $\mathcal{T}$. Since MIPS is a minimal conflict set, the minimal hitting set tree can be developed based on the selection of at least one element from each of its subsets.

The check mark symbols in the leaves indicate the successful diagnosis path from root of the tree. Thus, we can have the following minimal hitting sets:
$\{\{ax_1, ax_4\}, \{ax_2, ax_4\}, \{ax_1, ax_3, ax_7\}, \{ax_2, ax_3, ax_7\}, \{ax_1, ax_5, ax_7\}, \{ax_2, ax_5, ax_7\}\}$
In order to make the TBox $\mathcal{T}$ coherent, we can simply choose one element form the produced hitting set and omit the axioms of the chosen set. Note that, the members of the hitting set can be selected based on syntactic measures such as different scoring functions on axioms of MIPS.

III. THE PROPOSED APPROACH

In our approach, we focus on the representation of software product line feature models as Description Logic knowledge bases, and consequently use formal reasoning approaches on such a knowledge base to identify and possibly resolve any inconsistencies. In order to support our theoretical framework, we have developed an automated tool from which domain analysts can benefit for the purpose of domain and product consistency validation. First, the domain analysts can begin by designing a feature model using the *AUFM FM Editor*. Next, for the purpose of consistency validation, the feature model which is in SXFM format (SXFM is XML

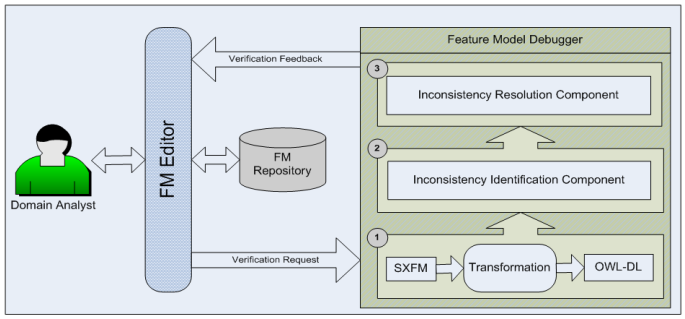

Fig. 2. A semi-automated inconsistency management framework.

TABLE II
DL-KNOWLEDGE BASE STRUCTURE DERIVED FROM A FEATURE MODEL.

Feature relation or integrity constraints	Description logic modeling
$G \sqsubseteq T$,and $GRule \sqsubseteq T$	hasG $\sqsubseteq$ ObjectProperty , T $\sqsubseteq \forall$hasG.G,
$F_1 \sqsubseteq T$,and $F_1 Rule \sqsubseteq T$ ...$F_n \sqsubseteq T$,	and GRule$\equiv \exists$hasG.G.
and $F_n Rule \sqsubseteq T$	Also,
Also,	has $F_i \sqsubseteq$ObjectProperty ,
$G \sqsubseteq \neg F_i$, for $1 \leq i \leq n$,	T $\sqsubseteq \forall$has $F_i.F_i$, and
$F_i \sqsubseteq \neg F_j$, for $1 \leq i, j \leq n$ where $i \neq j$.	$F_i Rule \equiv \exists has F_i.F_i$. for $1 \leq i \leq n$.

standard for representing a FM) can be sent to the *AUFM Feature Model Debugger*. Then, by applying three sequential processes: 1) *Transformation*; 2)*Inconsistency Identification*; and 3) *Inconsistency Resolution*, the domain analysts receive verification feedback. The feedback message contains, what the inconsistent features/constraints are, why those features are/cause inconsistency, and how the existing inconsistencies can be resolved. This information can be used by the domain analysts to correct the inconsistent FM and make it consistent. Note that, feature model verification is an iterative process and it will continue until the domain analysts are satisfied with their design and product configuration. An overview of this process is shown in Fig.2.

According to the framework, *AUFM feature model debugger* consists of three main components. In the following subsections, we will explain each of them in detail.

A. The FM to OWL- DL Transformation Component

In order to resolve inconsistencies in a feature model through Description Logic reasoning, the first step is to convert the feature model representation into some variant of Description Logics. We have chosen OWL-DL for this purpose. We utilize some theoretical concepts discussed in [20] to represent a FM in OWL-DL to benefit from the reasoning and expressiveness provided by it. The conversion is implemented in both the domain and the configuration levels.

