FAMA: Tooling a Framework for the Automated Analysis of Feature Models

David Benavides, Sergio Segura, Pablo Trinidad and Antonio Ruiz-Cortés
Department of Computer Languages and Systems
University of Seville
Av. de la Reina Mercedes S/N, 41012 Seville, Spain
e-mail: {benavides, segura, trinidad, aruiz}@tdg.lsi.us.es

Abstract

The automated analysis of feature models is recognized as one of the key challenges for automated software development in the context of Software Product Lines (SPL). However, after years of research only a few ad-hoc proposals have been presented in such area and the tool support demanded by the SPL community is still insufficient. In previous work we showed how the selection of a logic representation and a solver to handle analysis on feature models can have a remarkable impact in the performance of the analysis process. In this paper we present a first implementation of FAMA (FeAture Model Analyser). FAMA is a framework for the automated analysis of feature models integrating some of the most commonly used logic representations and solvers proposed in the literature. To the best of our knowledge, FAMA is the first tool integrating different solvers for the automated analyses of feature models.

1. Introduction

Feature Modelling is a common mechanism to manage variability in the context of software product lines (SPL). A Feature Model (FM) represents all possible products in an SPL in terms of features. A feature is an increment in product functionality. FMs can be used in different stages of development such as requirements engineering [14, 15] , architecture definition or code generation [1, 4].

Automated analyses of FMs is recognized in the literature as an important challenge in SPL research [1, 3]. The automated analyses of FMs is usually performed in two steps: i) The FM is translated into a certain logic representation ii) Off-the-shelf solvers are used to extract information from the result of the previous translation such as the number of possible products of the feature model, all the products following a criteria, finding the minimum cost configuration, etc [6]. In previous works we introduced how the use of different solvers [7] and logic representations [8] can have an important effect in the time and memory performance of the analysis process. We also showed that there is not an optimum logic representations and solver for all the operations that can be performed on an FM.

Existing tools to analyse FMs depend on concrete solvers and their performance could be improved if they used several solvers. Two options arise to support multiple solvers: i) Extending an existing tool to use several solvers, ii) create our own multi-solver framework that supports any kind of representation and solver and could interoperate with existing tools. We have chosen the second option because the analysed tools are either not open–source tools or they are not prepared to be adapted to support multiple solvers.

In this paper we present a first prototype implementation of FAMA. FAMA is an extensible framework for the automated analysis of feature models. FAMA allows the integration of different logic representations and solvers in order to optimize the analysis process. It can be configured to select automatically in execution time the most efficient of the available solvers according to the operation requested by the user. The current implementation of FAMA integrates three of the most promising logic representations proposed in the area of the automated analysis of feature models: CSP, SAT and BDD, but more solvers can be added if needed. The implementation is based on an Eclipse plug–in and uses XML to represent FMs so it can interoperate with other tools that support it.

The remainder of the paper is structured as follows: in Section 2 the automated analysis of FMs is outlined and details of the performance offered by the solvers used are presented. Section 3 focuses on the description of the functionality and some of the most relevant design and implementations details of the framework. A brief overview of some related tools is introduced in Section 4. Finally we summarize our conclusions and describe our future work in Section 5.

2. Automated Analysis of Feature Models

Once an FM is translated into a suitable representation it is possible to use off–the–shelf solvers to automatically perform operations to analyse a FM [5, 6].

The current implementation of the framework integrates three of the most commonly used logic representations proposed for the automated analyses of feature models: CSP, SAT and BDD. A complete performance test of solvers dealing with such representations and details about the translation of an FM into a CSP, SAT and BDD were introduced in [8, 5]. Following, we overview these representations and we outline the most relevant performance details obtained from such test.

2.1 Constraint Satisfaction Problem (CSP)

A *Constraint Satisfaction Problem* (CSP) [17] consists on a set of variables, finite domains for those variables and a set of constraints restricting the values of the variables. Constraint Programming can be defined as the set of techniques such as algorithms or heuristics that deal with CSPs. A CSP is solved by finding states (values for variables) in which all constraints are satisfied. CSP solvers can deal with numerical values such as integer domains. The main ideas concerning the use of constraint programming on FM analysis were stated in [5].

