Consistency Management Techniques for Variability Modeling

Alexander Felfernig[*,a], Thomas Gruber[a], Martin Stettinger[a]

[a] Graz University of Technology, Institute for Software Technology, Applied Software Engineering, Inffeldgasse 16b/2, A-8010 Graz, Austria.

Abstract. *Feature models are a means to represent software variability. Due to their logical grounding, such representations allow for automated reasoning about specific model properties. In this article, we show how the concepts of conflict detection and model-based diagnosis can be applied to analyze and improve the quality of a feature model. The example feature model used in this context is based on the variability information of a real-world event management environment.*

Keywords. Variability Modeling • Feature Models • Consistency Management • Configuration

1 Introduction

Requirements Engineering (RE) is considered as a crucial phase in software development (Kop and Mayr 1998; Leffingwell and Widrig 2003; Mayr and Kop 2002; Mayr et al. 2007). Low quality requirement models can trigger enormous follow-up costs manifested, for example, in terms of re-design needs, re-implementation, debugging, and testing. A specific task in the RE context is to model software variability (Benavides et al. 2010; Kang et al. 1990; Thum et al. 2009), i.e., which combinations of components are allowed when delivering a software. Feature models are an approach to variability modeling especially developed to support the construction of software product lines (Kang et al. 1990). Due to the increasing complexity of feature models, the identification of inconsistencies becomes a challenging task. In this article, we discuss concepts that support the engineering of feature models on the basis of consistency-based analysis approaches, more specifically, conflict detection (Junker 2004) and model-based diagnosis (Reiter 1987). These approaches support the automated identification of inconsistencies in feature models and can be used for further purposes such as *automated testing and debugging*, *redundancy detection*, and software quality improvement related activities (Felfernig et al. 2014).

Feature models can be distinguished with regard to the degree of expressiveness of feature representations and corresponding constraints (Benavides et al. 2010). *Basic feature models* (Kang et al. 1990) allow the definition of basic relationships between features, for example, the inclusion of a specific feature *x excludes* another feature *y*. *Cardinality-based feature representations* (Czarnecki et al. 2005) additionally allow the definition of cardinalities of relationships (in basic models, the upper bound of cardinalities is 1). Finally, *extended feature models* allow the definition of feature properties represented as feature attributes (Batory 2005). Without loss of generality, our examples are based on *basic feature models*. The presented concepts can also be applied to the advanced feature model types introduced in (Batory 2005; Czarnecki et al. 2005).

Designing feature models is an error-prone activity that results from a cognitive overload of engineers engaged in the development and maintenance of such models (Benavides et al. 2013). Artificial Intelligence (AI) techniques can support the identification of different types of inconsistencies in an automated fashion (Benavides et al. 2010). Inconsistency identification and resolution

* Corresponding author.
E-mail. alexander.felfernig@ist.tugraz.at

techniques for feature models have been introduced, for example, by (White et al. 2010) where dead features are interpreted as a model anomaly (a feature part of a model that can never be included in a configuration). In this context, feature models are translated into corresponding constraint-based representations (Tsang 1993) and the identification of dead features is directly encoded as a constraint satisfaction problem. An overview of different types of model anomalies that can occur in feature models is provided in (Benavides et al. 2010). These anomalies are also the basis for the discussions in this article.

Feature models can be regarded as specific type of configuration models (Felfernig et al. 2014) represented, for example, in terms of constraint satisfaction problems (Tsang 1993). The diagnosis of inconsistent constraint-based representations has first been introduced in (Bakker et al. 1993) where inconsistencies in constraint models are resolved on the basis of the concepts of model-based diagnosis (Reiter 1987). This work has been extended to test scenarios where diagnoses are calculated on the basis of a given set of test cases (Felfernig et al. 2004). This approach is based on the idea of the identification of minimal conflict sets (Junker 2004) and their resolution based on the concept of a Hitting Set Directed Acyclic Graph (HSDAG) (Reiter 1987). More recent approaches to model-based diagnosis omit the calculation of conflict sets and directly determine minimal diagnoses (see, e.g., Felfernig et al. 2018).

In this article, we focus on aspects related to the identification of erroneous constraints in feature models. In this context, our major contributions are the following. On the basis of a formalization of feature models as constraint satisfaction problems we discuss different types of inconsistencies that can occur in feature model development. Second, we show how these inconsistencies can be localized on the basis of the concepts of conflict detection and model-based diagnosis. Finally, we discuss open issues for future research. Our examples are based on a feature model derived from a commercial event management software.