The domain conversion phase includes several tasks that need to be performed in order to transform a FM into OWL-DL. The first step of conversion is to assign an OWL class named *Feature Class* for each feature in the domain; and all *Feature Classes* are considered mutually disjointed. The second step is defining a *Rule Class* for each *Feature Class*. Such *Rule Class* is related to its corresponding *Feature Class* with *necessary* and *sufficient* condition which is restricted by existential restriction. For example, assume that F is a *Feature Class*. So, corresponding $FRule$ class is defined as $FRule \equiv \exists hasF.F$. All relations that an individual *Feature Class* might have with other *Feature Classes* are embedded in its corresponding *Rule Class*. The aforementioned possible relations can be defined as structural or integrity constraints. In other words, each *Rule Class* represents all relations that the related *Feature Class* has with its children in the necessary condition definition; this is the main reason that a *Rule Class* is defined. Consequently, for a root feature G and its children $F_1, F_2, ..., F_n$, the associated DL-knowledge base has the general structure represented in Table II.

TABLE III
LOGIC REPRESENTATION FOR STRUCTURAL AND INTEGRITY CONSTRAINTS IN FMS

Structural or Integrity Constraints	Description Logic Modeling
Mandatory	*GRule*$\equiv \exists\ hasF_1.F_1$... *GRule*$\equiv \exists hasF_n.F_n$
Optional	Does not impose any constraints
Or	*GRule*$\sqsubseteq \sqcup(\exists hasF_i.F_i)\ for 1 \leq i \leq n$
Alternative	*GRule*$\sqsubseteq \sqcup(\exists hasF_i.F_i)\ for 1 \leq i \leq n$
	GRule$\sqsubseteq \neg \sqcup\ (\exists hasF_i.F_i \sqcap \exists hasF_j.F_j)\ \ for\ 1 \leq i \leq j \leq n$
Include(Require)	*GRule*$\equiv \exists hasF_1.F_1$...*GRule*$\equiv \exists hasF_n.F_n$
Exclude	*GRule*$\equiv \neg(\exists hasF_1.F_1)$...*GRule* $\equiv \neg(\exists hasF_n.F_n)$

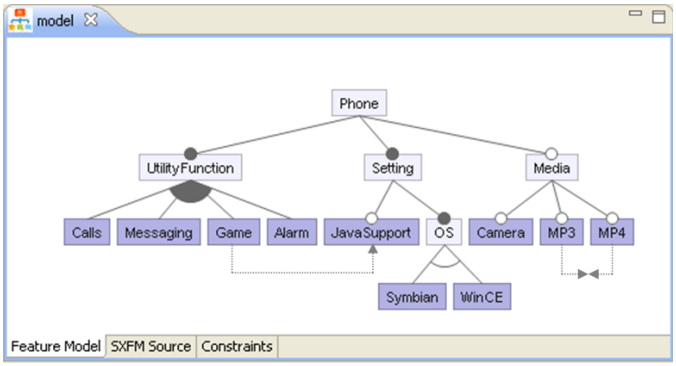

Fig. 3. The AUFM feature model editor.

As discussed before, there are two types of constraints including structural and integrity constraints in a FM. For a parent feature G and its children $F_1, F_2,..., F_n$, DL representation for all kinds of constraints are summarized in Table III. It is worth noting that, *Optional* relations do not impose any extra constraints to the feature model and DL-knowledge base, so we are not concerned with *Optional* relationships in the DL-knowledge base model.

The meaning of parent feature and its children is similar in an *Include/Excludes* constraint which means that if feature G is selected for one product, it is necessary that features $F_1, F_2,..., F_n$ are included/excluded in that product.

The products conversion phase is a straightforward task in light of the domain conversion step. In order to model a product of a FM using DL, we can add a class called $ProductClass$ and consider which features exist in that product and which do not. Mathematically speaking, for a FM with root feature G and set of features $F_1, F_2,..., F_n$, a product that includes features $F_1, F_2,..., F_i$ and does not contain features $F_{(i+1)}$,$F_{(i+2)}$,...,F_n is defined as follows:
$Product \sqsubseteq GRule$
$Product \equiv (\sqcap(\exists hasF_j.F_j\ \ for 1 \leq j \leq i \leq n) \sqcap (\neg\exists hasF_k.F_k for i < k \leq n))$

B. The Inconsistency Identification Component

Basically, this component is responsible for finding the inconsistencies in the FM and its related products. For a given feature model, the source of inconsistencies can be twofold: *structural constraint violation* and *integrity constraint violation*. After transforming feature model to OWL-DL, we can perform DL reasoning with any standard DL reasoner to pinpoint the inconsistent parts of the feature model. Several DL reasoners are already available such as FaCT++ [17], RACER [9], and Pellet [16] that implement Tableau based reasoning [1]. Pellet has been seamlessly plugged into our designed tool and we can take advantage of its facilities to implement Reiter's algorithm for inconsistency resolution, which will be discussed in the next subsection. Thus, we have employed Pellet as an integrated reasoning engine in AUFM and are using it for the purpose of inconsistency checking.

