Consistency maintenance for evolving feature models

Jianmei Guo [a,*], Yinglin Wang [a], Pablo Trinidad [b], David Benavides [b]

[a] Department of Computer Science and Engineering, Shanghai Jiao Tong University, 800 Dong Chuan Road, Minhang, Shanghai 200240, China [b] Department of Languages and Computer Systems, University of Seville, Av. Reina Mercedes s/n, 41012 Seville, Spain

Keywords:

Software product lines
Feature models Evolution
Consistency maintenance
Ontology
Semantics

A B S T R A C T

Software product line (SPL) techniques handle the construction of customized systems. One of the most common representations of the decisions a customer can make in SPLs is feature models (FMs). An FM represents the relationships among common and variable features in an SPL. Features are a representation of the characteristics in a system that are relevant to customers.

FMs are subject to change since the set of features and their relationships can change along an SPL lifecycle. Due to this evolution, the consistency of FMs may be compromised. There exist some approaches to detect and explain inconsistencies in FMs, however this process can take a long time for large FMs.

In this paper we present a complementary approach to dealing with inconsistencies in FM evolution scenarios that improves the performance for existing approaches reducing the impact of change to the smallest part of an FM that changes. To achieve our goal, we formalize FMs from an ontological perspective and define constraints that must be satisfied in FMs to be consistent. We define a set of primitive operations that modify FMs and which are responsible for the FM evolution, analyzing their impact on the FM consistency. We propose a set of predefined strategies to keep the consistency for error-prone operations.

As a proof-of-concept we present the results of our experiments, where we check for the effectiveness and efficiency of our approach in FMs with thousands of features. Although our approach is limited by the kinds of consistency constraints and the primitive operations we define, the experiments present a significant improvement in performance results in those cases where they are applicable.

1. Introduction

Software product line (*SPL*) engineering has emerged as one of the most promising software development paradigms for reducing development costs, enhancing quality, and shortening time to market (Clements & Northrop, 2001; Pohl, Bockle, & van der Linden, 2005; Sugumaran, Park, & Kang, 2006). An SPL is "a set of software-intensive systems that share a common, managed set of *features* satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way" (Clements & Northrop, 2001). Features are essential abstractions of product characteristics relevant to customers and are typically increments in product functionality (Benavides, Segura, & Cortes, 2010; Kang, Lee, & Donohoe, 2002). Every *product* or system of an SPL is represented by a unique combination of features. All the products in an SPL are usually captured in *feature models (FMs)* which describe the commonalities and variabilities of systems in terms of features (Kang, Cohen, Hess, Novak, & Peterson, 1990; Kang et al., 2002). An FM is a tree-like structure that contains the relationships among features in a hierarchical manner. Relationships can be of different kinds to remark which are the choices a customer can make to build a customized product.

As any other software systems, SPLs are subject to changes and evolution along their lifecycle. Those changes can affect FMs (Sugumaran et al., 2006). Even small changes to an FM could unintentionally break its consistency (Guo & Wang, 2010; Thum, Batory, & Kastner, 2009). Here, the *consistency* of an FM means that it remains well-formed (*syntactic consistency*) and it defines at least a valid product (*semantic consistency*). For example, the removal of a single feature from an FM, although could be valid from the point of view of FM syntax, could invalidate other related features, which must also be removed to make the resulting FM consistent. Inconsistency usually comes from contradictory constraints which impede producing any valid product (von der MaBen & Lichter, 2004). Consistent FMs are needed to any further steps in SPL engineering such as verifying product derivation (Lutz, 2008) or checking for the consistency of product requirements (Lauenroth & Pohl, 2008). Therefore, guaranteeing the FM consistency is a mandatory task in SPL development.

Many approaches have been proposed to automate the detection of inconsistencies in FMs (Benavides et al., 2010). They mostly

* Corresponding author. Tel.: +86 21 34204415; fax: +86 21 34204728.
E-mail addresses: guojianmei@sjtu.edu.cn (J. Guo), ylwang@sjtu.edu.cn (Y. Wang), ptrinidad@us.es (P. Trinidad), benavides@us.es (D. Benavides).

use SAT (Batory, 2005; Thum et al., 2009), BDD (Czarnecki & Wasowski, 2007), or CSP solvers (Benavides, Martin-Arroyo, & Cortes, 2005; Trinidad, Benavides, Duran, Ruiz-Cortes, & Toro, 2008) to automate the checking process. However, these approaches suffer from an NP-hard problem of feature combinatorics and take a long time to perform with large FMs (Batory, Benavides, & Ruiz-Cortes, 2006). Reports from industry have shown that practical FMs could have hundreds or thousands of features (Loesch & Ploedereder, 2007; Steger et al., 2004), so existing approaches can take hours or days to detect inconsistencies.

Moreover the information obtained from these detection mechanisms hardly assists domain analysts to resolve inconsistencies in FMs. Existing approaches delegate the reparation of inconsistent FMs to domain analysts who must use their experience to find the best way to repair those FMs. The impact of performance and manual reparation gets worse since FMs frequently change during their evolution processes and the consistency of the resulting FMs has to be checked after every change.

This paper approaches the problem of consistency maintenance in FMs focusing on the changes since last version of an FM rather than checking the overall consistency of the resulting FM. We assume that the initial FM is consistent and study if a requested change affects the consistency or not. In case an inconsistency is detected, a set of additional operations are executed to restore the consistency of the FM. For example, the removal of a single feature X from an FM initially only affects those features that are directly connected to it. To restore the consistency, the surrounding relations of feature X are removed. What if the removal of a relationship generates a new inconsistency? In the worst case, the operation derivation could propagate to the whole FM; but in most cases it only affects a limited range, that is where the main improvement in performance comes.

Our approach relies on ontology, which is a formal and explicit specification of a shared conceptualization of a domain of interest (Gruber, 1993). In our case we formalize FMs in Section 3, defining the primitive elements of FMs and the syntactical and semantic consistency constraints as the well-formedness rules of FMs. From this formalization, in Section 4 we obtain a set of primitive operations (Guo & Wang, 2010) which can represent any modification of an FM.

In Section 5 we apply and extend techniques from ontology evolution (Haase & Stojanovic, 2005; Stojanovic, 2004) to propose a systematical approach to consistency maintenance for evolving FMs. A dependency matrix, indicating the *cause and effect* relationships between changes, is built for supporting the derivation of additional operations from the requested change. Then we analyze the possible evolution strategies for all the primitive operations on FMs and propose a sequence of interdependent operations derived from the requested change to produce a unique consistent FM.

To demonstrate the realization of our approach, Section 6 presents the implementation of our approach based on *FeatureIDE*.[1] Section 7 evaluates our approach by experiments on randomly generated FMs with thousands of features. Section 9 briefly analyses the pros and cons of our approach and presents some future extensions of our work.

In order to introduce the readers in the context of our work, we complement our work with a brief definition of FMs in Section 2 and a discussion about the related work in Section 8.

2. Feature models background

In 1990, Kang et al. (1990) first proposed the original FMs (a.k.a. FODA FMs). An FM is organized hierarchically and is graphically depicted as an AND-OR *feature diagram* (Kang et al., 1990). *Cross-tree constraints* are used to represent non-hierarchical composition rules comprising mutual dependency (*requires*) and mutual exclusion (*excludes*) relationships (Kang et al., 1990). Czarnecki, Helsen, and Eisenecker (2005) proposed cardinality-based FMs where *cardinalities* (a.k.a. multiplicities) were introduced. Batory (2005) and Thum et al. (2009) distinguished among *terminal* (or *concrete*) and *non-terminal* (or *compound* or *abstract*) features.

By integrating former definitions of FMs (Batory, 2005; Czarnecki et al., 2005; Kang et al., 1990; Thum et al., 2009), we adopt the notation as shown in Fig. 1, which is a partial FM for the Home Integration Systems (HIS) SPL inspired from (Benavides et al., 2005; Kang et al., 2002). An FM is a tree of features. Every node in the tree has one parent except the *root feature* ('r: HIS'). A *terminal* feature (e.g., '*f*4') is a leaf and a *non-terminal* feature (e.g., '*f*1') is an interior node of a feature diagram (Batory, 2005; Thum et al., 2009). Connections between a feature and its group of children are classified as *And-* (e.g., '*f*1', '*f*2', and '*f*3'), *Or-* (e.g., '*f*10' and '*f*11'), and *Alternative-*groups (e.g., '*f*12', '*f*13', and '*f*14'). The members of And-groups can be either *mandatory* (e.g. '*f*1') or *optional* (e.g. '*f*3'). Or-groups and Alternative-groups have their own *cardinalities* (Czarnecki & Wasowski, 2007). Cross-tree constraints comprise *requires* and *excludes* relationships (Kang et al., 1990), e.g., '*f*4 requires *f*7'.