The remainder of this article is organized as follows. In Section 2, we introduce a feature model from the EventHelpr[1] environment. This model serves as a working example throughout this article. In Section 3, we sketch logic approaches to detect and resolve inconsistencies. In Section 4, we introduce and formalize different types of inconsistencies that can occur when developing feature models. Ways to resolve inconsistencies in our example feature model are discussed in Section 5. Section 6 provides a discussion of different open research issues. This article is concluded with Section 7.

2 Example Feature Model and Semantics

A feature model defines a complete set of allowed configurations, i.e., the combinations of features that can be jointly included in a configuration. The modeling concepts that can be used to build feature models are: *features*, *relationships between features* (represented as constraints), and so-called *cross-hierarchy constraints* (Kang et al. 1990). For a detailed discussion of feature modeling techniques we refer to (Batory 2005; Czarnecki et al. 2005; Kang et al. 1990).

Since feature models are a basic mechanism to define variability properties, we can interpret these models as configuration knowledge bases (Felfernig et al. 2014). Features are the structural elements of feature models. They can be either *included in* or *excluded from* a specific configuration. Each feature is associated with the domain $\{true, false\}$ where *true* specifies *feature inclusion* (into a corresponding configuration) and *false* defines *feature exclusion*. Features are connected via constraints (partially defined by relationships) that specify additional restrictions on possible combinations of features. In the following, we introduce six different types of constraints that are typically used in feature modeling: *mandatory*, *optional*, *alternative*, *or*, *requires*, and *excludes*.

The formalization of these constraints is a basis for performing automated reasoning on feature model properties. Feature configuration tasks can

[1] www.eventhelpr.com.

be formalized as *Constraint Satisfaction Problems* (*CSPs*) (Tsang 1993) as follows (see Definition 1).

Definition 1 (Feature Configuration Task). A *feature configuration task* can be defined by a triple (F, D, C) where F represents a set of features and each feature $f_i \in F$ has an associated domain $dom(f_i) \in D = \{true, false\}$. Finally, $FM \cup CREQ$ represents a set of constraints $c_i \in C$ where FM are domain model constraints contained in the feature model and $CREQ$ is a set of customer requirements used to specify customer-specific requirements with regard to a feature configuration ($FM \cup CREQ = C$).

Based on a feature configuration task definition, different configurations can be determined. A *feature configuration* (solution to a feature configuration task) can be defined as follows.

Definition 2 (Feature Configuration). A *feature configuration* is a complete set of value assignments ($val(f_i) \in \{true, false\}$) to features $f_i \in F$. A feature configuration is *consistent* if the value assignments are consistent with the constraints in C. Furthermore, it is regarded as *valid* if it is consistent and *complete* (each variable has a corresponding value assignment).

Constraint Types. The following six constraint types are often used to build feature models (Benavides et al. 2010). An example of a feature model using these constraint types is depicted in Figure 1. In the following, we define the semantics of these constraint types.

Mandatory. A feature f_2 is in a mandatory relationship with feature f_1 if whenever f_1 is part of the configuration, f_2 must be part of the configuration (and vice-versa). On the logical level, this property can be defined in terms of an equivalence, i.e., $f_1 \leftrightarrow f_2$. An example of a mandatory relationship in Figure 1 is the *participation* feature: in any case, the type of participation in an event has to be defined.

Optional. A feature f_2 is in an optional relationship with feature f_1 if whenever f_1 is part of the configuration, f_2 can be part of the configuration. Vice-versa, if f_2 is part of the configuration, f_1 must be included. On the logical level, this property can be defined in terms of an implication, i.e., $f_2 \rightarrow f_1$. An example of an optional relationship in Figure 1 is the *content* feature: sending emails and uploading photos is considered as optional in every EventHelpr instance.

Alternative. This type of relationship specifies an "xor" semantics, i.e., exactly one of a given set of features has to be included in a configuration. Given a feature f_1 and a set of sub-features $\{f_{11}, f_{12}, .., f_{1n}\}$, the alternative relationship can be formalized as follows: $f_1 = true \leftrightarrow ((f_{11} = true \wedge f_{12} = false \wedge ..f_{1n} = false) \vee (f_{12} = true \wedge f_{11} = false \wedge ..f_{1n} = false) \vee ...))$. An example of an *alternative* relationship depicted in Figure 1 are the two *participation subtypes* (*invitation & login*, *no login needed*).