C. Inconsistency Resolution Component

The inconsistency resolution component is an important part of the *AUFM feature model debugger*. The aim of this component is to find the minimum corrections needed to make an inconsistent FM into a consistent one. In other words, we are looking for a minimal subset of axioms in TBox (here TBox refers to the converted FM into DL) that need to be repaired or removed to render a correct FM and thus make it consistent again. Note that here the TBox contains all *Feature Classes*, *Rule Classes*, and their correlated properties which are discussed in Tables II and III. Also, at the product configuration time, the TBox will be extended by adding *Product Classes*. For any incoherent TBox (inconsistent FM), the reasoning engine returns the list of unsatisfiable axioms (features and constraints). At this point, AUFM automatically performs debugging and exposes the main reasons of inconsistency and provides reasonable solutions for resolving them. All these tasks are carried out via a user friendly interface.

We adopt the theory of debugging and diagnosis from [14] within our work. Based on this, for a given inconsistent FM, AUFM finds the minimal conflict set for corresponding TBox. Note that, here conflict set can be defined as a MIPS set. Afterwards, Reiter's hitting set algorithm is used to find the minimal hitting set. Indeed, MIPS set expresses why the feature model is inconsistent according to errors that are found by the reasoning engine. In addition, the hitting set represents how these errors can be resolved. AUFM supports both services inherently and domain analyst can benefit from them automatically.

IV. A SAMPLE PROCESS DEPICTION

To demonstrate the process of our framework, in this section we illustrate the resolution of a sample inconsistent feature model. According to our proposed approach, domain analysts should go through the following steps to validate any target feature model and its related products.

1. Design a feature model using the AUFM editor: We have provided the AUFM feature model editor which can

TABLE IV
RESULTS FROM INTERACTIVE DEBUGGING PROCESS

Steps	Inconsistencies	Recommended solutions	More Inconsistency?
1	Exclude violation	*MP3Rule*$\equiv \exists$*hasMP3.MP3*	Yes
2	Alternative violation	*OSRule*$\equiv \exists$*hasOS.OS*	Yes
3	Mandatory violation	*PhoneRule* $\subseteq \exists$*hasUtilityFunction.UtilityFunction*	No

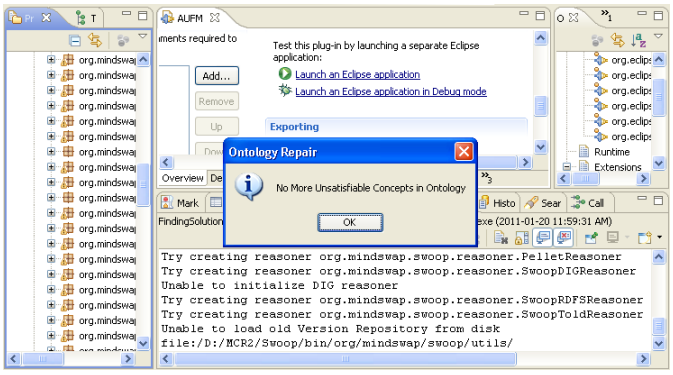

Fig. 4. Consistency checking result for Phone feature domain.

be used as a plug-in in the Eclipse platform. As depicted in Fig.3, the *Phone* domain is graphically designed with 16 features which is a simple part of a real phone domain. The corresponding SXFM file will be generated automatically by AUFM. We will use this feature model as an example for presenting our framework. The SXFM file for this can be accessed from *http://falcon.unb.ca/*$\sim j41z9$*/files.html*.

2. Transformation from SXFM to OWL-DL: In this stage, we need to transform the *phone* feature model into its DL representation. For this matter, the produced SXFM file will be used and fed as an input to the *transformation* phase. The result of transformation component is an OWL-DL knowledge base that represents the input feature model. The generated OWL file for *phone* feature model is available at *http://falcon.unb.ca/*$\sim j41z9$*/files.html*.

