SEKE 2005

Using Constraint Programming to Reason on Feature Models *

David Benavides, Pablo Trinidad, Antonio Ruiz-Cortés
Dpto. de Lenguajes y Sistemas Informáticos
University of Seville
Av. de la Reina Mercedes S/N, 41012 Seville, Spain
{benavides, trinidad, aruiz}@tdg.lsi.us.es

Abstract

Feature models have been cited as one of the main contributions to model software product families. However, there is still a gap in product family engineering which is the automated reasoning on feature models. In this paper we describe how to reason on feature models using constraint programming. Although, there are a few attempts to reason on feature models there are two main drawbacks in these proposals: $i)$ none of them associate parameters to features $ii)$ none of them use constraint programming as the reasoning base. Using constraint programming endows our proposal with a more powerful reasoning capacity and greater expressiveness than others.

1 Introduction

Most of the existing methods [3, 4] for Software Product Line(SPL) engineering consider that designing a compact model that represents all the possible products is an essential activity. In this context feature models [5, 9, 10, 15] have been cited as one of the most important contributions of SPL modeling [5, pag.82]. Feature models are used to model SPL in terms of features and relations among them. In these type of models, the number of potential products of a SPL may increases with the number of features. Thus, a big number of features may lead to have SPLs with a big number of potential products. That is way, automated reasoning on feature models is one of the main challenges.

On the other hand, constraint programming, as a way of reasoning, has been an active field of research in the recent decades. In this paper we propose using constraint programming to reason on feature models. To the best of our knowledge, it has not been proposed up to now.

*This is a improved version of [2]. This work was partially funded by the Spanish Ministry of Science and Technology under grant TIC2003-02737-C02-01 (AgilWeb) and PRO-45-2003 (FAMILIES)

The contribution of this paper is twofold. First, we add parameters to feature models and these parameters are taken into account in the reasoning process. Second, we describe how a feature model can be mapped onto a constraint satisfaction problem which allows make questions on the feature models, such as $i)$ how many potential products a model has $ii)$ which is the the resulting model after applying a filter (e.g. users constraints) to a model, $iii)$ which are the products of a model, $iv)$ is it a valid model, or $v)$ which is the best product from a model according to a criterion.

The remainder of this paper is structured as follow. In Section 2, we describe how to include parameters into feature models. In Section 3, we improve current reasoning on feature models. Thus, we give some definitions to be able to automatically answer to several questions about extended feature models. In Section 4, we compare our proposal regarding to others and we show how our approach can be used to obtain both variability and commonality information from a feature model. In Section 5, a running prototype implementation of our proposal is briefly described. Finally, we give some conclusion and future work in Section 6.

2 Adding Parameters to Feature Models

2.1 Feature Models

The main goal of feature modeling is to identify commonalities and differences among all products of a SPL. One of the main outputs of this activity is a compact representation of all potential products of a SPL, hereafter called *"feature model"* (FM). FMs are used to model SPL in terms of features and relations among them. Roughly speaking, a feature is a distinctive characteristic of a product. Depending on the development stage, it may refer to a requirement, a component in an architecture or even to pieces of code [12] of a SPL.

There are several notations to design FMs [5, 9, 10, 15]. We found the proposed by Czarnecki as the most comprehensible and flexible and also it is one of the most cited [5].

Figure 1 depicts the FM of JAMES [8] using Czarneky's notation. JAMES is a framework to develop web collaborative systems and it is a good example of a SPL.

Czarnecki's notation proposes four main relations, namely: mandatory, optional, alternative and or–relation. In these relations, there is always a parent feature and one (in the case or mandatory and optional) or more (in the case of alternative and or–relation) feature's childs. R_1 is an example of **Mandatory** relation (Every JAMES product has to have the $Core$ elements which are the base of the product to be operative). R_2 is an example of **Optional** relation(there are JAMES products with Web Services Interface ($WSInterface$) management and other without it). R_6 is an example of **Alternative** relation(Every JAMES product can have data base (DB), or $LDAP$ user authentication, but only one). R_7 is an example of an **Or–relation**(in a JAMES product there are products with PC or PDA Graphical User Interface, or both at the same time). There are also two more relations that are important to underline: requires and excludes relations which have the following meaning: i)**Requires**: Let A and B be two features, the relation $A\ requires\ B$ means that A needs B to be operative. For example, feature $CongressManagement$ needs feature $Repository$ to be operative, otherwise it would not work.ii)**Excludes**: Let A and B be two features, the relation $A\ excludes\ B$ means that it is not possible to have a product with A and B at the same time. For example, feature $Repository\ excludes\ PDA$ means that it is no possible to have a JAMES product with both features at the same time.