Or. This type of relationship specifies an "or" semantics, i.e., at least one of a given set of alternative features has to be included in a configuration. Given a feature f_1 and a set of sub-features $\{f_{11}, f_{12}, .., f_{1n}\}$, the *or* relationship can be formalized as follows: $f_1 = true \leftrightarrow f_{11} = true \vee f_{12} = true \vee ..f_{1n} = true$. An example of an *or* relationship depicted in Figure 1 is the inclusion of *notification* mechanisms, i.e., the sub-features of the *notification* feature.

Requires. This type of relationship specifies a "requires" semantics, i.e., the inclusion of a feature f_1 requires the inclusion of a feature f_2 ($f_1 \rightarrow f_2$). An example of a *requires* relationship depicted in Figure 1 is the needed inclusion of *emails* and *postings* as a precondition for the feature *new postings* notification. Requires relationships are regarded as one type of *cross-tree constraint.*

Excludes. Such a relationship specifies an "incompatibility" semantics, i.e., the inclusion of a feature f_1 excludes a feature f_2 (and vice-versa). On a logical level, this property can be specified as $\neg(f_1 \wedge f_2)$. An example of an *excludes* relationship depicted in Figure 1 is the incompatibility of the *interactive agenda* service and scenarios where *no login* is needed. Similar to *requires* relationships, *excludes* constraints are regarded as a type of *cross-tree constraint.*

Customer requirements ($CREQ$) are user preferences that should be taken into account by a constraint solver when searching for a solution. In

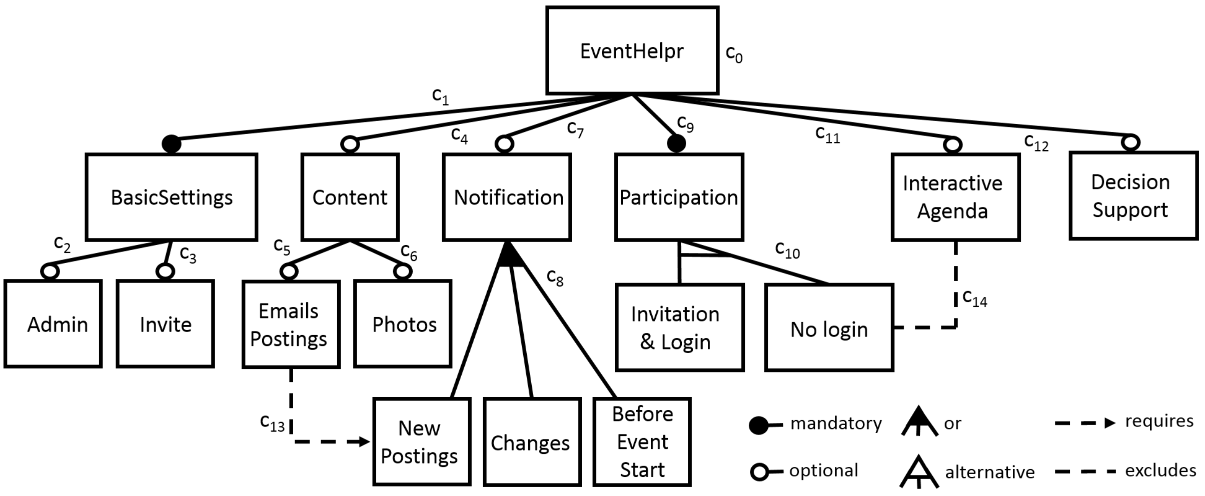

Figure 1: Variability model of a real-world event management environment (www.eventhelpr.com). The different variants describe ways in which the application can be presented to end-users (dashed arrows specify requires *and dashed lines* excludes *constraints).*

other words, $CREQ$ specifies those features that should be included in a feature configuration from the user point of view.