3. Consistency checking and inconsistency resolution: Consistency checking can be performed in two levels, feature domain and configuration levels. Here first, we perform consistency evaluation for *Phone* feature model in the feature domain level. Furthermore, by configuring a sample product we will show how AUFM can identify probable inconsistencies. In both levels, Pellet has been used as plug-in in AUFM to perform tableau based reasoning to check consistency in the TBox which is generated from the *phone* feature model.

1) Consistency checking in feature domain level*:* Fig.4 represents the consistency checking result for the *Phone* feature model at the feature domain level. As illustrated, the AUFM reasoning engine does not find any unsatisfiable concepts. Thus, *Phone* feature model is consistent in terms of feature model structural and integrity constraints.

2) Consistency checking and inconsistency resolution in configuration level*:* Before starting consistency checking in this level, we need to configure a product. We configured an inconsistent product, *P*, which violates one *Mandatory*, *Alternative*, and *Exclude* constraints. Consider*P* as follows:

P= {*Phone, Setting, Calls, Messaging, OS, Symbian, WinCE, Media, MP3, MP4, Camera*}.

The OWL file for product *P* can be accessed from *http://falcon.unb.ca/*$\sim j41z9$*/files.html*. According to the structural and integrity constraints of the *phone* feature model, product *P* cannot have both of the *Symbian* and *WinCE* features at the same time. Also, *MP3* and *MP4* are related with *Exclude* constraint. Thus, these two features cannot be included in product *P* simultaneously. In addition, *UtilityFunction* is defined as a *Mandatory* feature and it must be included in product *P*, while it is not.

AUFM performs consistency validation which is an iterative process. In each iteration, the system detects any possible inconsistencies and provides its corresponding solution and then sends a report back to the domain analysts. Then domain analysts are able to look at and resolve inconsistency (based on the recommendations from AUFM) and send the edited FM to *feature model debugger* for further inconsistency identification. This process will continue until there no more inconsistency could be found in the designed FM.

Product *P* has three inconsistencies. Thus, the consistent product can be gained after three steps of product debugging. In the first step, AUFM finds that the domain analysts have violated the *Exclude* constraint, which connects *MP3* and *MP4* features. Afterwards, the solution (which is *MP3Rule*$\equiv \exists$*has MP3.MP3*) for resolving this error would be recommended by AUFM. This problematic axiom is found based on Reiter's algorithm. As discussed in Section III, to implement the Reiter's algorithm, hitting set trees should be drawn and traversing in the edge weighted tree can determine the valid solutions for resolving inconsistencies. The drawn hitting set tree for resolving the first inconsistency which is *Exclude* violation is available at *http://falcon.unb.ca/*$\sim j41z9$*/files.html*. In order to resolve this inconsistency, domain analysts need to remove this axiom from the knowledge base. With regards to the meaning of the feature model, *MP3* should be removed from the product *P*. In other words, domain analysts interpret the meaning of the proposed solutions based on the semantics of the feature model. In product *P*, *MP3* and *MP4* are related with an *Exclude* constraint; in order to resolve this inconsistency, domain analyst needs to omit one of them from the configured product *P*. After domain analysts resolve the first error, they may ask whether there is more inconsistency or not. This process will continue until the second and third inconsistencies which are *Alternative* and *Exclude* violation are resolved. At the end of this interactive debugging process, product *P* is turned into a

TABLE V
THE CASE STUDY RESULTS (TIME IS BASED ON MINUTES)

Student	Number of feature	Number of structural constraint	Number of integrity constraint	Number of inconsistency	Elapsed time for debugging
st1	32	12	4	3	2.21
st2	27	10	6	7	5.10
st3	35	15	6	5	3.66
st4	30	11	3	4	2.45

consistent model. Table IV represents the inconsistencies and corresponding recommended solutions which are provided by AUFM.

V. EXPERIMENTAL EVALUATION

In this section, we report on a case study to investigate the suitability of the proposed approach in terms of its usability. Here the proposed framework is evaluated to find out whether our approach can be practically helpful for domain analysts in the case of feature model consistency checking and debugging. In the following, we first describe the initialization process for usability evaluation and then we present the results and provide discussion in this regards.

Initialization process: In this case study four participants were invited to comprehensively model a *Smart Phone* domain and design corresponding feature models using our tool, AUFM. Two of these participants were undergraduate students which had little background about domain modeling. The other two participants were graduate students which their research areas were software product line engineering.