Table 1 summarizes the semantics of FMs in propositional formulas. P represents a non-terminal feature and $C_1,\ldots,C_n$ are its child features. If the child features forms an And-group, then $M \subseteq \{1,\ldots,n\}$ denotes the mandatory features by their index. If a feature is selected, so too is its parent. If the parent is selected, all of its mandatory children of an And-group are selected; in Or-groups, at least one child must be selected, and in Alternative-groups, exactly one child is selected. Using the rules given in Table 1, an FM can be easily translated into a propositional formula with a variable for each feature.

3. Ontology-based formalization and consistency constraints

3.1. An ontology-based formalization of FMs

The representational primitives defined in ontology (Gruber, 2008) are typically *concepts* (*classes*) and *properties*. Each property must have at least one *domain* concept, while its *range* may either be a literal (*attributes*), or a set of at least one concept (*relations*). The definitions of the representational primitives include information about their meaning and constraints on their logically consistent application, which makes ontology work at the semantic level. Based on the ontology structure and the application context of FMs, we formalize FMs as follows.

Definition 1. An FM is defined as a 5-tuple:

$$FM = (C, R, A, Domain, Range)$$

where:

- $C\ (\neq\emptyset)$ is the set of concepts in the FM. $C = F \cup FG$. $F\ (\neq\emptyset)$ is the set of features in the FM. $FG\ (\neq\emptyset)$ is the set of feature groups. $FG = FG_{AN} \cup FG_{AL} \cup FG_{OR}$. FG_{AN} is the set of And-groups; fg_{AN} is an element of FG_{AN}, i.e., an And-group. FG_{AL} is the set of Alternative-groups. FG_{OR} is the set of Or-groups. (1) $F = \{root\} \cup NF \cup TF$. *root* is the root feature. $NF\ (\neq\emptyset)$ is the set of non-terminal features. $TF\ (\neq\emptyset)$ is the set of terminal features. (2) $F = \{root\} \cup F_{AN} \cup F_{AL} \cup F_{OR}$. F_{AN}, F_{AL}, and F_{OR} denote the set of features in all And-groups, in all Alternative-groups, and in all Or-groups respectively. Take the FM shown in Fig. 1 for example, *root* = "*r*"; "*f*1" $\in NF$; "*f*4" $\in TF$; $fg_{AN}1 = \{$"*f*1", "*f*2", "*f*3"$\}$, $fg_{AN}1 \in FG_{AN}$; $FG_{AL} = \{$"$fg_{AL}1$"$\}$, $fg_{AL}1 = \{$"*f*12", "*f*13", "*f*14"$\}$. $F_{OR} = fg_{OR}1 = \{$"*f*10", "*f*11"$\}$.

[1] http://wwwiti.cs.uni-magdeburg.de/iti_db/research/featureide.

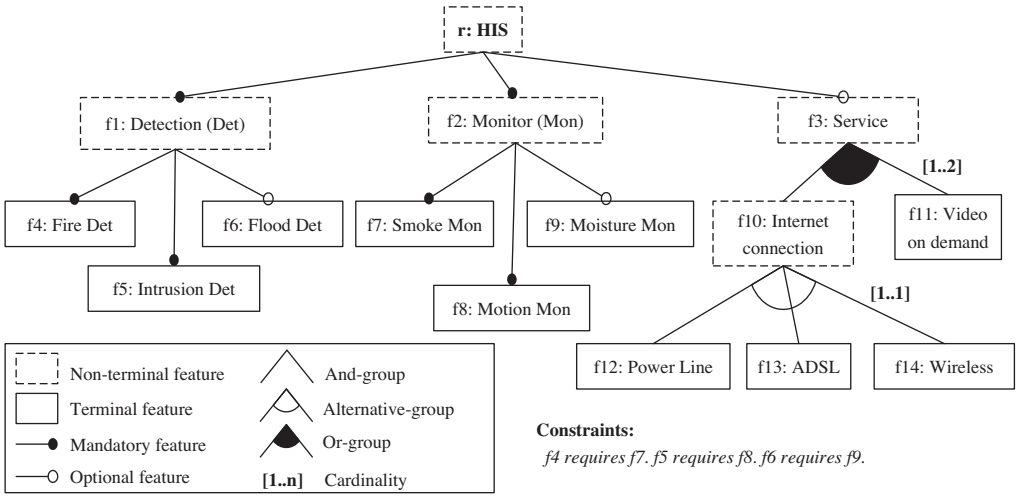

Fig. 1. A partial FM for the HIS SPL.

Table 1
FM Semantics in propositional formulas.

FM primitives	Semantics
Optional child	$C_1 \rightarrow P$
Mandatory child	$C_1 \leftrightarrow P$
And-group	$(P \rightarrow \bigwedge_{i \in M} C_i) \wedge (\bigvee_{1 \leqslant i \leqslant n} C_i \rightarrow P)$
Or-group	$P \leftrightarrow \bigvee_{1 \leqslant i \leqslant n} C_i$
Alternative-group	$(P \leftrightarrow \bigvee_{1 \leqslant i \leqslant n} C_i) \wedge \bigwedge_{i<j} (\neg C_i \vee \neg C_j)$
$f1$ requires $f2$	$f1 \rightarrow f2$
$f1$ excludes $f2$	$\neg(f1 \wedge f2)$

- R is the set of relations in the FM. $R = Parent \cup Req \cup Excl$. $Parent \subseteq F \times F$ returns the parent of a given feature. It also forms an acyclic relation called *feature hierarchy*. If $(f1, f2) \in Parent$, then $f1$ is a child of $f2$, $f2$ is the parent of $f1$. $Parent^*$ is the reflexive, anti-symmetric, and transitive closure of $Parent$. $Req \subseteq F \times F$ encodes the F–F *requires* constrains; $Excl \subseteq F \times F$ encodes the F–F *excludes* constrains. For example, ("$f1$","*root*") $\in Parent$; ("$f4$","$f7$") $\in Req$.
- A is the set of attributes in the FM. $A = Opt \cup Mincard \cup Maxcard$. $Opt : F_{AN} \rightarrow \mathbb{B}$ denotes the optionality of a given feature in an And-group, $\mathbb{B} = \{true, false\}$. If $Opt(f)$ returns true, f is optional, otherwise mandatory. For example, Opt("$f3$") = "*true*". $Mincard : FG_{AL} \cup FG_{OR} \rightarrow \mathbb{N}_0$ and $Maxcard : FG_{AL} \cup FG_{OR} \rightarrow \mathbb{N}_0 \cup \{\infty\}$ return the cardinality for a given Or- or Alternative-group. For example, $Mincard$("$fg_{OR}1$") = "1", $Maxcard$("$fg_{OR}1$") = "2".
- *Domain*: $R \cup A \rightarrow 2^C$ and *Range*: $R \cup A \rightarrow 2^C \cup L$ give the set of domain ($\subset 2^C$) or range ($\subset 2^C \cup L$) for some relation $r(\in R)$ or some attribute $a(\in A)$. Here, L denotes literal values of attributes. For example, $Domain(Req(f4,f7)) = \{f4\}$, $Range(Req(f4,f7)) = \{f7\}$; $Domain(Opt(f3)) = \{f3\}$, $Range(Opt(f3)) = \{true\}$.

3.2. Consistency constraints of FMs

We summarize 13 consistency constraints of FMs in terms of the syntax and semantics of FMs. These consistency constraints form a *feature consistency model (FCM)*:

$FCM = \{CC_i, 1 \leqslant i \leqslant 13\}$.

The following set of constraints is by no means an exhaustive list of consistency constraints for FMs, but it lays a foundation for constructing and maintaining a consistent FM. We can define a decision function *consistency(FM)* to judge whether a given FM is consistent or not.

$$consistency(FM) = \begin{cases} true, & \text{if an FM conforms to the } FCM \\ false, & \text{otherwise} \end{cases}$$

3.2.1. Syntactical consistency constraints

Schobbens, Heymans, and Trigaux (2006), Schobbens, Heymans, Trigaux, and Bontemps (2007), Metzger, Heymans, Pohl, and Saval (2007) defined a formal semantics of FMs and several well-formedness rules of FMs based on Free Feature Diagrams (FFD). We extend their well-formedness rules and summarize 10 syntactical consistency constraints (CC_1–CC_{10}) based on the ontology-based formalization of FMs.

CC_1 (***Distinct Identity Constraint***). Every concept has a distinct identity:

$$(NF \cap TF \cap \{root\} = \emptyset) \wedge (FG_{AN} \cap FG_{AL} \cap FG_{OR} \cap \{root\} = \emptyset) \wedge (F_{AN} \cap F_{AL} \cap F_{OR} \cap \{root\} = \emptyset).$$

CC_2 (***Feature Hierarchy Constraint***). The feature hierarchy is a directed acyclic graph:

$$\neg \exists f \in F \cdot (f, f) \in Parent^*.$$

CC_3 (***Root Constraint***). There is a unique feature $root \in F$ that is the direct or indirect parent of all other feature in F.

$$\exists root \in F \cdot (\forall f1 \in F \setminus \{root\} \cdot (f1, root) \in Parent^*) \wedge (\neg \exists f2 \in F \cdot (root, f2) \in Parent^*).$$

CC_4 (***Feature-Closure Constraint***). Every feature except *root* has one parent feature:

$$\forall f1 \in F \setminus \{root\} \cdot \exists f2 \in F \cdot (f1, f2) \in Parent.$$

The constraint CC_4 prevents the existence of the *orphaned* features. For example, the removal of the *Parent* relationship between the feature "$f1$" and the feature "r" in Fig. 1 would cause no parent feature to be defined for the feature "$f1$" any longer, which has to be prevented or resolved.