An example of parts of a feature model of the EventHelpr environment is depicted in Figure 1. The constraint-based representation (Tsang 1993) that can be derived thereof is the following.[2]

- $F = \{$*eventhelpr (eh), basicsettings (bas), admin (adm), invite (inv), content (con), emailspostings (email), photos (phot), notification (not), newpostings (newpost), changes (ch), beforeeventstart (bef), participation (part), invitationlogin (invit), nologin (nolog), interactiveagenda (intag), decisionsupport (dec)*$\}$.
- $D = \Cup_{f_i \in F} dom(f_i)$.
- $FM = \{c_0 : eh = true,\ c_1 : eh \leftrightarrow bas,\ c_2 : adm \rightarrow bas,\ c_3 : inv \rightarrow bas, c_4 : con \rightarrow eh, c_5 : email \rightarrow con, c_6 : phot \rightarrow con, c_7 : not \rightarrow eh, c_8 : not \leftrightarrow newpost \vee ch \vee bef, c_9 : part \leftrightarrow eh, c_{10} : part \leftrightarrow (invit \wedge \neg nolog \vee \neg invit \wedge nolog), c_{11} : intag \rightarrow eh, c_{12} : dec \rightarrow eh, c_{13} : email \rightarrow newpost, c_{14} : \neg(intag \wedge nolog)\}$.

[2] The names in brackets are introduced to increase the readability of constraints in FM.

A solution (configuration) for the feature model[3] depicted in Figure 1 is: $\{eh = true, bas = true,\ adm = true, inv = false, con = true,\ email = false,\ phot = true,\ not = true,\ newpost = false,\ ch = true,\ bef = false,\ part = true,\ invit = false,\ nolog = true,\ intag = false,\ dec = false\}$.

3 Resolving Inconsistencies

In this section, we discuss undesirable properties of feature models which trigger an unintended behavior in one way or another. We explain such properties in terms of *inconsistencies* on the logical level. Inconsistent models include conflicts (Junker 2004) that are induced *internally* by model constraints (in FM) or *externally*, for example, by test cases. Inconsistencies can be explained on the basis of *minimal conflict sets* (Junker 2004) (see Definition 3).

Definition 3 (Conflict Set). A *conflict set* $CS \subseteq FM$ is a set of constraints s.t. inconsistent(CS). CS is *minimal* if $\neg\exists CS'$: conflict set (CS').

Minimal conflict sets help to easily resolve conflicts since the deletion of at least one element

[3] Variable assignments are abbreviated, for example, $eh = true$ represents $val(eh) = true$.

from the set guarantees that the conflict is already resolved. A set that contains all elements needed to delete at least one element from each existing conflict is denoted as *hitting set* (Reiter 1987). Conflict resolution can be performed on the basis of the concepts of *model-based diagnosis* (Reiter 1987). Following the standard diagnosis approach introduced by (Reiter 1987), *Hitting Set Directed Acyclic Graphs* (*HSDAGs*) have to be constructed where individual paths of the tree represent hitting sets (diagnoses). A diagnosis task can be defined as follows (see Definition 4). For an overview of different conflict detection and related diagnosis algorithms we refer, for example, to (Felfernig et al. 2014).

Definition 4 (Diagnosis). A *diagnosis* $\Delta \subseteq FM$ is a set of constraints s.t. consistent($FM - \Delta$). Δ is *minimal* if $\neg\exists\Delta' \subset \Delta$: diagnosis ($\Delta'$).

Typically, we are interested in analyzing specific parts of C, i.e., either FM if we are interested in faulty constraints part of the feature model, or $CREQ$ if we are interested in (minimal sets of) customer requirements that induce an inconsistency. Conflict sets and diagnoses are then determined from the individual parts of C, i.e., CS is then either a subset of FM or $CREQ$, the same holds for corresponding diagnoses Δ.

In the following, we focus on the first aspect, i.e., constraints in the feature model responsible for an inconsistency (as mentioned in Defintions 3 and 4, conflict sets and diagnoses are composed of constraints in FM).[4]

4 Inconsistencies in Feature Models

In the following, we discuss feature model properties that make a feature model ill-formed. For each property, we characterize the underlying inconsistency on a formal level and show ways to resolve the inconsistency (see also Table 1). Thus, we try to explain ill-formed model properties in terms of inconsistencies between intended and unintended feature model properties.

[4] Details on diagnosing $CREQ$ can be found in (Felfernig et al. 2004).

Void feature models. A *void* feature model is inconsistent per-se, i.e., no solution can be identified for a given feature configuration task (F, D, FM). A diagnosis Δ for a void feature model FM is defined as $\Delta \subseteq FM : consistent(FM - \Delta)$.

Test-induced void feature models. Feature models can be tested on the basis of a test suite $T = \{t_1, t_2, .., t_m\}$ consisting of positive test cases[5] specifying the intended behavior of a knowledge base. A feature model is consistent with T if $\forall t_j \in T : FM \cup \{t_j\}$ is consistent. If some of the test cases induce conflicts (CS_k), these have to be resolved on the basis of the concepts of model-based diagnosis (Reiter 1987). A diagnosis Δ for a test-induced void feature model is defined as $\Delta \subseteq FM$ such that $\forall t_j \in T : consistent(\{t_j\} \cup FM - \Delta)$.