Constraint Programming is the most flexible proposal. It can be used to perform the most of the operations currently identified on feature models [6]. However, constraint programming solvers reveal a weak time performance when executing certain operations on medium and large size feature models as for example calculating the number of possible combinations of features [7, 8].

2.2 Boolean Satisfiability Problem (SAT)

A propositional formula is an expression consisting on a set of boolean variables (literals) connected by logic operators $(\neg, \wedge, \vee, \rightarrow, \leftrightarrow)$. The *propositional satisfiability problem* (SAT) [12] consists on deciding whether a given propositional formula is satisfiable, i.e., a logical values can be assigned to its variables in a way that makes the formula true. The basic concepts about the using of SAT in the automated analysis of FMs were introduced in [1].

The performance results of SAT solvers are slightly better than the results of CSPs however this approach is not so powerful [8]. To the best of our knowledge, there is not any approach in which feature models attributes can be translated to SAT in order to perform operations as maximizing or minimizing attribute values.

2.3 Binary Decision Diagrams (BDD)

A *Binary Decision Diagram* (BDD) [10] is a data structure used to represent a boolean function. A BDD is a rooted, directed, acyclic graph composed by a group of decision nodes and two terminal nodes called 0-terminal and 1-terminal. Each node in the graph represents a variable in a boolean function and has two child nodes representing an assignment of the variable to 0 and 1. All paths from the root to the 1-terminal represents the variable assignments for which the represented boolean function is true meanwhile all paths to the 0-terminal represents the variable assignments for which the represented boolean function is false.

Although the size of BDDs can be reduced according to some established rules, the weakness of this kind of representation is the size of the data structure which may vary between a linear to an exponential range depending on the variable ordering [10]. Calculating the best variable ordering is an NP-hard problem. However, the memory problem is clearly compensated by the time performance results offered by BDD solvers. While CSP and SAT solver are incapable of finding the total number of solutions of medium and large size feature models in a reasonable time, BDD solvers can work out it in an insignificant time, so it justifies their usage at least on counting operations.

3 FAMA framework

FAMA has been implemented as a complete tool for the edition and analysis of FMs. FAMA supports cardinality-based feature modelling (that includes traditional feature models, e.g FODA, Feature–RSEB, and so on), export/import of FMs from XML and XMI and analysis operations on FMs. In order to make our tool multiplatform and easy to access we implemented FAMA as a plugin for the Eclipse Platform[1]. In the next sections we overview of the functionality offered by the framework and we describe some of the most relevant design and implementation details.

3.1. General Description

FAMA offers two main functionalities: visual model edition/creation and automated model analysis. Once the user has created or imported (from XML/XMI) a cardinality-based FM, the analysis capability can be used. Most of the operation identified on FMs [6] are being currently implemented. At the moment of writing this article the operations fully supported by FAMA are:

[1] http://www.eclipse.org/

- Finding out if an FM is valid, i.e. it exists a a product satisfying all the constraints.
- Finding the total number of possible products of an FM (number of products).
- List all the possible products of a feature model (list of products).
- Calculate the commonality of a feature, i.e. the number of products where a feature appears in.

FAMA integrates different solvers in order to combine the best of all of them in terms of performance. The actual version of the framework integrates CSP [2], SAT [3] and BDD [4] Java solvers to perform the analysis tasks. However, an unlimited number of new analysis operations and solvers could be added.

One of the advantages of FAMA is the ability to select automatically, in execution time, the most efficient solver according to the operation requested by the user. The mapping from a FM onto the correspondent solver is done on demand. Therefore, if the user asks for the number of possible combinations of features of an FM the framework will select automatically the BDD solver to get it (the most efficient known approach for this operation). The automated selection of a solver is based on the value of some configuration parameters establishing the priority between the available solvers for each operation. The values of these parameters were set according to the results from a complete performance test of the solvers integrated in the framework [8].

3.2. Design and Implementation

FAMA is logically divided into two modules: the analysis engine and the visual editor (Figure 1).