CC_5 (***Relation-Closure Constraint***). Any relation ($\in R$) must be built between two legal features:

$$\forall f1 \cdot \forall f2 \cdot (f1, f2) \in R \rightarrow f1 \in F \wedge f2 \in F.$$

CC_6 (***Attribute-Closure Constraint***). Any attribute ($\in A$) must be built between a legal features and a literal value ($\in L$):

$$\forall f1 \cdot \forall l \cdot (f1, l) \in A \rightarrow f1 \in F \wedge l \in L.$$

The constraints CC_5 and CC_6 demand that any relation ($\in R$) or any attribute ($\in A$) must be established only between two right objects. For example, the addition of the *Req* relationship between the features "*f*8: Motion Monitor" and "Camera Surveillance" would provoke an inconsistency because the latter is not yet defined as a legal feature ($\in F$).

CC_7 (***Domain-Closure Constraint***). The *Domain* concept can be established between a relation and a concept or between an attribute and a concept:

$$\forall c \cdot \forall ra \cdot c \in Domain(ra) \rightarrow c \in C \wedge (ra \in R \cup A).$$

CC_8 (***Range-Closure Constraint***). The *Range* concept can be established between a relation and a concept or between an attribute and a literal:

$$\forall cl \cdot \forall ra \cdot cl \in Range(ra) \rightarrow (cl \in C \wedge ra \in R) \vee (cl \in L \wedge ra \in A).$$

CC_9 (***Cardinality-Closure Constraint***). Cardinality must be specified for Alternative- or Or-groups:

$$\forall fg \cdot Mincard(fg) \vee Maxcard(fg) \rightarrow fg \in FG_{AL} \cup FG_{OR};$$

CC_{10} (***Cardinality Constraint***). Cardinality must be well-formed:

$$\forall fg \in FG_{AL} \cdot Mincard(fg) = 1 \wedge Maxcard(fg) = 1;$$
$$\forall fg \in FG_{OR} \cdot Mincard(fg) \geqslant 1 \wedge Mincard(fg) \leqslant Maxcard(fg) \leqslant \|fg\|.$$

The constraints CC_9 and CC_{10} reflect cardinality-based feature modeling (Czarnecki et al., 2005). They formulate the Alternative-group and Or-group relationships through a set of well-formed rules about cardinality.

3.2.2. Semantic consistency constraints

von der MaBen and Lichter (2004) presented four situations that lead to semantic inconsistencies in FMs: (1) exclusion between full-mandatory features; (2) exclusion between relative-full-mandatory features; (3) implication between alternative child features; (4) exclusion and implication. We extend their work and introduce two concepts.

Definition 2 (*Mandatory path*). We define a path between a mandatory feature *X* and the *root* feature in an FM as a *mandatory path MandPath (X)* where every intermediate node (feature) is either a mandatory feature in an And-group or the sole child in an Alternative- or an Or-group.

Definition 3 (*Requires chain*). Due to the transitivity of the *Req* relationship, we define a chain from the start node (feature) *S* to the end node *T* as a *requires chain ReqChain(S,T)* where all nodes are connected by the *Req* relationship to each other.

Thus the first two inconsistent situations presented by von der MaBen and Lichter (2004) are merged and extended to the exclusion between any two features in one same or two different mandatory paths. Their fourth inconsistent situation (von der MaBen & Lichter, 2004) is extended to the exclusion between any two features in a requires chain. Examples of extended inconsistencies of FMs are shown in Fig. 2. Correspondingly, we summarize three semantic consistency constraints (CC_{11}–CC_{13}) as follows.

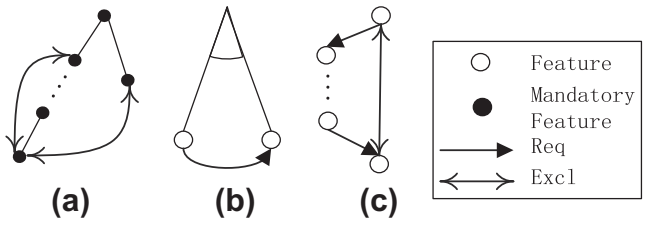

Fig. 2. Examples of semantic inconsistencies of FMs.

CC_{11} (***Excl-MandPath Constraint***). Any two features in one same or two different mandatory paths of an FM cannot have the *Excl* relationship. Counter examples are shown in Fig. 2(a). $\forall f1, f2, X, Y \in F \cdot \neg\exists(Excl(f1, f2) \wedge (f1 \in MandPath(X) \wedge (f2 \in MandPath(X) \vee f2 \in MandPath(Y))))$.

CC_{12} (***Req-Alternative Constraint***). Any two child features in an Alternative-group cannot have the *Req* relationship. A counter example is shown in Fig. 2(b).
$\forall fg_{AL} \in FG_{AL} \cdot \forall f1, f2 \in fg_{AL} \cdot \neg\exists(Req(f1, f2) \vee Req(f2, f1))$.

CC_{13} (***Excl-ReqChain Constraint***). Any two features in a requires chain cannot have the *Excl* relationship. A counter example is shown in Fig. 2(c). $\forall f1, f2 \in F \cdot \neg\exists(ReqChain(f1, f2) \wedge (Excl(f1, f2)))$.

4. Changes to feature models

Based on the above formalization of FMs, we can easily obtain the primitive elements of FMs, which include *concepts* (*non-terminal features*, *terminal features*, *feature groups*), *relations* (*parent*, *requires*, *excludes*) and *attributes* (*name*, *group type*, *optionality*, *cardinality*) of FMs. Since each primitive element of FMs can be changed by one of the meta-change transformations (Huersch, 1997; Rundensteiner, Leem, & Ra, 1998), we suggest a set of primitive operations on FMs in Table 2 (Guo & Wang, 2010). These operations are defined by the cross product of the set of FM primitive elements and the set of meta-changes ('Add', 'Remove', and 'Set'). They represent the changes to FMs at the lowest level of complexity and can compose various complex change operations such as the 16 operations for refactorings and generalizations of FMs (Alves et al., 2006) and the 5 operations for arbitrary edits to FMs (Thum et al., 2009).

Further, we formalize changes to FMs as follows.

Definition 4. A change to FMs *Ch* is a 4-tuple:

$$Ch = (name, args, preconditions, postconditions)$$

where:

- *name* is the identifier of a change. Table 2 lists all the names of primitive operations. In the following chapters, we simplify the notation of changes as *name (args)*.
- $args \in (C \cup R \cup A \cup L)^n, 1 \leqslant n \leqslant 3$, is a list of one or more change arguments. A change could have one, two, or three arguments. Take the FM shown in Fig. 1 for example, to remove the non-terminal feature "*f*1" from the FM, the change *RevNF* has only one argument "*f*1". To modify the name of the node "r", the change *SetName("r", "Home Integration Systems")* is applied. The change *AddRL("rl4–7", "f4", "f7")* is applied to add a requires link "rl4–7" between the features "*f*4" and "*f*7".
- *Preconditions* of a change comprise a set of assertions that must be true to be able to apply the change. If a precondition fails, a change is never performed. For example, the precondition for *RevNF*("*nf*") is $nf \in NF$.

Table 2
Primitive operations on FMs.

Primitive Elements/Meta-Changes		Add	Remove	Set
Concepts	Non-terminal Feature	AddNF(new-nf, old-f)	RevNF(old-nf)	
	Terminal Feature	AddTF(new-tf, old-nf)	RevTF(old-tf)	
	Feature Group	AddFG(new-fg, old-nf)	RevFG(old-fg)	
Relations	Parent Link (pl ∈Parent)	AddPL(new-pl, start-f, end-f)	RevPL(old-pl)	
	Requires Link (rl ∈Req)	AddRL(new-rl, start-f, end-f)	RevRL(old-rl)	
	Excludes Link (el ∈Excl)	AddEL(new-el, start-f, end-f)	RevEL(old-el)	
Attributes	Name (for all entities)			SetName(c, 'name')
	Group Type (for all groups)			SetGT(fg, 'And/Or/Alternative')
	Optionality (for AND-group members)			SetOpt(f, 'Mandatory/Optional')
	Cardinality (for OR-/Alternative-groups*)			SetCard(fg, 'mincard', 'maxcard')

* Every Alternative-group has the fixed cardinality [1..1].