Dead features. A feature $f \in F$ is regarded as a dead one if it is not part of any of the theoretically possible solutions. More formally, $\{f = true\} \cup FM$ is inconsistent. A diagnosis Δ for a dead feature is defined as consistent($FM - \Delta \cup \{f = true\}$).

Fully mandatory features. A feature f is fully mandatory, if it is included in every possible configuration, i.e., inconsistent($\{f = false\} \cup FM$). If we want to allow configurations with f not included ($f = false$), a diagnosis Δ for a fully mandatory feature is defined as consistent($FM - \Delta \cup \{f = false\}$).

False optional features. A feature f_2 is *false optional*, if it is modeled as optional with regard to a parent feature f_1 but in fact is contained in every configuration f_1 is included. A diagnosis Δ for a false optional feature is defined as consistent($FM - \Delta \cup \{f_2 = false\} \cup \{f_1 = true\}$), i.e., there exists at least one configuration with $f_2 = false$ and $f_1 = true$.

Redundant constraints in FM. A constraint $c_i \in FM$ is regarded as redundant if $FM - \{c_i\} \models c_i$, i.e., c_i logically follows from $FM - \{c_i\}$. Consequently, deleting c_i from FM preserves the semantics of the feature model. In other

[5] For a discussion of the handling of negative test cases we refer to (Felfernig et al. 2004).

words, the solution space of the feature model remains exactly the same. If $FM = \{c_1, .., c_n\}$ is a set of non-redundant constraints of a feature model, and $\widehat{FM}$ is the negation of FM (i.e., $\widehat{FM} = \{\neg c_1 \vee \neg c_2 \vee ... \vee \neg c_n\}$, then $FM \cup \widehat{FM}$ is inconsistent. As a consequence, a redundant constraint set FM_r can be made non-redundant by determining a minimal set of constraints $FM \subseteq FM_r : inconsistent(FM \cup \widehat{FM})$. This way, conflict detection algorithms can be applied to make knowledge bases non-redundant.

Implicit feature groups. If two features have exactly the same value in each possible configuration, there is a strong dependency between these features. In some cases, these features can even be combined to reduce the number of needed constraints and thus increase model understandability and maintainability. Two features f_1 and f_2 form an implicit group if inconsistent($\{f_1 \wedge \neg f_2 \vee \neg f_1 \wedge f_2\} \cup FM$), i.e., there does not exist a solution in which the two features have different values.

5 Inconsistencies in Example Model

In order to exemplify the discussed model properties, we introduce an adapted version of the feature model shown in Section 2 (see Figure 2). The constraint-based representation (Tsang 1993) that can be derived thereof, is the following.

- $F = \{$*eventhelpr (eh), basicsettings (bas), admin (adm), invite (inv), content (con), emailspostings (email), photos (phot), notification (not), newpostings (newpost), changes (ch), beforeeventstart (bef), participation (part), invitationlogin (invit), nologin (nolog), interactiveagenda (intag), decisionsupport (dec)*$\}$.
- $D = \Cup_{f_i \in F} dom(f_i)$.
- $FM = \{c_0 : eh = true, c_1 : eh \leftrightarrow bas, c_2 : adm \rightarrow bas, c_3 : inv \rightarrow bas, c_4 : con \leftrightarrow eh, c_5 : email \leftrightarrow con, c_6 : phot \rightarrow con, c_7 : not \rightarrow eh, c_8 : not \leftrightarrow newpost \vee ch \vee bef, c_9 : part \leftrightarrow eh, c_{10} : part \leftrightarrow (invit \wedge \neg noglog \vee \neg invit \wedge nolog), c_{11} : intag \leftrightarrow eh, c_{12} : dec \leftrightarrow eh, c_{13} : email \rightarrow newpost, c_{14} : \neg(intag \wedge nolog), c_{15} : dec \rightarrow nolog\}$.