The FAMA analysis engine is the responsible for performing the analysis tasks requested by the user. It receives as main input what we define as questions. A question is composed by an FM in XML format and the parameters, if any, associated to the operation requested by the user. The FMs supplied as main input of the analysis engine are instances of an FM Metamodel [9] defined in an XML Schema. The metamodel defines two kinds of relations: binary relations (such as mandatory, optional and cardinality relations) and set relations (such as alternative, or-relations and cardinality groups). It accepts *depends* and *excludes* constraints and feature attributes.

The FAMA graphical user interface was implemented using the *Eclipse Modeling Framework* (EMF). EMF is a modelling framework and code generation facility for building tools based on a structured data model. From a model specification described in XMI, EMF provides tools and runtime support to produce a set of Java classes for the model, a set of adapter classes that enable editing of the model, and a basic editor. The EMF editor of the presented tool is based on an Extended FM Metamodel containing all the information of the metamodel used in the analysis engine plus information specifying the way in which the FMs can be edited.

The FMs created are saved in XMI format in order to maximize the interoperability with other applications using the OMG specifications. However, importation and exportation of FMs in XML format is also supported.

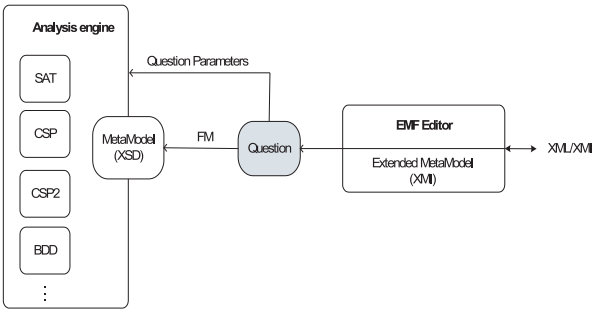

Figure 1. Analysis Engine and EMF Editor

FAMA has been implemented as a plugin of the Eclipse Platform. Eclipse is an open development platform comprised of extensible frameworks, tools and runtimes for developing and managing software. The integration of a tool in the Eclipse Platform not only reduce significantly the development effort but also guarantee that the built software will be easy to access and install in any platform using Java. A screenshot of our FAMA Eclipse plug–in deployed is shown in Figure 2.

The GUI of the plug–in is composed by two main customised tree views: the modelling view (Figure 3) and the analysis view (Figure 4). The modelling view allows the edition and modification of existing FMs. Roughly speaking, the edition process is based on the selection of existing nodes and the use of the context menu to add new subnodes. When a new FM is created the root node is automatically added. The attributes of the nodes (name, description, min and max cardinality, etc.) can be edited using a property view implemented for such end.

Once the FM is created or imported, the user can start using the analysis view to automatically extract information from it. When the analysis view is enabled all the features of the FM adopt a selection state. The user can apply filters by labelling the feature as selected or removed.

By default, FAMA uses the most efficient solver for

[2] we used JaCoP solver http://www.cs.lth.se/home/Radoslaw_zymanek/
[3] we used SAT4j http://www.sat4j.org
[4] we used JavaBDD, http://javabdd.sourceforge.net