- *Postconditions* of a change comprise a set of assertions that must be true after applying a change. They describe the effect of a change. For example, the postcondition for *RevNF*("*nf*") is $nf \notin NF$.

For example, a full definition of the change *RevNF* can be as follows:

Change	Remove non-terminal feature
Syntax	RevNF("*nf*")
Semantics	Remove a non-terminal feature "*nf*" from an FM
Preconditions	$nf \in NF$
Postconditions	$nf \notin NF$

Similarly, the preconditions and postconditions for other changes can also be deduced according to general logical constraints and the consistency constraints defined above. Two decision functions are defined as follows:

$$preconditions(FM, Ch) = \begin{cases} true, & \text{if an } FM \text{ satisfies the preconditions of a } Ch \\ false, & \text{otherwise} \end{cases}$$

$$postconditions(FM, Ch) = \begin{cases} true, & \text{if an } FM \text{ satisfies the postconditions of a } Ch \\ false, & \text{otherwise} \end{cases}$$

5. Semantics of change

The evolution of FMs can be seen as a sequence of interdependent changes to FMs (Guo & Wang, 2010). Such changes are composed of a set of primitive operations defined in Table 2. A change to FMs can be seen as a mapping between FMs. As shown in Fig. 3, given an *FM* and a requested change *Ch*, the application of the change *Ch* to the *FM* results in another *FM*′, i.e., $FM' = Ch(FM)$, under $preconditions(FM, Ch) = true \wedge postconditions(FM', Ch) = true$.

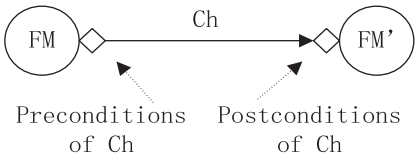

Fig. 3. Applying a change *Ch* to an FM.

Since the application of a single change will not always leave an FM in a consistent state, it often derives a series of additional changes. Hence, the resolution of the requested change requires obtaining and executing these derived changes to maintain the consistency of the FM. Thus:

Definition 5. Given an *FM* and a requested change *Ch*, the semantics of change to *FM* is defined as:

$$SemanticsOfChange(FM, Ch) = (Ch^1, \ldots, Ch^i, Ch^{i+1}, \ldots, Ch^{n-1})$$

where:

- *FM* is a given consistent FM, i.e., $consistency(FM) = true$;
- *Ch* is a requested change that can be applied to the *FM*, i.e., $preconditions(FM, Ch) = true$;
- $FM^1 = Ch(FM)$ is an FM representing the result of applying the requested change *Ch* to the *FM*, i.e., $postconditions(FM^1, Ch) = true$;
- Ch^i, $1 \leqslant i \leqslant n-1$, is a derived change that satisfies the following set of conditions:
  - $FM^{i+1} = Ch^i(FM^i)$, which implies that $preconditions(FM^i, Ch^i) = true$ and $postconditions(FM^{i+1}, Ch^i) = true$;
  - $consistency(FM^i) = false$, $1 \leqslant i \leqslant n-1$, and $consistency(FM^n) = true$.

Thus, as shown in Fig. 4, the final result of applying and resolving the requested change *Ch* to the *FM* is the *FM*′:

$$FM' = FM^n = Ch^{n-1}(\ldots Ch^{i+1}(Ch^i(\ldots Ch^1(Ch(FM))))).$$

Next, how to find and organize these derived changes that resolve the requested change and maintain the consistency of the FM? It is impractical to demand for domain analysts to track down and keep in mind all the changes that are pending. Hence, we adopt the procedural approach (Stojanovic, 2004) to realize the task automatically. The procedural approach comprises five steps (Stojanovic, 2004): first, a request is represented as a series of primitive operations defined in Table 2; second, the illegal operations are prohibited by checking the preconditions of each

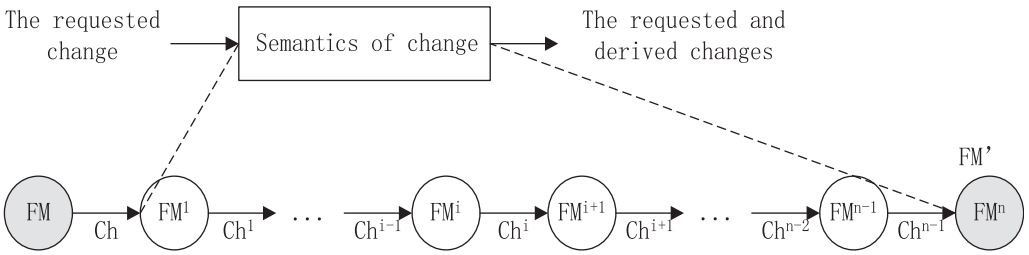

Fig. 4. The semantics of change to an FM.

operation; third, additional operations are derived from the requested operations for keeping consistency; fourth, the execution order of the requested and derived operations is determined; fifth, all the confirmed operations are applied to the FM. Among the above steps, the third and the fourth steps are the key to consistency maintenance of FMs, other steps are straightforward. Hence, we explain how to implement the two steps as follows.

5.1. Dependency matrix

We analyze the cause and effect relationship between primitive operations on FMs and build the *dependency matrix* to conduct the derivation of additional operations from a requested operation. As shown in Table 3, the rows and columns of the matrix list all the primitive operations defined in Table 2. If an element of the matrix

Table 3
The dependency matrix *Dependency*[*i*][*j*].

Changes	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)	(13)	(14)	(15)	(16)
(1)AddNF					X		X		X		X		X	X	X	X
(2)RevNF					X	X		X		X		X		X		X
(3)AddTF							X		X		X		X	X	X	X
(4)RevTF		X				X		X		X		X		X		X
(5)AddFG	X		X				X		X		X		X	X	X	X
(6)RevFG		X		X				X		X		X				
(7)AddPL														X	X	X
(8)RevPL														X		X
(9)AddRL																
(10)RevRL																
(11)AddEL																
(12)RevEL																
(13)SetName																
(14)SetGT															X	X
(15)SetOpt																
(16)SetCard																

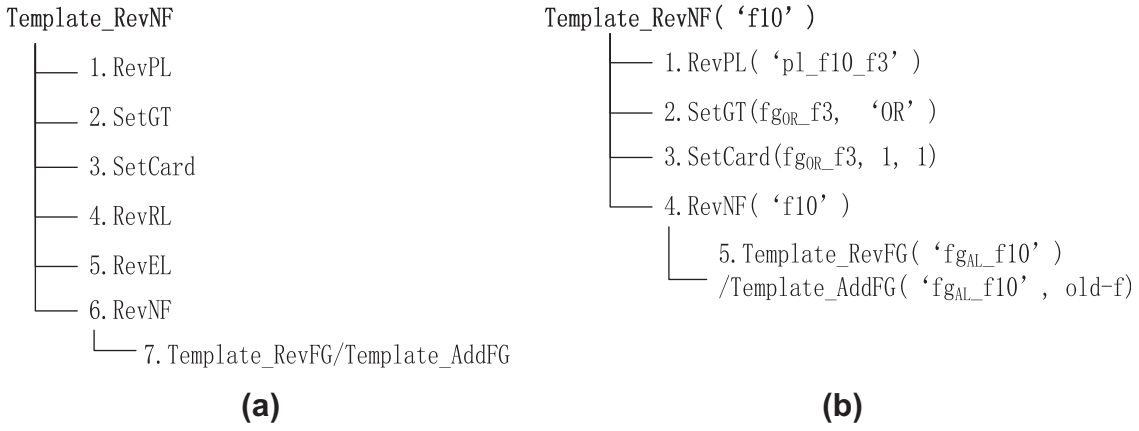

Fig. 5. Operation templates generated by the dependency matrix. (a) The general operation template for resolving the change *RevNF*. (b) A concrete template for resolving the change *RevNF*("*f*10") in the FM shown in Fig. 1.

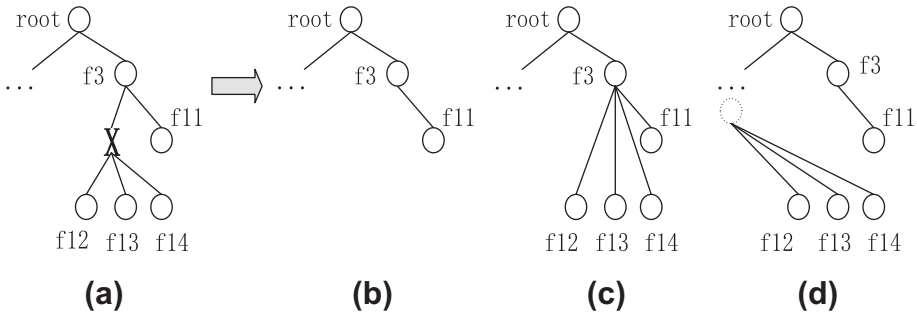

Fig. 6. Three evolution strategies for resolving the operation "RevNF". (a) Applying the *RevNF*("*f*10") alone to the FM shown in Fig. 1. (b) All children are removed. (c) All children are reconnected to the parent. (d) All children are reconnected to another non-terminal feature.