Void feature model. No solution exists for the feature model defined in Figure 2 since both features, *interactiveagenda (intag)* and *decisionsupport (dec)* are mandatory, *dec* requires *nolog* and *intag* excludes *nolog*. The (singleton) resulting minimal conflict set is $CS_1 : \{c_0, c_{11}, c_{12}, c_{14}, c_{15}\}$. In this example, only one minimal conflict set exists and the deletion of any of these constraints restores consistency, i.e., each individual constraint represents a diagnosis Δ (e.g., $\Delta = \{c_{14}\}$). We want to emphasize that constraint c_0 has a specific role: if it is set to *true*, it is guaranteed that only non-empty feature configurations, i.e., feature configurations with at lest one activated feature, are taken into account. For this reason, c_0 has a special role and is in most of the cases not considered a diagnosis candidate.

Test-induced void feature model. In the following, we assume that constraint c_{14} in the feature model of Figure 2 has been deleted to restore consistency. An example of a test-induced void feature model is the model depicted in Figure 1 combined with the test-case $t_1 : intag \wedge nolog$, i.e., inconsistent($\{t_1\} \cup FM$). The minimal conflict set is $CS : \{c_{14}\}$ since inconsistent($\{t_1\} \cup \{c_{14}\}$).

Dead features. In our example feature model of Figure 2, feature *invitationlogin (invit)* can be considered as dead since it is not possible to include this feature in a configuration. The reason is that feature *decisionsupport (dec)* requires the inclusion of feature *nolog*. Due to the *alternative* relationship between *participation (part)*, *inv*, and *nolog*, it is not possible to include feature *inv*.

False optional feature. Since the inclusion of *emailspostings (email)* requires the inclusion of *newpostings (newpost)*, *notification (not)* will be included in every configuration. For this reason, $notification$ can be regarded as *false optional*.

Redundant constraint. Constraint c_9 can be regarded as redundant (assuming that constraint c_{14} has been deleted) since feature *participation* is part of every configuration (it is required by feature *decisionsupport*).

property	property check	analysis operation
void feature model	inconsistent(FM)	$\Delta \subseteq FM : consistent(FM - \Delta)$
test-induced void feature model	$\exists t_i \in T : inconsistent(FM \cup \{t_i\})$	$\Delta \subseteq FM, \forall t_i \in T : consistent(FM - \Delta \cup \{t_i\})$
dead feature f_i	$inconsistent(\{f_i = true\} \cup FM)$	$\Delta \subseteq FM : consistent(FM - \Delta \cup \{f_i = true\})$
full mandatory feature f_i	$inconsistent(\{f_i = false\} \cup FM)$	$\Delta \subseteq FM : consistent(FM - \Delta \cup \{f_i = false\})$
false optional feature f_i	$inconsistent(\{f_i = false \wedge f_{i-1} = true\} \cup FM)$	$\Delta \subseteq FM : consistent(FM - \Delta \cup \{f_i = false \wedge f_{i-1} = true\})$
redundant constraints $c_i \in FM$	$\exists c_i \in FM : FM - \{c_i\} \models c_i$	$FM_{nr} \subseteq FM : \forall c_i \in FM_{nr} : FM_{nr} - \{c_i\} \not\models c_i$
implicit feature groups $\{f_a, f_b\} \in FM$	$\exists \{f_a, f_b\} \subseteq F : inconsistent(\{f_a = true \wedge f_b = false \vee f_a = false \wedge f_b = true\} \cup FM)$	$\Delta \subseteq FM : consistent(FM - \Delta \cup \{f_a = true \wedge f_b = false \vee f_a = false \wedge f_b = true\})$

Table 1: Summary of analysis operations for feature models. FM_{nr} denotes a non-redundant feature model.

6 Research Issues

There are a couple of open issues for future research directly related to the detection and resolution of inconsistencies in feature models.

Personalized Conflict Detection and Resolution for Feature Modeling. When developing feature models, inconsistencies in models can be resolved in different ways (in many cases, there exist different conflict sets). A task in this context is to propose conflict resolutions that are relevant, i.e., will be accepted by engineers. When developing knowledge bases, this means to figure out relevant faulty constraints where preferences can be derived from the development history of the feature model (e.g., how often a constraint has been changed in the last year, how often a constraint has been activated when determining a solution etc.). Initial related work can be found, for example, in (Felfernig et al. 2015).

Anytime Conflict Detection. There is a tradeoff between efficiency and accuracy that has to be taken into account when applying conflict detection algorithms (Felfernig et al. 2018). A challenge in this context is to identify conflicts of relevance for the user within time limits acceptable for interactive settings. Anytime conflict detection and diagnosis algorithms are an area of research that have to be investigated and further developed in the context of feature model development.