We recorded each participant's activities based on some metric criteria such as number of features, integrity constraints, structural constraints, and inconsistencies in the *Smart Phone* domain. Also, we recorded the elapsed time of the iterative debugging process which mainly includes inconsistency identification and inconsistency resolution process.

Result and discussion: The case study observations are summarized in Table V.

As seen in Table V, students subjectively design *Smart Phone* feature models with average 31 features and 17 constraints. Also, the number of inconsistencies that has been discovered by AUFM varies between 3 to 7. In fact, the source of this variation for inconsistency is twofold. First, each individual participant has various levels of knowledge about domain modeling and second, the number of features and constraints that have been used by the students could affect the number of inconsistencies. That is, the more features and constraints, the more the designed model is susceptible to error.

The last column of Table V represents the elapsed time for the debugging process. The debugging process begins once the students send their initial inconsistent feature models to *AUFM feature model debugger*. The process continues until all the inconsistencies are resolved. In this case study, we have observed that the debugging process is varying almost between 2 to 5 minutes. In fact, we noticed that for the experienced users, the elapsed times are fairly acceptable for the feature model with average 31 features. Although, by considering the fact that st2 is the least experienced in comparison with the others; 5 minute of debugging could be considered acceptable.

After evaluating the results and discussing with students, we conclude that the AUFM tool is a user-friendly tool such that students with different levels of knowledge could easily design whatever they have in their minds. AUFM provides the means for students to interactively identify the inconsistencies of their designs and resolve them based on the recommended solutions. However, some of the students pointed that the recommended solution is at times ambiguous and it takes time to understand them in some cases. The main reason for this problem is that the recommended solutions are based on the DL language. They suggest that, it would be more efficient, in terms of time, if the users have this solution according to the feature modeling language rather than DL. Thus, they would be able to just concentrate on their design and directly correct their designed domain.

VI. RELATED WORK

Since the advent of feature models in the early nineties [10], a lot of work has been done in this area. Researchers have devoted their efforts to various stages of software product line engineering including works such as SAT-based feature configuration [21], AHP-based model ranking and preference elicitation [2] and machine learning-based external attributes quality prediction [4].

Some attempts have also recently been made to to find inconsistencies in feature models. Some of them have concentrated on the conversion of feature models to logical representations to benefit from the reasoning capabilities that exist in these domains [19], [5], [6], [8]. In [5], [6], feature models have been transformed into propositional logic and non-functional domain attributes have been modeled with fuzzy variables; and an semi-automated method uses propositional logic feature selection approach to check the satisfaction of domain's constraints. The characteristics of First Order Logics (FOL) and constraint programming have been applied to the analysis of feature models in [7]. In [8], Description Logics has been used to build an ontology for features and some external services. Later on, using ontology matching algorithms, the proper services can be bound to corresponding features. Then using a DL-based reasoning engine the consistency validation is applied on feature models. Most work that deal with inconsistent feature models have focused on validity checking or finding inconsistencies in feature models. There is the lack of approaches to propose solutions to resolve inconsistencies in feature models. The most important approach that suggests solutions for inconsistency resolution in feature models is [19]. Wang et al in [19] have extended an incremental algorithm,

called SkyBlue. They propose a dynamic approach to prioritize constraints in a feedback control system that adjusts itself based the opinions of the domain analysts. In contrast to [19], we have transformed feature models into Description Logic (OWL-DL) to utilize the strength of DL reasoning approaches for solving inconsistency in feature models. Furthermore, by employing Reiter's algorithm the proper solution is provided to the domain analysts.

VII. Conclusion and Future Work

In this work, we strive to design and implement a framework to identify and resolve inconsistencies in feature models based on DL reasoning. In our approach, feature models can be automatically converted into OWL-DL in order to perform reasoning and check the consistency of the models. We have seamlessly integrated Pellet as a plug-in into our AUFM tool to discover inconsistencies in the feature model. Based on the proposed solutions, domain analysts can simply remove or keep the inconsistent features. Our preliminary evaluations demonstrate that our framework can help a domain analyst model and correct the desired domain dynamically.

One of the main contributions of our work is that we provide a correspondence between feature model representations and Description Logic knowledge bases and benefit from formal DL reasoning for identifying and resolving inconsistencies in feature models. Given the clear semantics of DL revisions, we have been able to guarantee meaningful revisions in feature models through DL based semi-automatic inconsistency resolution algorithms. For future work, we believe that our tool support, AUFM, needs more evaluations in terms of scalability testing. It is required to examine what would be the performance of AUFM and basically DL based reasoning methods for handling a feature model with large number of features. In addition, based on the feedback we obtained from the case study participants, we find that the resolution process would be more efficient if AUFM provides recommended solutions in FM language rather than in pure DL. So, we also need to think about this matter and improve the usability of AUFM at this point.