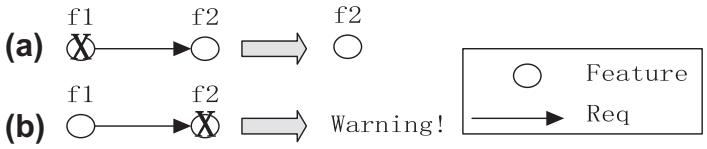

Fig. 7. Two evolution strategies for removing a feature in a *Req* link. (a) Remove the domain. (b) Remove the range.

Dependency[Ch_i][Ch_j] is blank, it means that the operation Ch_i that is assigned to the row i can never induce the operation Ch_j denoting the column j. Otherwise, the operation Ch_i could generate the operation Ch_j when their necessary preconditions and postconditions are fulfilled. The symbol "X" is used as the replacement for all the conditions.

Most of the cause and effect relationships between primitive operations are deduced in terms of the consistency constraints defined in the FCM. The principles for generating the dependency matrix are as follows. First, an operation on a concept would affect the related attributes and relations of the concept. For example, since the *RevNF* operation causes the removal of all "edges" pointing to the feature or from it, the operations *"RevPL"*, *"RevRL"*, and *"RevEL"* are triggered. Second, an operation on an attribute such as the operations "SetName", "SetOpt", and "SetCard" do not initiate additional operations because they do not affect other elements but their own literal values. However, the operation "SetGT" is a special case because it would affect the logical structure of some feature group and thus could cause the operations "SetOpt" and "SetCard".

According to the dependency matrix, an operation would cause a set of additional operations. Further, each of these derived operations would cause another set of operations. Such operation derivation continues to propagate until there is no more new derived operations. All of these derived operations can form a general operation template for resolving a certain operation. Fig. 5(a) shows a general multilevel operation template for resolving the change *RevNF*. The general template would be trimmed as a concrete template when applying to a practical scenario and its operations would be parameterized. Fig. 5(b) demonstrates a concrete operation template for resolving the change *RevNF ("f10")* in the FM shown in Fig. 1. The execution order indicated by the sequence number does not matter very much, but we often handle the operations from outer level to inner and aggregate similar operations.

5.2. Evolution strategy

Most of the primitive operations on FMs can be directly resolved based on the operation templates generated by the dependency matrix. For example, all of the "Add" and "Set" operations defined in Table 2 can be executed straightforwardly once domain analysts determine right parameters. The operations "RevRL" and "RevEL" can also be executed directly. However, the other four "Remove" operations cannot be resolved automatically by the dependency matrix and often need extra decision making by domain analysts. For example, after executing the operations "RevFG" or "RevTF", domain analysts must determine how to handle those non-terminal features at leaf position. Executing the operations "RevNF" or "RevPL" is more complex because domain analysts must determine how to handle the remaining orphaned part of the resulting FM. For these four operations, the operation templates often provide multiple choices, e.g., the step 7 in Fig. 5(a) and the step 5 in Fig. 5(b). Therefore, *evolution strategies* are introduced to direct how to execute these operation and their derived operations resulting not in an arbitrary consistent state.

An evolution strategy unambiguously defines the way in which a change will be resolved. It generally formulates an ordered sequence for the requested change and its derived changes, i.e., the sequence "$Ch, Ch^1, \ldots, Ch^{i-1}, Ch^i, \ldots, Ch^{n-1}$" shown in Fig. 4. Take the operation "RevNF" for example, there are three evolution strategies: removing all children, reconnecting all children to its parent, reconnecting all children to another non-terminal feature. Fig. 6 demonstrates the three evolution strategies for resolving the change *RevNF ("f10")* in the FM shown in Fig. 1. Domain analysts can choose a particular evolution strategy in order to tailor the evolution of FMs to suit their needs. Resolving the operation "RevPL" can also apply these three evolution strategies. Resolving the operations "RevFG" and "RevTF" often needs adding additional features as terminal features.

In addition, two evolution strategies are used for resolving the "RevTF" or "RevNF" operations on the *Req* links. As shown in Fig. 7, for the situation (b), i.e., "*f*1 *requires f*2", "*f*2" cannot be removed arbitrarily. In this case, domain analysts would be warned that the requested change could be an illegal operation. If domain analysts confirm the requested change, then they can first remove the link "*Req*(*f*1,*f*2)" and then remove the feature "*f*2".

6. Implementation

We implemented our approach[2] based on FeatureIDE, which is an open-source Eclipse-based IDE that supports building program

[2] An implementation of our approach is available in http://code.google.com/p/fmconmain/.

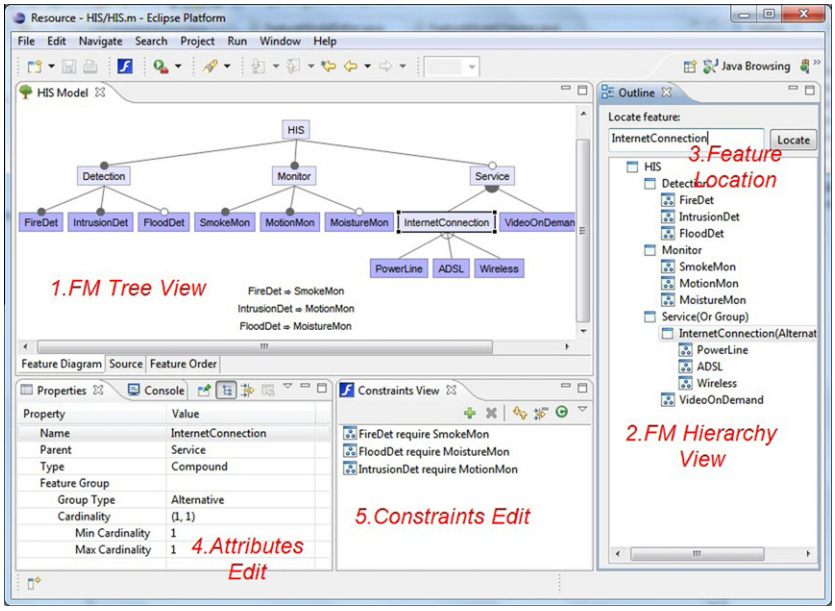

Fig. 8. Extended FM editor based on FeatureIDE.

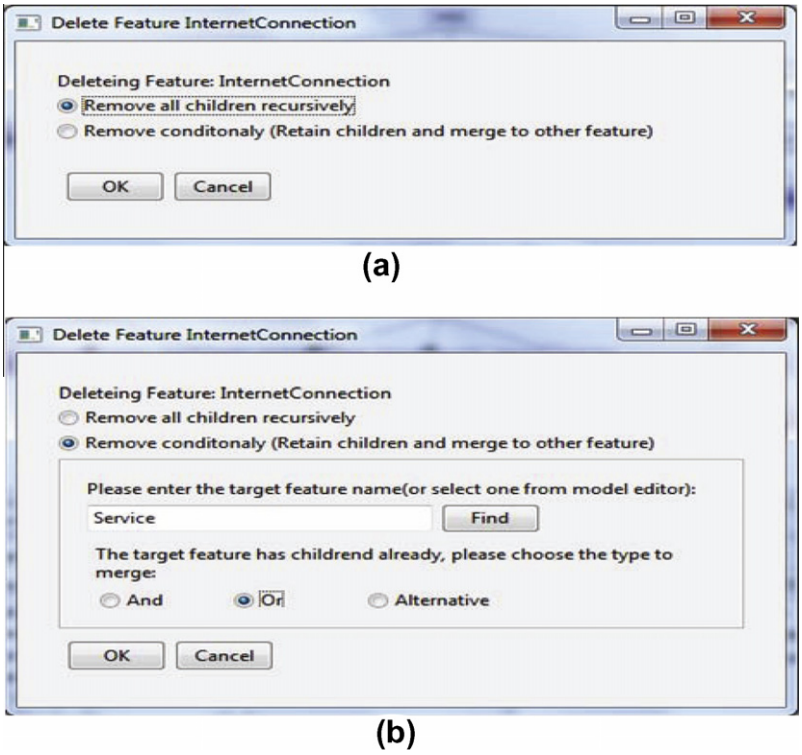

Fig. 9. Evolution strategies implementation.

families following the *AHEAD*[3] architecture model and provides tools for the feature oriented design process and the implementation of SPLs. An extended FM editor based on FeatureIDE is shown in Fig. 8. It provides users with two views of FMs: tree view and hierarchy view. Users can input the keyword of some feature and then locate it. Attributes of features and feature groups, defined in Table 2, can be easily edited. Constraints also have a separate view and an edit area.

