Explanations for Agile Feature Models

Pablo Trinidad   David Benavides   Antonio Ruiz-Cortés   Sergio Segura

Miguel Toro

Dpto. Lenguajes y Sistemas Informáticos

University of Seville, Av. de la Reina Mercedes S/N, 41012 Seville, Spain

{trinidad,benavides,aruiz,segura,mtoro}@tdg.lsi.us.es

Abstract

Feature models are widely used to represent product lines and they are key parts of the development process in agile methodologies such as generative and feature-oriented programming. One of the most common problems in these cases is checking a feature model to be error-free. Some works cope with detecting errors in feature models, but automated explaination of errors is demanded in order to help the modeler to solve them. In this paper we propose a implementation-independent solution to explain the reasons why dead features, a common kind of error, appear in feature models, giving an implementation example.

1   Introduction

Agile methodologies follow a rapid development of software giving attention to coding and reducing the amount of documentation produced. In the other hand, SPL development pursues reducing the products time-to-market and cost by analyzing the commonalities and variabilities within a set of products in a concrete domain. Although both share the objective of reducing the costs and time-to-market, their methodologies are very different, but may be complemented. SPL development may be agile, it means that reducing the amount of documentation and process definition is also possible in this context. To make it possible, the users should have at their disposal a set of development tools that help them with rapidly developing SPL.

In this context, some tools [5, 16] have been proposed to support the development of SPL focusing on feature models as the core of the process. Feature models have been one of the most important proposals in the context of software product lines (SPL). They are used to describe SPL in terms of features, it means visible characteristics of the products. Feature models are a key piece in generative programming and feature oriented programming [3], two techniques that pursue speeding up the SPL development pro-

cess. The treatment of errors in feature models is a matter that many authors have demanded [1, 2, 10, 18]. However these tools do not face up to an exhaustive analysis of the kinds of error to be made in feature modeling and they only deal with the detection of errors[5, 17, 18], but are not able to explain the origin of the errors neither to give solutions to them in order to help the modeler obtaining error-free models.

It is very important to define the complete process to deal with errors in feature models as a first step, and afterwards developing a set of tools that holds the agile development of SPL, where changes on requirements are supported by them and where feature models are permanently checked to be error-free.

In a previous work [14], we dealt with the detection of dead features, a frequent case of error in feature models. In this paper we define how to explain the possible reasons why a dead feature appears as a second step in their treatment, following this structure: in Section 2, we briefly describe the concept of dead features. In Section 3, we propose a general solution to explain dead features appearance. In Section 4, we implement the explanation, giving a running example of it in Section 5. Last, we summarize the works on errors treatment in feature models in Section 6 and some conclusions and guidelines of our future work are given in Section 7.

2   Dead Features

A feature model is a tree-like structure consisting of a set of features linked by hierarchical relations and optionally, cross-tree constraints. Feature models capture the set of products of a product line in terms of features. They have a simple syntax that may be checked automatically with few effort. However, it must be checked that a feature model captures the correct set of products in the product line, and it is not a trivial task.

In Figure 1, a feature model representing the set of products within a Home Integration System (HIS) product line is
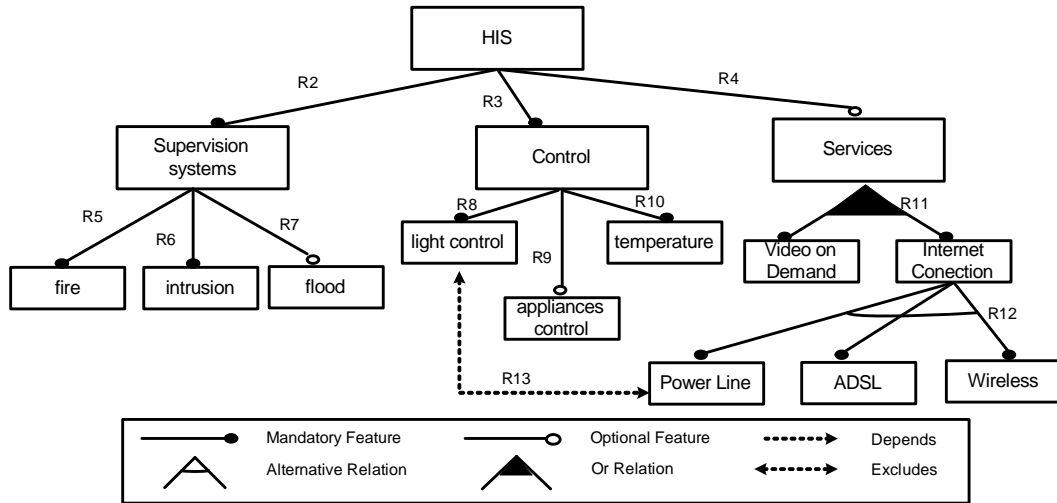
Figure 1. Home Integration System(HIS) feature model