2.2 A Notation for Extended Feature Models

The most cited proposals [5, 9, 10, 15] just deal with feature as described formerly. However, there are some voices proposing that, in addition to all those characteristics, it would be very important to include parameters (also called attributes) in FMs [6, 14]. There are several concepts that we would like to clarify before giving a notation for extended FMs:

- Feature: a prominent characteristic of a product. Depending on the development stage, it may refer to a requirement (if products are requirement documents), a component in an architecture (if products are component architectures) or even to pieces of code [12] (if products are binary code in a feature oriented programming approaches) of a SPL.

- Parameter: is any characteristic of a feature that can be measured. There are two kinds of parameters i) basic parameters: parameters that are directly related to the feature or ii) derived parameters: parameters that are composed by one or more values of other parameters of the same or different features.

- Parameter domain: the space of possible values where the parameter takes its values. Every attribute belongs to a domain. It is possible to have discrete domains (e.g:integers, booleans, enumerated) or continuous domains (e.g.:reals).

Every feature of Figure 1 may have associated one or more parameters. For instance, consider $Repository$, it is possible to identify parameters related to it, such as max_size referred to the maximum size of the files that can be uploaded when using $Repository$ (real domain). Likewise any of the child features of $Modules$ can have $version$ referred to the versions of PHP (JAMES is implemented using PHP) that the module requires to work [1] (integer domain). These are examples of basic parameters.

An example of derived parameter would be the $version$ parameter of the $Modules$ feature due that it depends on the version of the modules selected in every product. In this case the version required for modules to run will be the maximum of the versions of the child features of $Modules$. It would depend on the type of relation and the type of parameter how the derived parameters would be made up. It is in accordance to the FM designer to define this composition rules.

We use an extended notation of the Czarneki's feature models that allows to represent parameters. Using the JAMES example, every feature may have one or more parameters. Thus, it would be possible to decorate the graphical FM with this kind of information. Figure 2 illustrates a piece of the FM of Figure 1 with parameters with our own notation.

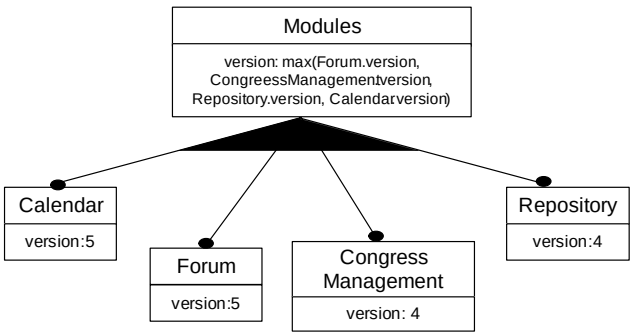

Figure 2. Extended FM for JAMES

In this example, every child feature of the $Modules$ feature have a parameter: $version$. This parameter would represent the PHP version that is needed to run the module [2].

[1] PHP 5 uses object-oriented primitives that are not available in version 4, but all PHP modules written in version 4 should work using a PHP 5 interpreter

[2] these values are just illustrative, they may have nothing to do with real values