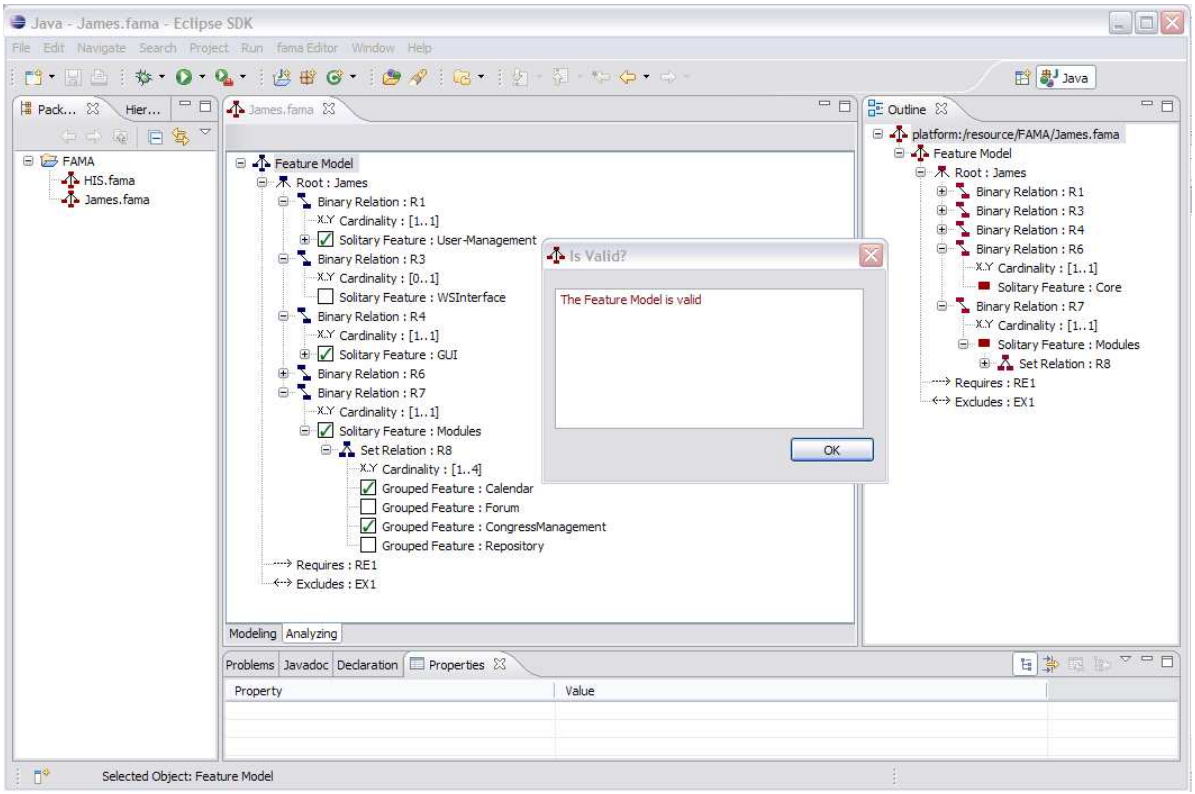

Figure 2. Eclipse Plugin

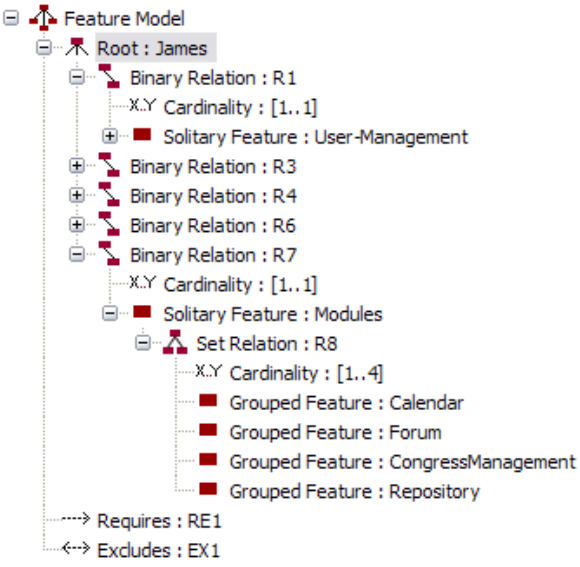

Figure 3. Modeling View

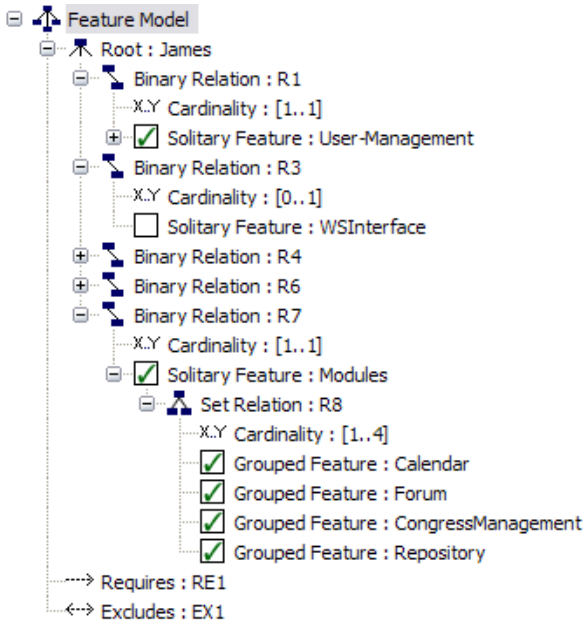

Figure 4. Analysis View

each operation. Current implementation uses JaCoP[5] solver for CSP representation, Sat4j[6] for SAT and JavaBDD[7] for BDD. However, the plugin is equipped with a fine–grained configuration utility which allow selecting what of the available solver will be used to perform each operation. Figure 5 shows a screenshot of the property page used to set the configuration options.

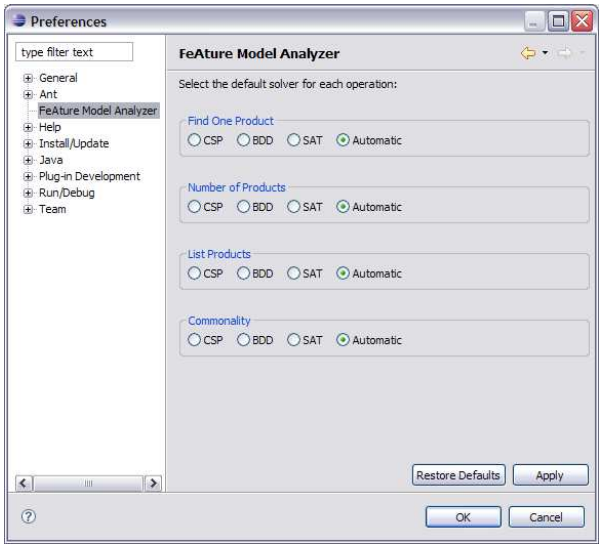

Figure 5. Preferences Page

4 Related Work

The Feature Model Plugin[8] (FMP) [13] has also been implemented as an Eclipse plug–in. It supports cardinality-based feature modelling, specialization of feature diagrams and configuration based on feature diagrams. FMP uses a BDD solver to work out the number of possible combination of features in an FM. It also supports the use of filters. FMP is becoming a mature tool with interesting extensions but it does not have the analysis of FMs between its main goals. It does not support attributed feature models and do not include more than one solver for the analysis.

XFeature[9] [11] is an XML-based feature modelling tool also implemented as an Eclipse plug–in. XFeature supports the modelling of product families and applications instantiated from them. This tool does not support the automated analysis of FMs.

CaptainFeature[10] is a feature modelling tool using the FODA notation to render feature diagrams. It also includes an integrated configurator to specialize the created feature diagrams. CaptainFeature does not support the automated analysis of FMs.

Requiline[11] [18] is defined as a requirement engineering tool for the efficient management of software product lines. From the edition of a group of features and requirements Requiline derives product configurations. It also includes a consistency checker and a query and XML interface. Apart from the consistency checking, RequiLine does not perform any of the others analysis operation identified on FMs [6].

Pure::Variants[12] is a commercial tool supporting feature modelling and configuration. This actual version of tool does not support cardinalities. Pure::Variants supports basic analysis operations through a Prolog-based constraint solver.

The *AHEAD Tool Suite*[13] (ATS) [2] is a set of tools for product-line development that support feature modularizations and their compositions. AHEAD can perform certain analysis operations on FMs by means of SAT solvers [1]. It does not include feature models attributes in the analysis. Feature models are saved as grammars and no XML representation is provided.

Table 1 summarizes the exposed proposals.