depicted. Among all the relations, an special attention must be paid to the mutual exclusion between *Power Line* and *Light control* features. It means that there could be no product with these two features at the same time. *Light control* is a full-mandatory feature, so it appears in every product. Although, *Power Line* feature is represented as an alternative of *Internet Connection*, no product can contain it because it is incompatible with *Light Control*. *Power Line* is a dead feature [5], and is a common case of error in feature models. We define the concept of dead feature as:

Dead Feature A feature that does not appear in any of the products in the family.

In Figure 2, some cases that cause the appearance of dead features are exposed [17]. The treatment of dead features or any other kind of error just starts with detection but the final purpose is producing error-free feature models. This is the reason why in [14] we proposed a 3-steps process that deals with errors in feature models in general:

1. The detection of the set of errors in a feature model.

2. The detection of the relationship/s that cause each error to appear.

3. Giving possible solutions to solve each error.

Taking Figure 1 feature model as example, these are results of applying this process (Figure 3):

1. *Power Line* is a dead feature.

2. These are the relationships that may cause *Power Line* to be a dead feature: excludes relationship and/or *Light control* mandatory relationship.

3. Some possible solutions can be given: removing excludes relationship; *Light Control* being an option of *Control* feature; or removing *Power Line* feature from alternative.

In this paper we deal with the second step in the process, detecting the relationships that may cause a dead feature to appear, supposing that we have already detected them in a previous step as shown in [14].

3 Diagnosing Feature Models

3.1 Theory of Diagnosis

In [6, 13], Reiter proposed a way to diagnose systems, it means, determining the set of components that are functioning abnormally and makes the system present a different behaviour than expected. It has been widely used in detecting the components failing in circuits, and we are applying it to feature models. To deal with it, some concepts are defined:

System System is a pair $(SD, COMPS)$ where $COMPS$ is the set of components that composes that system and SD the set of default logic (a logic that generalizes first-order logic) sentences [12] that describes the system, it means the relations among components.

Abnormal predicate How to detect if a component is abnormal or not should be described in SD using abnormal predicates $AB(c)$ that identifies that component $c \in COMPS$ works abnormally.

Observation A set OBS of default logic sentences describing an observation of the behaviour or state of the system. From an observation, it can be evaluated the
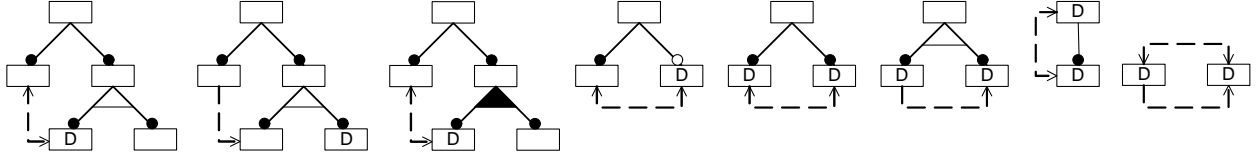
Figure 2. Typical situations that generate dead features. Dead features are depicted by D
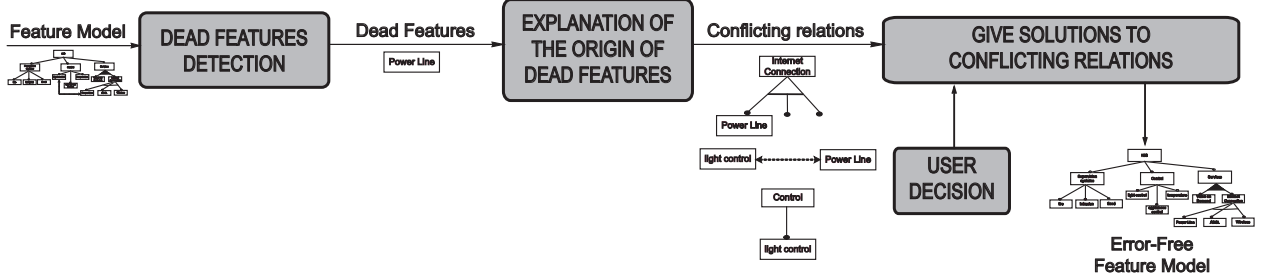

Figure 3. 3-Steps dead features treatment

set of components that may be faulty in a system. We shall write $(SD, COMPS, OBS)$ to represent the observation OBS of the system $(SD, COMPS)$.