Cognitive Effects. An important question to answer is how knowledge engineers and domain experts without computer science background interpret the semantics of models (Fliedl et al. 2000; Michael and Mayr 1998). A related question is how easy it is for them to understand existing feature models and in which way we have to define natural language statements that represent constraints in feature models. In a typical feature model development process, constraints and feature hierarchies are defined by domain experts and then formalized by knowledge engineers. In this context, it has to be assured that knowledge engineers understand (textually defined) domain constraints to reduce the number of interaction cycles between domain experts and knowledge engineers (Fliedl et al. 2007; Kop and Mayr 1998; Shekhovtsov et al. 2014).

Model Development Practices. A major issue in the context of feature model development and maintenance is how to structure the domain knowledge in such a way that changing chunks of knowledge can be easily managed. Maintainability of variability models is still an open research issue and especially in the context of large and

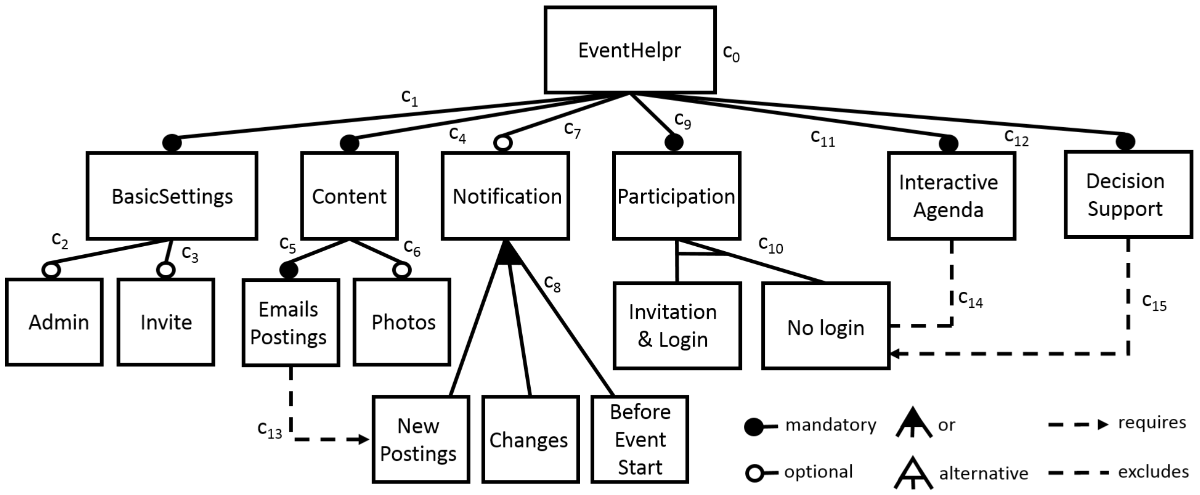

Figure 2: Example of a faulty feature model. Dashed arrows specify requires *and dashed lines* excludes *constraints.*

complex models, structuring mechanisms have to be developed that help to achieve the goal of easy maintenance. There is a branch of research dealing with collecting knowledge from the crowd and synthesizing configuration knowledge thereof (Ulz et al. 2016). This is also a promising area for the development of feature models.

Merging of Feature Models. In global contexts, for example, a car company selling cars in countries with different legal contexts, feature models have to take into account contextual information (sales strategies can differ and also country-specific legal aspects have to be taken into account). To analyze (e.g., in how many countries is it possible to sell feature x?) and adapt the underlying feature models, country-individual models have to be integrated. Existing research in the area of model integration (Liberatore and Schaerf 1998) has to be analyzed with regard to applicability in the context of variability modeling.

7 Conclusions

Requirements Engineering (RE) is a critical phase in software development processes. In this article, we focused on software variability modeling scenarios where features are used to specify the inclusion or exclusion of specific software functionalities. With increasing size and complexity of those models, the probability of including inconsistencies increases. We provided an overview of different types of inconsistencies and showed how to automatically detect these inconsistencies on the basis of the concepts of conflict detection and model-based diagnosis.