Evolution strategies are implemented by an interactive manner. For example, if the change *RevNF ("f10: InternetConnection")* is applied in the FM shown in Fig. 1, a dialog box (as shown in Fig. 9(a)) is displayed for users to choose an evolution strategy. Users can choose to remove all children of the feature "*f*10" (the evolution strategy defined in Fig. 6(b)) or to reconnect these children to another non-terminal feature (the evolution strategies defined in Fig. 6(c) and (d)). If the latter is chosen, the dialog box is extended to prompt users to input the target feature, as shown in Fig. 9(b). Note that the target feature group and the children group of the removed feature could have different group type,

[3] http://userweb.cs.utexas.edu/users/schwartz/.

so users must confirm a certain group type to merge the two feature groups.

7. Evaluation

According to Stojanovic (2004), the computation complexity of resolving a change to FMs (e.g., *RevNF*) is about $O(n^m)$ where m is the average depth of the feature hierarchy starting from the considering feature and n the average number of child features. Generally, m and n are not large numbers because FMs usually contain limited layers and limited children for one feature. Thus, we make a preliminary evaluation that our approach accomplishes the consistency maintenance of evolving FMs in an acceptable time.

Further, we give more comprehensive evaluation by experimental studies. Although industries reported FMs with hundreds or thousands of features (Loesch & Ploedereder, 2007; Steger et al., 2004), authors typically published only a small excerpt of their FMs. Large FMs are difficult to find for a thorough evaluation. Thus, we adopt Thum's method (Thum et al., 2009) to perform experiments using randomly generated FMs with different characteristics.

7.1. Experimental setup

We first generated FMs randomly and then performed a set of primitive operations (defined in Table 2) randomly on the generated FMs. Based on the dependency matrix, a set of additional operations are derived automatically from the requested operations. Further, according to predefined evolution strategies, the derived operations are executed automatically to maintain the consistency of those changed FMs. During the above process, we parametrically control the size of FMs, the number and kind of operations, and the kind of evolution strategies for a thorough runtime evaluation.

Independent parameters in our experiment are (a) the number of features in an FM, (b) number of operations, (c) kind of operations, (d) kind of evolution strategies. The time needed to perform the requested and derived operations is measured as a dependent variable. To reduce the fluctuations in the dependent variable caused by the random generation, we performed 200 repetitions for each configuration of independent parameters, i.e., we generated 200 random FMs with the same parameters and each performed the same number of random operations of the same kind. All measurements were performed on the same Windows 7 PC with Intel Core Duo CPU 1.5 GHz and 3 GB RAM.

7.1.1. Feature models generation

The algorithm to randomly generate FMs of size n is as follows (Thum et al., 2009): starting with a single *root* node, it runs several iterations. In each iteration, an existing node without children is randomly selected, and one to ten (random amount) of child nodes are added. Those child nodes are connected either by And- (50% probability), Or- (25% probability) or Alternative-group (25% probability). Children in an And-group are optional by a 50% probability. This iteration is continued until the FM has n features. All features with children are considered non-terminal. Moreover, we also generate cross-tree constraints (*requires* and *excludes*). For every 10 features, one constraint is generated by the following algorithm: two different features are randomly selected, and then are connected randomly by *requires* (50% probability) or *excludes* (50% probability) link.

The above generated FMs can be easily translated into propositional formulas according to the rules give in Table 1. We use the SAT solver *sat4J*[4] to validate these FMs and discard all FMs that do not have a single valid configurations (mostly by unfortunate choice of cross-tree constraints). We repeat the entire process until the appropriate number of valid FMs are generated.

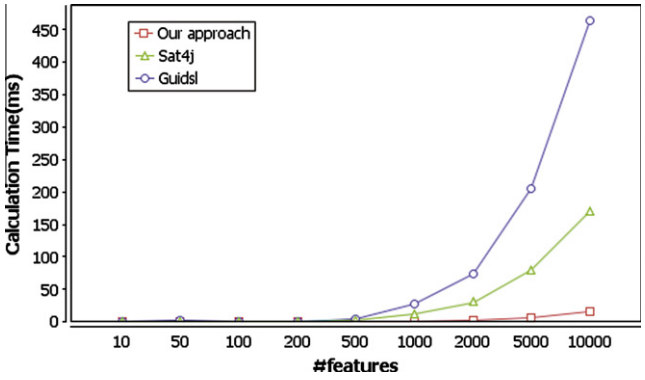

Fig. 10. Calculation time in milliseconds for different scales of FMs using consistency checking (Sat4j and Guidsl) and consistency maintenance (our approach).

We fixed the following parameters: maximum number of children = 10; type of child group = (50%,25%,25%); optional child = 50%; number of cross-tree constraints = 0.1 ∗ n; variables in cross-tree constrains = 2. According to Thum's survey (Thum et al., 2009), these parameters are backed up by most of the surveyed FMs and represent a rough average. Thus, these generated FMs basically reflect the characteristics of realistic FMs.

7.1.2. Operation generation

We randomly generated operations on an FM as well. Our generator takes an FM and the number of operations as input. The 20 primitive operations defined in Table 2 are implemented. They are classified as three main types: 'add', 'remove', and 'set'. Our generator can limit the kind of input operations to 'add', 'remove', 'set', or 'arbitrary'. For a fixed number of input operations, 'arbitrary' operations are composed of 'add' (33% probability), 'remove' (33% probability), and 'set' (33% probability) operations.

7.2. Experimental results and discussion

7.2.1. Effectiveness

We first verify whether our approach can ensure the consistency of the resulting FMs after the requested changes are resolved. We varied the size of the generated FMs between 10 and 10,000 features. For each FM, we performed 10 random arbitrary operations. 200 repetitions are performed for each model size. Each resulting FM is checked by Sat4j and Guidsl. *Guidsl*[5] is a tool developed by Batory (2005) that relies on grammars definition and propositional logic to support feature modularizations and their compositions. Results show that all the resulting FMs generated by our approach are validated by Guidsl and Sat4j. Therefore, our approach can effectively maintain the consistency of evolving FMs.

7.2.2. Number of features

In the same experimental setting as the above experiment, we also measured how calculation time scales as FMs increase in size. We also perform 10 random arbitrary operations on each FM

[4] http://www.sat4j.org.
[5] http://userweb.cs.utexas.edu/schwartz/ATS/fopdocs/guidsl.html.

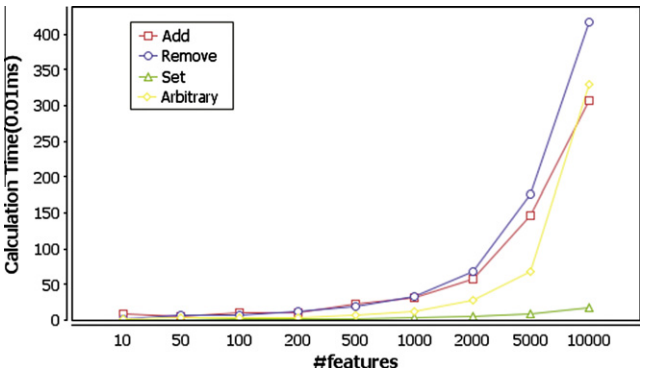

Fig. 11. Calculation time in 0.01 ms for different kinds of operations.

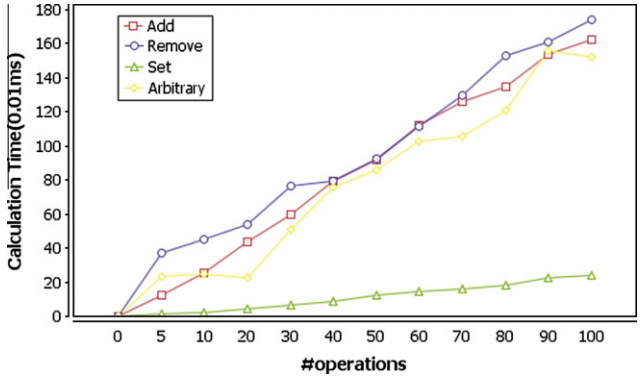

Fig. 12. Calculation time in 0.01 ms for different number of operations on an FM with 1000 features.

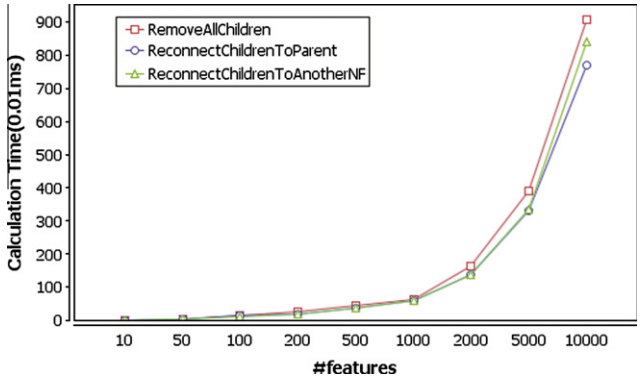

Fig. 13. Calculation time in 0.01 ms for different evolution strategies of the "RevNF" operation.

whose size varies from 10 to 10,000. That is repeated 200 times and we only take the mean value. Fig. 10 shows the results of this measurement.