Suppose we have a system $(SD, \{c_1, \cdots, c_n\})$ that we want to evaluate if it is faulty or not. $SD \cup \{\neg AB(c_1), \cdots, \neg AB(c_n)\}$ is the set of predicates that represents the system behaviour assuming that all the components work correctly. An observation OBS of the system can detect an abnormal behaviour if

$$SD \cup \{\neg AB(c_1), \cdots, \neg AB(c_n)\} \cup OBS$$

is inconsistent. The objective of diagnosis is finding a set of components that are faulty, considering that the rest of components work properly.

Diagnosis A diagnosis $\mathcal{D}(\Delta, COMPS - \Delta)$ for an observation of a system $(SD, COMPS, OBS)$ is a set $\Delta \subseteq COMPS$ such that

$$\begin{aligned} SD \cup OBS \quad & \cup \quad \{AB(c) \mid c \in \Delta\} \\ & \cup \quad \{\neg AB(c) \mid c \in COMPS - \Delta\} \end{aligned}$$

But for an observation, there are many possible diagnosis or sets of components that explained a faulty behaviour, so the concept of minimal diagnosis is introduced:

Minimal Diagnosis A diagnosis $\mathcal{D}(\Delta, COMPS - \Delta)$ is minimal iff for no subset $\Delta' \subseteq \Delta$ is $\mathcal{D}(\Delta', COMPS - \Delta')$ a diagnosis.

Therefore, a minimal diagnosis can be understood as a minimal set of faulty components that explains the abnormal behaviour of the system.

Theory of dianosis offers a framework where the problems of finding the abnormal components of systems can be expressed independently from the implementation. The advantage of representing problems using theory of diagnosis is that there are many algorithms and techniques to diagnose systems, and any of them can be used in implementation.

3.2 Applying Theory of Diagnosis to Feature Models

Our purpose is to evaluate the possible reasons why a feature model has dead features and how the theory of diagnosis offers a background to evaluate feature models. The first step to take leverage from theory of diagnosis is identifying all the elements that it provides in our problem.

The system we are evaluating is a feature model and it is composed by features (F) and relations (R), it means those are the components in $COMPS$. The system description SD characterizes the kind of relations between features using default order logic. A possible mapping of feature models onto first order logic can be found at [10]. As stated in previous section, abnormal predicate must be used when defining the system. In this case, we want to find the faulty relations, not faulty features. Therefore, for each sentece that represents a relation $r \in R$, $AB(r)$ predicate must be used in SD.

Let $F = \{f_1, \cdots, f_n\}$ be the set of features in a feature model, and $R = \{r_1, \cdots, r_m\}$ the set of sentences that represents the relations among features. To build SD, we define for each element $r_i \in R$, an expression r_i' such as follows:

$$r_i' \equiv XOR(r_i, AB(r_i))$$

It means that, if a relation represented by r_i sentence is consistent, that relation is not faulty, else the relation is faulty and abnormal predicate is defined.

To be able to diagnose the system, an observation is needed. In this case we part from a list of dead features. We want to evaluate for each dead feature, which are the relations that make it impossible to appear in any product. Then the observation of the system forces that dead feature to appear. This observation will cause $SD \cup \{\neg AB(r_1), \cdots, \neg AB(r_n)\} \cup OBS$ to be inconsistent, because the definition of the system avoids the dead feature to appear on it.

Let $f_d \in F$ be a dead feature previously detected. We define the detection of the relations that cause a dead feature to appear as a diagnosis problem this way:

$$SD = \{r'_1, \cdots, r'_m\}$$
$$COMPS = F \cup R$$
$$OBS = \{f^d\}$$

The objective is finding the minimal diagnosis $\mathcal{D}(\Delta, COMPS - \Delta)$ to the observation of the feature model such that $\Delta \subseteq COMPS$. As you may notice, no abnormal predicate is defined for the features in $COMPS$, so $F \cap \Delta = \emptyset$. Therefore, all the components in Δ are relations, it means that the minimal diagnosis of the observation of the system will be the minimal set of relations that cause feature f^d to be dead.