References

Bakker R., Dikker F., Tempelman F., Wogmim P. (1993) Diagnosing and Solving Over-determined Constraint Satisfaction Problems. In: 13th International Joint Conference on Artificial Intelligence. Chambery, France, pp. 276–281

Batory D. (2005) Feature Models, Grammars, and Propositional Formulas. In: Software Product Lines Conference. Springer Lecture Notes in Computer Science Vol. 3714, pp. 7–20

Benavides D., Felfernig A., Galindo J., Reinfrank F. (2013) Automated Analysis in Feature Modelling and Product Configuration. In: 13th International Conference on Software Reuse. Lecture Notes in Computer Science 7925. Springer, Pisa, Italy, pp. 160–175

Benavides D., Segura S., Ruiz-Cortes A. (2010) Automated Analysis of Feature Models 20 years Later: a Literature Review. In: Information Systems 35(6), pp. 615–636

Czarnecki K., Helsen S., Eisenecker U. (2005) Formalizing Cardinality-based Feature Models and their Specialization. In: SoftwareProcess: Improvement and Practice 10(1), pp. 7–29

Felfernig A., Friedrich G., Jannach D., Stumptner M. (2004) Consistency-based Diagnosis of Configuration Knowledge Bases. In: Artificial Intelligence 152(2), pp. 213–234

Felfernig A., Hotz L., Bagley C., Tiihonen J. (2014) Knowledge-based Configuration: From Research to Business Cases, 1st. Elsevier/Morgan Kaufmann

Felfernig A., Reiterer S., Stettinger M., Tiihonen J. (2015) Intelligent Techniques for Configuration Knowledge Evolution. In: Vamos 2015 Workshop. Hildesheim, Germany, pp. 51–60

Felfernig A., Walter R., Galindo J., Benavides D., Atas M., Polat-Erdeniz S., Reiterer S. (2018) Anytime Diagnosis for Reconfiguration. In: Journal of Intelligent Information Systems (JIIS)

Fliedl G., Kop C., Mayr H., Mayerthaler W., Winkler C. (2000) Linguistically based requirements engineering - The NIBA-project. In: Data & Knowledge Engineering 35 (2), pp. 111–120

Fliedl G., Kop C., Mayr H., Salbrechter A., Vöhringer J., Weber G., Winkler C. (2007) Deriving static and dynamic concepts from software requirements using sophisticated tagging. In: Data & Knowledge Engineering 61 (3), pp. 433–448

Junker U. (2004) QuickXPlain: Preferred Explanations and Relaxations for Over-constrained problems. In: 19th Intl. Conference on Artifical Intelligence (AAAI'04). AAAI Press, San Jose, California, pp. 167–172

Kang K., Cohen S., Hess J., Novak W., Peterson S. (1990) Feature-oriented Domain Analysis (FODA) – Feasibility Study. In: Technical Report, CMU-SEI-90-TR-21

Kop C., Mayr H. (1998) Conceptual predesign bridging the gap between requirements and conceptual design. In: 3rd International Conference on Requirements Engineering. Colorado Springs, CO, USA, pp. 90–98

Leffingwell D., Widrig D. (2003) Managing Software Requirements: A Use Case Approach, 2nd. Addison-Wesley

Liberatore P., Schaerf M. (1998) Arbitration (or how to merge knowledge bases). In: IEEE Transactions on Knowledge and Data Engineering 10 (1), pp. 76–90

Mayr H., Kop C. (2002) A User Centered Approach to Requirements Modeling. In: Modellierung 2002, pp. 75–86

Mayr H., Kop C., Esberger D. (2007) Business Process Modeling and Requirements Modeling. In: International Conference on the Digital Society (ICDS 2007). Guadeloupe, French Caribbean, p. 8

Michael J., Mayr H. (1998) Intuitive understanding of a modeling language. In: Australasian Computer Science Week Multiconference (ACSW 2017). Geelong, Australia, 35:1–35:10

Reiter R. (1987) A Theory of Diagnosis From First Principles. In: Artificial Intelligence 32(1), pp. 57–95

Shekhovtsov V., Mayr H., Kop C. (2014) Facilitating effective stakeholder communication in software development processes. In: Forum at the Conference on Advanced Information Systems Engineering (CAiSE), pp. 116–132

Thum T., Batory D., Kastne C. (2009) Reasoning about edits to feature models. In: 31st IEEE International Conference on Software Engineering. IEEE, pp. 254–264

Tsang E. (1993) Foundations of Constraint Satisfaction. Academic Press, London, San Diego, New York

Ulz T., Schwarz M., Felfernig A., Haas S., Reiterer S., Stettinger M. (2016) Human Computation for Constraint-based Recommenders. In: Journal of Intelligent Information Systems (JIIS)

White J., Benavides D., Schmidt D., Trinidad P., Dougherty B., Ruiz-Cortez A. (2010) Automated Diagnosis of Feature Model Configurations. In: Systems and Software 83(7), pp. 1094–1107