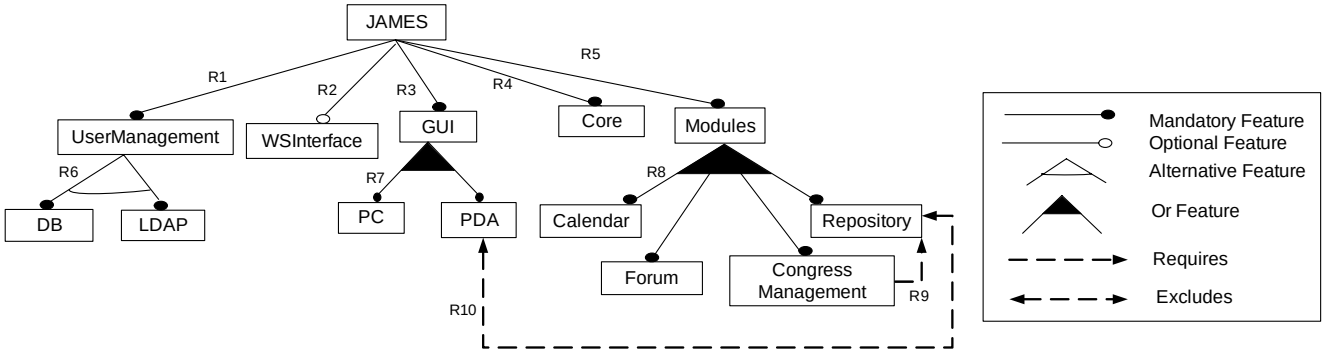

Figure 1. JAMES feature model

The parameters referred to other feature are represented using the name of the feature and the name of the parameter.

3 Automatic Reasoning on Extended Feature Models

3.1 Preliminaries

We propose to use constraint programming to reason on extended features models. The reasoning framework that we propose is depicted in Figure 3: ψ_M represents a CSP extracted from a FM following the mapping described in [1, 2]. Ω represent an operand that would serve as an input to the reasoning system that in several cases may be equals to null. This is when the operation has just ψ_M as an input. $\oplus$ represent the operation and finally φ represents the response of the question. If φ is another CSP, the composition of several operations becomes possible.

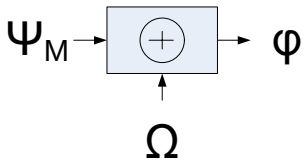

Figure 3. Reasoning Framework

3.2 Filter

There should be a way to apply filters to the model. These filters can be imposed by the users. A filter acts as a limitation for the potential products of the model. A typical application of this operation is when a customer is looking for a product with a specific set of characteristics, this is to say, they are not interested on all potential products but on some of them (those passing the filter).

According to Figure 3, $\oplus$ represents the filter operation. ψ_M represents a CSP of a FM. Ω is a CSP that represents a filter. It is important to underline that the set of variables of Ω would be a subset of the set of variables of ψ_M ($V(\psi_M)$. Finally, φ represents a CSP according to the following definition.

Definition **1 (Filter)** *Let ψ_M be a CSP representing a FM and Ω a CSP representing a filter. The result of the filter operation would be a CSP with the same set of variables and domains of ψ_M ($V(\psi_M)$ and $D(\psi_M)$) and the union of constraint of ψ_M and Ω.*

$$filter(\psi_M, \Omega) = \langle V(\psi_M), D(\psi_M), C(\psi_M) \cup C(\Omega) \rangle$$

Since the result of the filter operation is a CSP any other operation can be applied taking this result as an input.

3.3 Number of products

One of the questions to be answered is how many potential products a FM contains. This is a key question when following a SPL engineering because if the number of products increase the SPL becomes more flexible as well as more complex.

According to Figure 3, $\oplus$ represents the cardinal operation. ψ_M represents a CSP of a FM. Ω is null and φ is an integer representing the number of potential products.

Definition **2 (Cardinal)** *Let ψ_M be a CSP representing an extended FM, the number of potential products of ψ_M, here in after cardinal, is equals to the number of solutions of ψ_M.*

$$cardinal(\psi_M) = |sol(\psi_M)|$$

In the JAMES example of figure 1 $cardinal(\psi_J) = 68$, just adding for example a new Module like *FAQ* (a module for the management of Frequent Asked Questions), the number of potential products raises to 148. Likewise, a possible filter for the JAMES example would be to ask for all products with $CALENDAR$ and $FORUM$, then the number of potential products decrease from 68 to 20. Moreover, it is possible to apply a filter also to attributes. Thus,

it would be possible to ask for the products that are compatibles with version 4, then

$$\Omega = \langle V(\psi_J), D(\psi_J), MODULES.VERSION \leq 4) \rangle$$

$$cardinal(filter(\psi_J, \Omega)) = 16$$

(it decreases from 68 when any filter was imposed, to 16).