	FMP	XFeature	CaptainFeature	RequiLine	Pure::Variants	AHEAD Tool Suite	**FAMA**
CSP	-	-	-	-	-	-	+
SAT	-	-	-	-	-	+	+
BDD	+	-	-	-	-	-	+
Multi Solver	-	-	-	-	-	-	+
Basic FMs	+	+	+	+	+	+	+
Cardinality-based FMs	+	+	-	-	-	-	+
XML/XMI	+	+	-	+	+	-	+
Maturity	+	+	+	+	+	+	∼

Table 1. Summary of the proposals

5 Conclusions and Future Work

In this paper we introduced the first prototype implementation of FAMA which is a framework for the automated analysis of feature models. We introduced the logic representations currently used in the framework and we exposed

[5] http://www.cs.lth.se/home/Radoslaw zymanek/
[6] http://www.sat4j.org
[7] http://javabdd.sourceforge.net
[8] http://gp.uwaterloo.ca/fmp/
[9] http://www.pnp-software.com/XFeature/
[10] https://sourceforge.net/projects/captainfeature/
[11] http://www-lufgi3.informatik.rwth-aachen.de/TOOLS/requiline/
[12] http://www.pure-systems.com/
[13] http://www.cs.utexas.edu/users/schwartz/ATS.html

some of the most relevant design and implementation details. Finally, we compared our proposal with some other feature modelling tools and we concluded that this is the first tool integrating different solvers to optimize the analysis of feature models. Although FAMA is not a mature tool yet, its promising capabilities of extensibility, interoperability and integration make it a tool to take into account in the future.

Several challenges remain for our future work. The integration of new solvers and new analysis operations are currently in process. All the integrated solvers performance must be analysed in order to define the selection criterion for each kind of operation. We are also studying licenses issues in order to release our tool. We really trust that formal semantics [16] are needed for the verification of our framework and we are working on that direction too.

Acknowledgments

The work reported in this article was supported by the Spanish Ministry of Science and Technology under grants TIC2003-02737-C02-01 and TIN2006-00472. We would like to thank Salvador Trujillo and Daniel Le Berre for their helpful comments on an earlier draft of this paper.

References

[1] D. Batory. Feature models, grammars, and propositional formulas. In *Software Product Lines Conference, LNCS 3714*, pages 7–20, 2005.

[2] D. Batory. A tutorial on feature oriented programming and the ahead tool suite. In *Summer school on Generative and Transformation Techniques in Software Engineering*, 2005.

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

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

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

[6] D. Benavides, A. Ruiz-Cortés, P. Trinidad, and S. Segura. A survey on the automated analyses of feture models. In *Jornadas de Ingeniería del Software y Bases de Datos (JISBD)*, 2006.

[7] D. Benavides, S. Segura, P. Trinidad, and A. Ruiz-Cortés. Using java csp solvers in the automated analyses of feature models. *LNCS*, 4143:389–398, 2006.

[8] D. Benavides, S. Segura, P. Trinidad, and A. Ruiz-Cortés. A first step towards a framework for the automated analysis of feature models. In *Managing Variability for Software Product Lines: Working With Variability Mechanisms*, 2006.

[9] D. Benavides, S. Trujillo, and P. Trinidad. On the modularization of feature models. In *First European Workshop on Model Transformation*, September 2005.

[10] R. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35(8):677–691, 1986.

[11] V. Cechticky, A.Pasetti, O. Rohlik, and W. Schaufelberger. Xml-based feature modelling. *LNCS, Software Reuse: Methods, Techniques and Tools: 8th ICSR 2004. Proceedings*, 3107:101–114, 2004.

[12] S. Cook. The complexity of theorem-proving procedures. In *Conference Record of Third Annual ACM Symposium on Theory of Computing*, pages 151–158, 1971.

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

[14] S. Jarzabek, W. Ong, and H. Zhang. Handling variant requirements in domain modeling. *The Journal of Systems and Software*, 68(3):171–182, 2003.

[15] 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.

[16] P. Schobbens, P. Heymans, J. Trigaux, and Y. Bontemps. Feature Diagrams: A Survey and A Formal Semantics. In *Proceedings of the 14th IEEE International Requirements Engineering Conference (RE'06)*, Minneapolis, Minnesota, USA, September 2006.

[17] Edward Tsang. *Foundations of Constraint Satisfaction*. Academic Press, 1995.

[18] T. von der Massen and H. Lichter. Requiline: A requirements engineering tool for software product lines. In F. van der Linden, editor, *Proceedings of the Fifth International Workshop on Product Family Engineering (PFE-5)*, LNCS 3014, Siena, Italy, 2003. Springer Verlag.