Acknowledgments

The authors would like to acknowledge the work of Mostafa Karami Bekr and his contributions to AUFM.

References

[1] Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter Patel-Schneider. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2003.

[2] Ebrahim Bagheri, Mohsen Asadi, Dragan Gasevic, and Samaneh Soltani. Stratified analytic hierarchy process: Prioritization and selection of software features. In *The 14th International Software Product Line Conference*. Springer, 2010.

[3] Ebrahim Bagheri and Faezeh Ensan. Evidential reasoning for the treatment of incoherent terminologies. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC '10, pages 1381–1387, New York, NY, USA, 2010. ACM.

[4] Ebrahim Bagheri and Dragan Gasevic. Assessing the maintainability of software product line feature models using structural metrics. In *Software Quality Journal*. Springer, 2010.

[5] Ebrahim Bagheri, Tommaso Di Noia, Dragan Gasevic, and Azzurra Ragone. Formalizing interactive staged feature model configuration. In *Journal of Software Maintenance and Evolution: Research and Practice*. Wiley, 2010.

[6] Ebrahim Bagheri, Tommaso Di Noia, Azzurra Ragone, and Dragan Gasevic. Configuring software product line feature models based on stakeholders' soft and hard requirements. In *The 14th International Software Product Line Conference*. Springer, 2010.

[7] D. Benavides, Ruiz A. Cortés, and P. Trinidad. Using Constraint Programming to Reason on Feature Models. In *The Seventeenth International Conference on Software Engineering and Knowledge Engineering (SEKE'05)*, July 2005.

[8] Marko Boskovic, Ebrahim Bagheri, Dragan Gasevic, Bardia Mohabbati, Nima Kaviani, and Marek Hatala. Automated staged configuration with semantic web technologies. In *International Journal of Software Engineering and Knowledge Engineering*, volume 20, pages 459–484. World Scientific, 2010.

[9] Volker Haarslev and Ralf Möller. Racer system description. In *Proceedings of the First International Joint Conference on Automated Reasoning*, IJCAR '01, pages 701–706, London, UK, 2001. Springer-Verlag.

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

[11] Kyo C. Kang, Jaejoon Lee, and Patrick Donohoe. Feature-oriented product line engineering. *IEEE Software*, 19:58–65, 2002.

[12] Klaus Pohl, Günter Böckle, and Frank J. van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[13] R Reiter. A theory of diagnosis from first principles. *Artif. Intell.*, 32:57–95, April 1987.

[14] Stefan Schlobach, Zhisheng Huang, Ronald Cornet, and Frank van Harmelen. Debugging incoherent terminologies. *Journal of Automated Reasoning*, 39:317–349, 2007. 10.1007/s10817-007-9076-z.

[15] Richard W. Selby. Enabling reuse-based software development of large-scale systems. *IEEE Transactions on Software Engineering*, 31:495–510, 2005.

[16] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical owl-dl reasoner. *Web Semantics: Science, Services and Agents on the World Wide Web*, 5(2):51 – 53, 2007. Software Engineering and the Semantic Web.

[17] D. Tsarkov and I. Horrocks. FaCT++ description logic reasoner: System description. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 4130 LNAI:292–297, 2006.

[18] T. von der Maen and H. Lichter. Deficiencies in feature models. In *Proceedings of SPLC'04 Workshop on Software Variability Management for Product Derivation*, 2004.

[19] Bo Wang, Yingfei Xiong, Zhenjiang Hu, Haiyan Zhao, Wei Zhang, and Hong Mei. A dynamic-priority based approach to fixing inconsistent feature models. In *Model Driven Engineering Languages and Systems*, volume 6394, pages 181–195. Springer, 2010.

[20] Hai H. Wang, Yuan Fang Li, Jing Sun, Hongyu Zhang, and Jeff Pan. Verifying feature models using owl. *Web Semant.*, 5:117–129, June 2007.

[21] Jules White, Brian Dougherty, Douglas C. Schmidt, and David Benavides. Automated reasoning for multi-step feature model configuration problems. In *SPLC*, pages 11–20, 2009.