3.4 Products

There should be a way to get the solutions of the model, this is to say the products of ψ_M. A product is made up of the features with $true$ value in the solution and the parameter values. The features with $false$ values are considered to be out of the product.

According to Figure 3, $\oplus$ represents the products operation. ψ_M represents a CSP of a FM. Ω is null and φ is a set representing all the solutions of ψ_M.

Definition **3 (Products)** *Let ψ_M be a CSP representing an extended FM, the potential products of the model ψ_M, here in after products, is equals to solutions of ψ_M.*

$$products(\psi_M) = \{s \in sol(\psi_M)\}$$

3.5 Number of Features

There should be a way to know the number of features that are present in a single product. This is important due that a product would be more complex if it has a large amount of features.

According to Figure 3, $\oplus$ represents the features operation. ψ_M represents a CSP of a FM. Ω is a filter that impose the features of the product that we want to know the number of features. Thus, $cardinal(\Omega) = 1$. φ is an integer representing the number of features of the product imposed by the filter.

Definition **4 (Features)** *Let ψ_M be a CSP representing an extended FM and Ω a CSP representing a product of ψ_M, the number of features of the product, here in after features, is equals to the number of variables with $true$ values of the solution of P.*

$$features(\psi_M, \Omega) = |products(filter(\psi_M, \Omega))|$$

In the JAMES example there are several products composed by several features. For example if $P = \{DB, PC, CORE, CALENDAR, FORUM\}$ then, $features(P) = 5$.

3.6 Validation

A valid extended FM is a model where at least a product can be selected. This is to say, it is a model where ψ_M has at least one solution.

According to Figure 3, $\oplus$ represents the valid operation. ψ_M represents a CSP of a FM. Ω is null and φ is a boolean.

Definition **5 (Valid model)** *Let ψ_M be a CSP representing an extended FM, ψ_M is valid iff its equivalent CSP is satisfiable.*

$$valid(\psi_M) = (\mid sol(\psi_M) \mid > 0)$$

The JAMES model of the example is valid, but there might be situations where the constraints are not satisfiable therefore the model becomes invalid. For instance, if a JAMES product with $FORUM$ and compatible with PHP 4 is desired, then the model is not valid:

$$C = (FORUM = true \wedge MODULES.VERSION \leq 4)$$
$$\Omega = \langle V(\psi_J), D(\psi_J), C \rangle$$
$$valid(filter(\psi_M, \Omega)) = false$$

3.7 Optimum products

There should be a way of finding out the best products following a criterion. This is to say, in addition to select the features of a product, it would be interesting to select the best product following a criterion. In this case, we define the operation opt.

According to Figure 3, $\oplus$ represents the opt operation. ψ_M represents a CSP of an extended FM. Ω is an objective function and φ is a solution of ψ_M.

Definition **6 (Optimum)** *Let ψ_M be a CSP representing an extended FM and Ω an objective function, then the optimum set of products, here in after opt, is equals to the optimum space of ψ_M.*

$$opt(\psi_M, \Omega) = min(\psi_M, \Omega)$$

It is also possible to ask for an optimal product in the JAMES example. Thus, a possible optimum would be to ask for the product with the minimum number of features. In this case selected products P_{opt} are:

$$\Omega = {}_{features(\psi_M, P)}$$
$$opt(\psi_M, \Omega) = min(\psi_M, \Omega)$$

The model presented in this section can support current features models. The only difference is that current feature models do not support parameters. Hence, to use our model to reason on current feature models, parameters are not taken into account. Thus, the all definitions presented formerly remain valid for current feature models.