The theory of diagnosis is intended to be a framework where different diagnosis techniques and implementations can take part of it. We can use any of the implementations that deal with finding the set of minimal diagnoses given a system observation. Many authors have expressed diagnosis problems, a.k.a. explanations problems, as Constraint Satisfaction Problems (CSP), propositional satisfiability problems (SAT) or description logic (DL).

There are some explanations algorithms [7, 8, 9] independent of the usage of CSP, SAT or DL which may be used to diagnose feature models. Next section we are proposing a simple implementation that defines the diagnosis problem as a CSP. However, it must be taken into account that the description above permits the use of any of the mentioned approaches.

4 Implementation: Explanations in CSP

A Constraint Satisfaction Problems (CSP) [15] represented by ((V,D),C) is defined as a set of variables (V), each ranging on a finite domain (D), and a set of constraints C restricting all the values that variables can take simultaneously. A solution to a CSP is an assignment of a value from its domain to every variable, in such a way that all constraints are satisfied simultaneously. In the common usage of CSP, we may want to find all the solutions or one solution, following or not an optimization criterion.

In [4], a mapping of feature models onto CSP is proposed, allowing to realize different operations for extracting relevant information and reasoning on them. CSP is shown as a flexible technique that permits the user to extend the questions and operations to be performed very easily.

The diagnosis problem may be defined as a CSP. The components in $COMPS$ are the variables of the CSP. Then, the variables are features ($F = \{f_1, \cdots, f_n\}$) and one sensor for each relation ($S = \{s_1, \cdots, s_m\}$), and both of them take values from the $\{0, 1\}$ domain. The system description SD is depicted by the set of constraints c'_j that are defined this way:

$$\forall i \cdot 1 \leq i \leq m$$
$$c'_i \equiv c_i \vee s_i = 0$$

where c_i is the constraint resulting from mapping a relation onto a constraint [4].

Sensors are used to detect when a constraint can or cannot be satisfied. If a constraint c_i cannot be satisfied, its sensor s_i is forced to be zero. If the constraint can be satisfied, its sensor takes value from $\{0, 1\}$.

Last, the observation OBS must be added to the CSP as part of the constraints set without any linked sensor. We propose implementing the problem as a Constraints Optimization Problem (COP) which is just a particular case of CSP with an optimization criterion O. Our objective is obtaining the maximum number of constraints satisfied and therefore the minimal set of non-satisfied constraints, so we use the following optimization criterion:

$$max \sum_{i=1}^{m} s_i$$

Summarizing, the COP $((V, D), C, O)$ that represents our diagnosis problem may be depicted this way:

$$(V, D) = (F \cup S) \times \{0, 1\}$$
$$C = \{c'_1, \cdots, c'_m\} \cup \{f^d = 1\}$$
$$O = \sum_{i=1}^{m} s_i$$

Solving the COP will generate a set of solutions with the maximum numbers of constraints holding, it means with the minimal number of sensors being zero. One or more solutions can be obtained, but all of them will have the same minimal number of zero-sensors. Determining the relations

System Description			
Relations	**Original Constraint**	**Sensor**	**Constraint**
Root	$c_1 \equiv HIS = 1$	s_1	$c'_1 \equiv c_1 \vee s_1 = 0$
R2	$c_2 \equiv SUPERVISION = HIS$	s_2	$c'_2 \equiv c_2 \vee s_2 = 0$
R3	$c_3 \equiv CONTROL = HIS$	s_3	$c'_3 \equiv c_3 \vee s_3 = 0$
R4	$c_4 \equiv SERVICES \Rightarrow HIS$	s_4	$c'_4 \equiv c_4 \vee s_4 = 0$
R5	$c_5 \equiv FIRE = SUPERVISION$	s_5	$c'_5 \equiv c_5 \vee s_5 = 0$
R6	$c_6 \equiv INTRUSION = SUPERVISION$	s_6	$c'_6 \equiv c_6 \vee s_6 = 0$
R7	$c_7 \equiv FLOOD \Rightarrow SUPERVISION$	s_7	$c'_7 \equiv c_7 \vee s_7 = 0$
R8	$c_8 \equiv LIGHT = CONTROL$	s_8	$c'_8 \equiv c_8 \vee s_8 = 0$
R9	$c_9 \equiv APP \Rightarrow CONTROL$	s_9	$c'_9 \equiv c_9 \vee s_9 = 0$
R10	$c_{10} \equiv TEMPERATURE = CONTROL$	s_{10}	$c'_{10} \equiv c_{10} \vee s_{10} = 0$
R11	$c_{11} \equiv (VIDEO \vee INTERNET) \Leftrightarrow SERVICES$	s_{11}	$c'_{11} \equiv c_{11} \vee s_{11} = 0$
R12	$c_{12} \equiv POWER \Leftrightarrow (INTERNET \wedge \neg ADSL \wedge \neg WIFI) \wedge$ $ADSL \Leftrightarrow (INTERNET \wedge \neg POWER \wedge \neg WIFI) \wedge$ $WIFI \Leftrightarrow (INTERNET \wedge \neg ADSL \wedge \neg POWER)$	s_{12}	$c'_{12} \equiv c_{12} \vee s_{12} = 0$
R13	$c_{13} \equiv LIGHT = 1 \Rightarrow POWER = 0$	s_{13}	$c'_{13} \equiv c_{13} \vee s_{13} = 0$
Observation			
Dead Feature	$POWER = 1$	–	$POWER = 1$
Components / Variables			
$HIS, SUPERVISION, CONTROL, SERVICES, FIRE, INTRUSION, FLOOD, LIGHT$ $APP, TEMPERATURE, VIDEO, INTERNET, POWER, ADSL, WIFI,$ $s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, s_9, s_{10}, s_{11}, s_{12}, s_{13}$			