To obtain a consistent FM from the requested operations on an original FM, previous approaches to ensuring the consistency of FMs mostly execute the operations first and then check the consistency of the resulting FM. During the process, most of the time is spent on consistency checking using various off-the-shelf solvers. Instead, our approach directly resolves the requested operations on the original FM and preserves the consistency of the resulting FM by construction. Fig. 10 compares the calculation time (the mean value for 200 repetition) to generate a consistent FM from 10 random operations on an original FM whose size varies from 10 to 10,000 using these two approaches. Here, we use *Guidsl* and Sat4j to check the consistency of the resulting FM.

For small FMs (<500 features), there is no marked difference among the calculation time for the three cases. For large FMs (1000 features), our approach has better efficiency than previous approaches using Guidsl and Sat4j. Even for very large FMs with up to 10,000 features, our approach only spends 7.2 ms.

7.2.3. Kind of operations

We performed the same measurement varying the model size from 10 to 10,000, but distinguished different kinds of operations. We distinguished between four kinds of operations: 'add', 'remove', 'set', and 'arbitrary'. Again we applied 10 random operations of this classification (i.e., 10 adds, 10 removes, 10 sets, or 10 arbitrary operations) to each FM.

Fig. 11 shows the results of our measurement (mean value of 200 repetitions for each combination of FM size and kind of operations). The 'remove' operations spend the most time and the 'set' operations spend the least. There is not much difference between 'remove' and 'add' operations. The calculation time of 'arbitrary' operations is in the middle of that of the other three kinds of operations.

7.2.4. Number of operations

We measured the calculation time when fixing the FM size to 1000 features and varying the number of operations from 0 to 100. Again we distinguished between the four kinds of operations: 'add', 'remove', 'set', and 'arbitrary'.

Fig. 12 shows the results of our measurement (mean value of 200 repetitions for each combination of the number and kind of operations in an FM with 1000 features). The 'set' operations still spend the least time and increase slowly with rising number of operations. The 'add', 'remove', and 'arbitrary' operations spend similar time. Their calculation time increases markedly with rising number of operations.

7.2.5. Kind of evolution strategies

We measured the calculation time varying the FM size from 10 to 10,000 for different kinds of evolution strategies. We distinguished between three evolution strategies for removing a non-terminal feature (the "RevNF" operation): removing all children, reconnecting the children to the parent, and reconnecting the children to another non-terminal feature, as shown in Fig. 6(b)–(d). For each situation, we applied 10 "RevNF" operations with random operation objects to each FM.

Fig. 13 shows the results of our measurement (mean value of 200 repetitions for each combination of FM size and each kind of evolution strategy). There is no significant difference between the three evolution strategies in simulation environment. But in practice, only the first evolution strategy can be implemented automatically, as shown in Fig. 9(a). The execution of the last two evolution strategies requires users' interactive participation, as shown in Fig. 9(b).

7.3. Discussion and threats to validity

Experiments show that our approach works effectively and efficiently for large FMs. Mostly independent from the kind and number of operations, our approach can generate a consistent resulting FM in less than 0.1 second even for very large FM (up to 10,000 features). This comfortably allows an implementation in our extended FM editor that shows how to resolve a change to an FM on the fly and maintain the consistency of the FM at runtime.

Threats to internal validity are influences that can affect the calculation time that have not been considered. We cannot guarantee that computation time depends on certain shapes of an FM, or

certain kinds of operations. Especially, we cannot guarantee "Arbitrary" operations are composed of "Add", "Remove", and "Set" operations exactly in the proportion of 1:1:1. Since the calculation time for "Add" and "Remove" operations is markedly greater than that for "Set" operations, the uneven proportion of the three kinds of operations could cause fluctuation in the calculation time for "Arbitrary" operations. However, to avoid effects of certain FMs, all of input FMs are generated automatically by simulating known FMs and each measurement is repeated 200 times with freshly generated FMs.

Threats to external validity are conditions that limit our ability to generalize the results of our experiment to industrial practice. First, we generated FMs with the described algorithm and parameters, and confirmed that they align well with those known FMs acquired from existing publications in SPL community. Second, we generated a set of primitive operations on FMs by the cross product of all of FM primitive elements and a set of meta-changes ("Add", "Remove", and "Set"). We cannot guarantee these primitive operations are complete and typical in practice, however all reasonable operations we wanted to perform in our manual experiments, including the high-level operations defined by Alves et al. (2006), Thum et al. (2009), could be performed with one or a sequence of these operations.

8. Related work

Many formalizations and notations of FMs have been proposed (Batory, 2005; Czarnecki et al., 2005; Kang et al., 1990). Schobbens et al. formalized feature diagrams and detailed their generic semantics through a generic construction called Free Feature Diagrams (Metzger et al., 2007; Schobbens et al., 2006, 2007). However, there is a lack of detailed discussions about operations on FMs and their interrelationships. We give a new formalization of FMs from an ontological perspective. Based on this, we can easily obtain the primitive elements of FMs and then generate a set of primitive operations on FMs.

Some researchers classified the modifications of FMs as specializations (Czarnecki et al., 2005), refactorings (Alves et al., 2006), generalizations (Alves et al., 2006; Thum et al., 2009), or arbitrary edits (Thum et al., 2009). Czarnecki et al. (2005) introduced specializations for deriving configurations of an FM, which result in FMs where some products are deleted. Janota and Kiniry (2007) formalized specifications between two FMs having the same set of features. Alves et al. (2006) discussed refactorings and generalizations that maintain the set of products or add new products to an SPL. They also suggested 16 operations for refactorings and generalizations. Thum et al. (2009) complemented five additional operations for arbitrary edits. These operations work at a high level to explain the effects of FM evolution, but they are still not enough to explain the details of the FM evolution process. That is, how one changes an FM X into a target FM Y using a sequence of concrete and sound operations is still not obvious. We suggest a set of primitive operations on FMs by the cross product of all of FM primitive elements and three meta-changes. We do not think primitive operations are better than those aggregated or high-level operations because primitive operations could aggravate the consistency problem (many primitive operations will necessarily leave an FM in an inconsistent state). However, primitive operations are fit to explain the details of resolving a requested change to an FM and maintaining the consistency of the FM.

von der MaBen and Lichter (2004) proposed a framework for describing deficiencies of FMs. They characterized inconsistency as one of the most severe deficiencies of FMs. They also identified four situations that lead to semantic inconsistencies of FMs. Based on their work and Schobbens et al.'s (Metzger et al., 2007; Schobbens et al., 2006, 2007), together with our ontology-based formalization of FMs, we define a set of syntactical and semantic consistency constraints as the well-formedness rules of FMs.

Many approaches focus on automated analysis of deficiencies of FMs (Benavides et al., 2010). Mannion (2002) first used propositional formulas to analyze FMs. Batory (2005) proposed an approach to debugging FMs using off-the-shelf SAT solvers. Czarnecki and Wasowski (2007) applied BDD tools to analyze FMs. Benavides et al. (2005) first adopted constraint programming to analyze FMs. Wang, Li, Sun, Zhang, and Pan (2005) first proposed the automated analysis of FMs using description logic. These approaches mostly use SAT (Batory, 2005; Thum et al., 2009), BDD (Czarnecki & Wasowski, 2007), CSP solvers (Benavides et al., 2005), or description logic reasoning engines (Wang et al., 2005) to automate various reasoning tasks, e.g., checking satisfiability, detecting "dead" features, computing commonalities. They, however, suffer from the NP-hard problem of feature combinatorics and thus take a long time to perform with large FMs (Batory et al., 2006). Moreover, these approaches only focus on the checking of deficiencies of FMs. Our approach resolves the possible inconsistencies caused by the requested changes to FMs by analyzing and applying the semantics of change to FMs and the interrelationships among primitive operations on FMs. It limits the consistency maintenance of evolving FMs in a local range affected by the requested changes not in the whole FM, which guarantee the efficiency and scalability of our approach for large FMs.

Trinidad et al. (2008) provided a framework for explaining deficiencies of FMs based on constraint programming, but they do not give a solution to the deficiencies and the scalability of their approach is also not clear. White et al. (2010) detected errors on the configurations of an FM, and proposed changes in the configurations according to features to be selected or deselected to correct the errors. Our work purely focuses on the evolution and consistency maintenance of FMs themselves, not the configurations of FMs. Some researchers also investigated relationships to other variability models or other core assets in SPLs. For example, Kastner and Apel (2008) proposed a formal approach to type-checking SPLs on the background of an FM. Metzger et al. (2007) proposed a formalization of Orthogonal Variability Models (OVMs) and also followed the SAT approach to analyze OVMs automatically. Lauenroth and Pohl (2008) presented a consistency checking technique for dynamic properties of the domain requirements specification based on OVMs.