4 Realizing the Benefits

Our approach is very flexible regarding to others. To the best of our knowledge, there are only a couple of limited attempts by Van Deursen *et al.* and Mannion [7, 11] that treat automatic manipulation of feature models. It is important to note that these two proposals do not consider parameters in the features which is something that we do. Van Deursen *et al.* [7] explore automated manipulation of features descriptions providing an algebra to operate on the feature diagrams proposed by [5]. Mannion's proposal [11] uses first–order logic for product line reasoning. However it only provides a model based on propositional–logic using AND, OR and XOR logical operators to model SPLs based on feature models. Both attempts have several limitations:

1. They do not allow deal with parameters(both of them just say to be a future work).
2. They basically answer to the single question of how many products a model has.
3. As far as we know, they do not have an implementation available.

In addition, Mannion's model uses the XOR ($\oplus$) operator to model alternative relations, which is either a mistake or a limitation. Thus, the model becomes invalid if more than two features take part of an alternative relation.

Moreover, our proposal is very extensible. Next, we shown two more definitions that are based on the definitions of previous section.

4.1 Variability

As seen before, FMs are composed of a set of features and relations among them. If relations restrict the number of product to only one, we are considering the lowest variability. Let us take into account that a FM defining no possible product would be considered a non-valid model. On the other hand, considering no relations, the number of products within the FM would be the highest. This case would represent the highest variability. Relations restrict the number of potential products, so variability depends on relation types.

Let a leaf feature be a feature that has no child feature. Parent features add no variability to the model, because they are features aggregates. Thus, we define the variability factor as follow.

Definition **7 (Variability Factor(VF))** *Let ψ_M be a CSP representing an extended FM. Let ψ_{M^V} be another CSP representing another extended FM, considering the leaf features in ψ_M and no relation among its features. Then the VF is defined as follow*

$$VF(\psi_M) = \frac{cardinal(\psi_M)}{cardinal(\psi_{M^v})} = \frac{|sol(\psi_M)|}{|sol(\psi_M^V)|}$$

Variability factor would take values in the real domain within the range from 0 to 1. In the case of JAMES example, $VF(\psi_J) = \frac{68}{1024} \approx 0.07$

VF can assist decision making. For instance, one of the first decisions to be taken when many products are going to be developed, is whether SPL approach or traditional approach is going to be applied. A high VF may suggest a SPL approach; a low VF may suggest a traditional approach.

4.2 Commonality

In a FM, some features will appear in every product, some in only one product and others in some products. When deciding the order in which features are going to be developed, knowing which are the most common features is very important in order to prioritize their building. Obtaining commonality information from the FM can be feasible by asking questions to our model. We define feature commonality as the percentage of products where that feature is part of the product.

Definition **8 (Commonality)** *Let ψ_M be a CSP representing an extended FM and F the feature we want to know its commonality.*

$$commonality(\psi_M, F) = \frac{cardinal(filter(\psi_M, F = true))}{cardinal(\psi_M)}$$

The feature $Core$ in the JAMES example has a commonality equals to 1 and the feature $Forum$,

$$commonality(\psi_J, Forum) = \frac{40}{68} \approx 0.58$$

5 Implementation

We have already implemented some of the ideas presented in this paper available at *http://www.tdg-seville.info/topics/spl*. This implementation uses OPL Studio, a commercial CSP solver.

Three modules have been developed in our implementation: first, a feature markup language and XML Schema were agreed. This language allows to represent the Czarnecki's FM [5]. Second a parser to transform this XML documents to a CSP following the algorithm described in

[1, 2] was developed. Finally, a web–based prototyping interface is available that allows to test some of the capabilities of the model. In order to test our implementation, we have modeled five problems (two academical and three real product lines) that are accessible at the web site.

In order to evaluate the implementation, we measured its performance and effectiveness. We implemented the solution using Java. We run our tests on a WINDOWS XP PROFESSIONAL machine that was equipped with a 1.5Ghz AMD Athlon XP microprocessor, and 496 MB of DDR 266Mhz RAM memory. We based our testing in the FM in Figure 1, adding new features. Several tests were made on each FM in order to avoid as much exogenous interferences as possible.