Table 1. Mapping sample feature model onto CSP constraints and variables

that are faulting is just a question of mapping the zero-sensors to their correspondent conflictive relationships, it means, for any $s_i = 0$, finding its linked constraint c_i.

5 A Running Example

Let us take the feature model diagram in Figure 1. We will use it as an example of how to apply the CSP implementation to explain the reasons why a dead feature appears. It is supposed that a detection technique [14] has previously been applied in order to detect that D is a dead feature.

Applying the structure shown in section 4, there is a variable for each feature in the feature model and one sensor for each relation ($S = \{s_1, \cdots, s_{13}\}$). In Table 1, the result of mapping feature model relations and adding sensors is shown. After solving the problem as a COP, two solutions (only sensors values are considered) are obtained where the only sensors with a zero value are s_8 and s_{13}. A sensor

with a zero value indicates a relation that may cause a dead feature to appear. These solutions suggest that there are two possible diagnosis affecting to the relations linked to sensors $\{\{s_8\}, \{s_{13}\}\}$. It means that the relations causing the appearance of a dead feature in this feature model are mandatory relation between $Control$ and $LightControl$ and the relation $LightControl$ requires $PowerLine$. In a further step, we are analyzing in Table 2 the results of applying this method to the examples shown in Figure 2.

6 Related Works

Characterising all the possible kinds of error and demonstrating their completeness is an important point. Von der Massen and Lichter [17] introduce the concept of deficiencies in feature models, identifying three different levels of errors:

- Redundancy: information is modeled in multiple

Feature Model	Conflicting Relations	Feature Model	Conflicting Relations
	{R1},{R4}		{R1},{R3},{R4}
	{R1},{R4}		{R3}
	{R1},{R3}		{R1},{R2}
	{R2}		{R1},{R2}

Table 2. Explanations for dead features in Figure 2

ways, being redundant. That redundancy can be removed if it has been accidentally introduced or kept to explicitly model an important relationship.

- Anomalies: senseless information is modeled and potential configurations are lost, though configurations should be possible. As a consequence, a non-mandatory feature becomes full-mandatory or can never be selected.

- Inconsistency: a model contains contradictory information, making most of the times impossible to derive any product configuration. In the best of the cases, a set of features can never be selected in configuration process.

A list of examples for each type is given, but it lacks of exhaustive and complete definitions and no automated support is given to detect them. They also defined a normalized feature model as the one that has no redundancy, anomaly or inconsistency, considering the need of the assistance of a domain expert in most of the cases to produce them.

In [4], one feature model is considered to be valid if it defines at least one product giving an automated support based on CSP. In [10], Mannion also proposes the usage of first order logic to validate feature models. Although they define and automatically support the detection of non-valid feature models, this case is only the consequence of an inconsistency that affects the root.

On the other hand, many authors have dealt with the detection of errors in configuration, it means when the user selects features from the feature model to build concrete products. Regarding to this process, [17] indicates that inconsistencies arise when a variation point is not resolved or a full-mandatory feature is missing in a concrete product. Many authors have dealt with the problem of errors detection in product configuration: D.Batory[1], proposes selecting the features one by one and checking if each selection is right or wrong by using SAT solvers; Czarnecki[5] uses Binary Decision Diagrams (BDD) to check the satisfiability of the model and propagating the decisions made in configuration; In [11], it is proposed the usage of configurators to follow the same intention.