This paper greatly expands our previous work (Guo & Wang, 2010). In that paper (Guo & Wang, 2010), we only suggested a set of primitive operations on FMs and proposed a preliminary framework for analyzing the semantics of change to FMs. In this paper, we systematically give an ontology-based formalization of FMs and a set of consistency constraints, which build the theoretical foundations for generating primitive operations on FMs and analyzing their interrelationships. We detail how to obtain and apply the dependency matrix and the evolution strategies for maintaining consistency of FMs. We also perform experiments to verify the effectiveness and efficiency of our approach.

9. Conclusions

This paper proposes an approach to consistency maintenance for evolving FMs. We formalize FMs from an ontological perspective and suggest a set of syntactical and semantic consistency constraints as the well-formedness rules of FMs. We generate a set of primitive operations on FMs and analyze their interrelationships. By analyzing the semantics of change to FMs, the process of resolving a requested change to an FM and maintaining the consistency of the FM is decomposed as a sequence of interdependent

operations. The dependency matrix and evolution strategies are introduced to obtain and organize the operations sequence. Our approach is implemented and applied to randomly generated FMs. By experiments, we verify that our approach can effectively and efficiently maintain the consistency of evolving FMs with thousands of features.

Our approach identifies a desirable property of FMs whose consistency maintenance can be achieved by construction. It is particularly suitable for incremental management of FMs and their evolution in practice.

Our approach improves significantly the complexity of the problem since we do not study a whole FM but the only part that changes since last consistent version. Although our approach depends on the completeness of the consistency constraints and the primitive operations we define, and if there were any situations where they are not applicable, previous approaches can still be applied. So our approach has to be seen as an optimization for those cases where our consistency constraints and primitive operations can apply.

Although FMs are the most popular kind of variability models, there are other alternatives that are still used and that we think our approach can be adapted to support consistency maintenance. Specifically, we plan to apply our approach to FMs with attributes (Benavides et al., 2010), OVMs (Metzger et al., 2007), and requirements consistency checking (Lauenroth & Pohl, 2008).

Acknowledgments

The authors thank Prof. Robyn R. Lutz (Iowa State University) and Wei Zhang (Peking University) for their invaluable comments on the earlier draft of this paper. The authors also thank the anonymous reviewers and the attendees of SPLC'10 for their greatly helpful comments. Funding was provided by the National Natural Science Foundation of China (NSFC No. 60773088), the National High-tech R&D Program of China (863 Program No. 2009AA04Z106), the Key Program of Basic Research of Shanghai Municipal S&T Commission (No. 08JC1411700), the European Commission (FEDER) and Spanish Government under the CICYT project SETI (TIN2009-07366), and the Andalusian Local Government under the projects THEOS (TIC-5906) and ISABEL (P07-TIC-2533).

References

Alves, V., Gheyi, R., Massoni, T., Kulesza, U., Borba, P., & Lucena, C. (2006). Refactoring product lines. In *Proceedings of GPCE'06, Portland, Oregon, USA* (pp. 201–210).

Batory, D. (2005). Feature models, grammars, and propositional formulas. In *Proceedings of SPLC'05, Rennes, France* (pp. 7–20).

Batory, D., Benavides, D., & Ruiz-Cortes, A. (2006). Automated analysis of feature models: challenges ahead. *Communications of the ACM, 49*, 45–47.

Benavides, D., Martin-Arroyo, P. T., & Cortes, A. R. (2005). Automated reasoning on feature models. In *Proceedings of CAiSE'05, Porto, Portugal* (pp. 491–503).

Benavides, D., Segura, S., & Cortes, A. R. (2010). Automated analysis of feature models 20 years later: A literature review. *Information Systems, 35*, 615–636.

Clements, P., & Northrop, L. (2001). *Software product lines: Practices and patterns*. Boston, MA, USA: Addison-Wesley.

Czarnecki, K., & Wasowski, A. (2007). Feature diagrams and logics: there and back again. In *Proceedings of SPLC'07, Kyoto, Japan* (pp. 23–34).

Czarnecki, K., Helsen, S., & Eisenecker, U. W. (2005). Formalizing cardinality-based feature models and their specialization. *Software Process: Improvement and Practice, 10*, 7–29.

Gruber, T. R. (1993). A translation approach to portable ontologies. *Knowledge Acquisition, 5*, 199–220.

Gruber, T. (2008). Ontology. In *Encyclopedia of database systems*. Springer-Verlag.

Guo, J., & Wang, Y. (2010). Towards consistent evolution of feature models. In *Proceedings of SPLC'10, Jeju Island, South Korea. LNCS* (Vol. 6287, pp. 451–455).

Haase, P., & Stojanovic, L. (2005). Consistent evolution of OWL ontologies. In *Proceedings of ESWC'05, Heraklion, Greece* (pp. 182–197).

Huersch, W. (1997). Maintaining consistency and behaviour of object-oriented systems during evolution. *ACM SIGPLAN Notices, 32*, 1–21.

Janota, M., & Kiniry, J. (2007). Reasoning about feature models in higher-order logic. In *Proceedings of SPLC'07, Kyoto, Japan* (pp. 13–22).

Kang, K. C., Cohen, S. G., Hess, J. A., Novak, W. E., & Peterson, A. S. (1990). *Feature-oriented domain analysis (FODA) feasibility study*. Technical Report CMU/SEI-90-TR-021, Software Engineering Institute, CMU.

Kang, K. C., Lee, J., & Donohoe, P. (2002). Feature-oriented product line engineering. *IEEE Software, 19*, 58–65.

Kastner, C., & Apel, S. (2008). Type-checking software product lines – A formal approach. In *Proceedings of ASE'08, L'Aquila, Italy* (pp. 258–267).

Lauenroth, K., & Pohl, K. (2008). Dynamic consistency checking of domain requirements in product line engineering. In *Proceedings of RE'08, Barcelona, Spain* (pp. 193–202).

Loesch, F., & Ploedereder, E. (2007). Optimization of variability in software product lines. In *Proceedings of SPLC'07, Kyoto, Japan* (pp. 151–162).

Lutz, R. R. (2008). Enabling verifiable conformance for product lines. In *Proceedings of SPLC'08, Limerick, Ireland* (pp. 35–44).

Mannion, M. (2002). Using first-order logic for product line model validation. In *Proceedings of SPLC'02, San Diego, CA, USA* (pp. 176–187).

Metzger, A., Heymans, P., Pohl, K., & Saval, P.-Y. S. G. (2007). Disambiguating the documentation of variability in software product lines: A separation of concerns, formalization and automated analysis. In *Proceedings of RE'07, New Delhi, India* (pp. 243–253).

Pohl, K., Bockle, G., & van der Linden, F. (2005). *Software product line engineering: foundations, principles, and techniques*. Berlin, Heidelberg: Springer-Verlag.

Rundensteiner, E., Leem, A., & Ra, Y. (1998). Capacity-augmenting schema changes on object-oriented databases: towards increased interoperability. In *Proceedings of OOIS'98, Paris, France* (pp. 349–368).

Schobbens, P.-Y., Heymans, P., & Trigaux, J.-C. (2006). Feature diagrams: A survey and a formal semantics. In *Proceedings of RE'06, Minneapolis/St.Paul, Minnesota, USA* (pp. 136–145).

Schobbens, P.-Y., Heymans, P., Trigaux, J.-C., & Bontemps, Y. (2007). Generic semantics of feature diagrams. *Computer Networks, 51*, 456–479.

Steger, M., Tischer, C., Boss, B., Muller, A., Pertler, O., Stolz, W., et al. (2004). Introducing PLA at Bosch gasoline systems: Experiences and practices. In *Proceedings of SPLC'04, Boston, MA, USA* (pp. 34–50).

Stojanovic, L. (2004). *Methods and tools for ontology evolution*. Ph.D. Thesis, University of Karlsruhe.

Sugumaran, V., Park, S., & Kang, K. C. (2006). Software product line engineering: Introduction. *Communications of the ACM, 49*, 28–32.

Thum, T., Batory, D. S., & Kastner, C. (2009). Reasoning about edits to feature models. In *Proceedings of ICSE'09, Vancouver, Canada* (pp. 254–264).

Trinidad, P., Benavides, D., Duran, A., Ruiz-Cortes, A., & Toro, M. (2008). Automated error analysis for the agilization of feature modeling. *Journal of Systems and Software, 81*, 883–896.

von der MaBen, T., & Lichter, H. (2004). Deficiencies in feature models. In *Proceedings of workshop on software variability management for product derivation, SPLC'04, Boston, MA, USA*.

Wang, H., Li, Y., Sun, J., Zhang, H., & Pan, J. (2005). A semantic web approach to feature modeling and verification. In *Proceedings of workshop on semantic web enabled software engineering (SWESE'05)*.

White, J., Benavides, D., Schmidt, D. C., Trinidad, P., Dougherty, B., & Cortes, A. R. (2010). Automated diagnosis of feature model configurations. *Journal of Systems and Software, 83*, 1094–1107.