We have experimentally inferred that the implementation presented has an exponential behavior while increasing the number of features in the FM and maintaining a constant variability factor. We have measured the solving time for $products(M)$, which is the most complex to obtain, and have considered it for different values of VF as shown in Figure 4. Our test determines our model has a good performance until 25 features while the VF is kept constant.

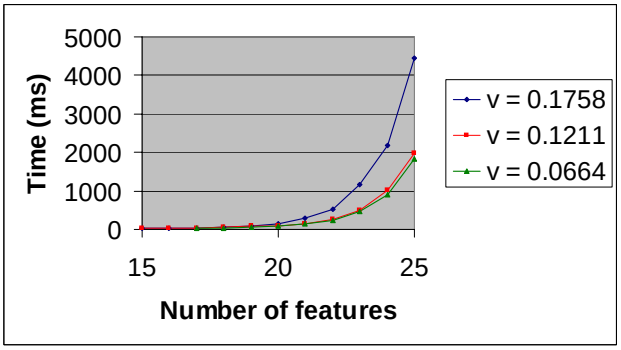

Figure 4. Empirical performance test for $products(M)$

6 Conclusion and Further Work

In this paper we put the basis for reasoning on FM with features and parameters at the same time and in the same model using constraint programming.

There are some challenges we have in the near feature, namely: $i)$ developing a case tool to validate our model on an industrial context, $ii)$ perform a more rigorous validation of our implementation studying the influences as well as the number of solutions, the types of relations, the number of features, and so on, $ii)$ comparing our work regarding others in the field of product configuration [13] and $iv)$ obtaining heuristics related to variability and commonality in an empirical software engineering context.

References

[1] D. Benavides, A. Ruiz-Cortés, and P. Trinidad. Coping with automatic reasoning on software product ines. In *Proceedings of the 2nd Groningen Workshop on Software Variability Management*, Nov. 2004.

[2] D. Benavides, A. Ruiz-Cortés, and P. Trinidad. Automatic reasoning on feature models. In *The 17th Conference on Advanced Information Systems Engineering (CAiSE'05)*. LNCS, June 2005.

[3] J. Bosch. *Design and Use of Software Architectures*. Addison-Wesley, 1^{th} edition, 2000.

[4] P. Clements and L. Northrop. *Software Product Lines: Practices and Patterns*. SEI Series in Software Engineering. Addison–Wesley, Aug. 2001.

[5] K. Czarnecki and U. Eisenecker. *Generative Programming: Methods, Techniques, and Applications*. Addison–Wesley, may 2000. ISBN 0–201–30977–7.

[6] K. Czarnecki, S. Helsen, and U. Eisenecker. Staged configuration using feature models. In *Proceedings of the Third Software Product Line Conference 2004*, pages 266–282. Springer, LNCS 3154, 2004.

[7] A. v. Deursen and P. Klint. Domain–specific language design requires feature descriptions. *Journal of Computing and Information Technology*, 10(1):1–17, 2002.

[8] P. Fernandez and M. Resinas. James project. *Available at http://jamesproject.sourceforge.net/*, 2002-2005.

[9] K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson. Feature–Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, Nov. 1990.

[10] K. Kang, J. Lee, and P. Donohoe. Feature–Oriented Product Line Engineering. *IEEE Software*, 19(4):58–65, July/Aug. 2002.

[11] 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.

[12] C. Prehofer. Feature-oriented programming: A new way of object composition. *Concurrency and Computation: Practice and Experience*, 13(6):465–501, 2001.

[13] T. Soininen, J. Tiihonen, T. Männistö, and R. Sulonen. Towards a general ontology of configuration. *AI EDAM*, 12(4):357–72, 1998.

[14] D. Streitferdt, M. Riebisch, and I. Philippow. Details of formalized relations in feature models using ocl. In *Proceedings of 10th IEEE International Conference on Engineering of Computer–Based Systems (ECBS 2003), Huntsville, USA. IEEE Computer Society*, pages 45–54, 2003.

[15] J. van Gurp, J. Bosch, and M. Svahnberg. On the notion of variability in software product lines. In *Proceedings of the Working IEEE/IFIP Conference on Software Architecture (WICSA'01), IEEE Computer Society*, pages 45–54, 2001.