7 Conclusions and Future Work

In this paper we have stated a general framework to explain the origin of dead features and have presented an implementation of this solution based on constraint optimization problems. For the integration of errors explanation in SPL modeling tools, it is needed a good performance that makes this task agile and efficient. One of our open issues is analyzing the best implementation to this solution regarding to performance and efficiency.

Based only on explanations, a developer may be able to manually solve dead features. However we want to go further and deal with the third step in errors treatment, automatedly giving alternatives that solves the dead features.

Another issue to address in the future is the empirical characterization of the causes why errors in feature models appear. This could lead us to defining specific algorithms that improve the performance of some steps in errors treatment.

We proposed a general process to deal not only with dead features, but with any other kind of error. It is our purpose to extend our works to detect, explain and solving other types of error.

Acknowledgments

References

[1] D. Batory. Feature models, grammars, and propositional formulas. In J. H. Obbink and K. Pohl, editors, *SPLC*, volume 3714 of *Lecture Notes in Computer Science*, pages 7–20. Springer, 2005.

[2] D. Batory, D. Benavides, and A. Ruiz-Cortés. Automated analysis of feature models: Challenges ahead. *Communications of the ACM*, Conditionally accepted, 2006.

[3] D. Batory, J. Sarvela, and A. Rauschmayer. Scaling stepwise refinement. *IEEE Trans. Software Eng.*, 30(6):355–371, 2004.

[4] D. Benavides, P. Trinidad, and A. Ruiz-Cortes. Automated reasoning on feature models. *LNCS, Advanced Information Systems Engineering: 17th International Conference, CAiSE 2005*, 3520:491–503, 2005.

[5] K. Czarnecki and P. Kim. Cardinality-based feature modeling and constraints: a progress report. In *Proceedings of International Workshop on Software Factories, OOPSLA 2005*, 2005.

[6] J. de Kleer, A. K. Mackworth, and R. Reiter. Characterizing diagnoses. In *AAAI*, pages 324–330, 1990.

[7] M. J. G. de la Banda, P. J. Stuckey, and J. Wazny. Finding all minimal unsatisfiable subsets. In *PPDP*, pages 32–43. ACM, 2003.

[8] U. Junker. Quickxplain: Conflict detection for arbitrary constraint propagation algorithms. In *IJCAI'01 Workshop on Modelling and Solving problems with constraints (CONS-1)*, Seattle, WA, USA, August 2001.

[9] U. Junker. Quickxplain: Preferred explanations and relaxations for over-constrained problems. In D. L. McGuinness and G. Ferguson, editors, *AAAI*, pages 167–172. AAAI Press / The MIT Press, 2004.

[10] M. Mannion. Using First-Order Logic for Product Line Model Validation. In *Proceedings of the Second Software Product Line Conference (SPLC2)*, LNCS 2379, pages 176–187, San Diego, CA, 2002. Springer.

[11] M. Raatikainen, T. Soininen, T. Männistö, and A. Mattila. Characterizing configurable software product families and their derivation. *Software Process: Improvement and Practice*, 10(1):41–60, 2005.

[12] R. Reiter. A logic for default reasoning. *Artif. Intell.*, 13(1-2):81–132, 1980.

[13] R. Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–95, 1987.

[14] P. Trinidad, D. Benavides, and R. A. Cortés. Isolated features detection in feature models. In *Conference on Advanced Information Systems Engineering (CAiSE'06) Forum*, June 2006.

[15] E. Tsang. *Foundations of Constraint Satisfaction*. Academic Press, 1995.

[16] T. von der Maßen and H. Lichter. Requiline: A requirements engineering tool for software product lines. In F. van der Linden, editor, *PFE*, volume 3014 of *Lecture Notes in Computer Science*, pages 168–180. Springer, 2003.

[17] T. von der Massen and H. Lichter. Deficiencies in feature models. In T. Mannisto and J. Bosch, editors, *Workshop on Software Variability Management for Product Derivation - Towards Tool Support*, 2004.

[18] W. Zhang, H. Zhao, and H. Mei. A propositional logic-based method for verification of feature models. In J. Davies, W. Schulte, and M. Barnett, editors, *ICFEM*, volume 3308 of *Lecture Notes in Computer Science*, pages 115–130. Springer, 2004.