# FaMa

**David Benavides, Pablo Trinidad, Antonio Ruiz-Cortés, and Sergio Segura**

***What you will learn in this chapter***

- *The scope of the automated analysis of variability models.*
- *How to use a tool for the automated analysis of variability models.*
- *A product line architecture to build analysis tools.*

## 1 Introduction

Extracting relevant information from variability models is an important task to support decision-making in product line development and product configuration. Examples of questions that can arise during development are:

- Which are the products that contain a certain subset of features?
- Does the variability model contain any errors, i.e. contradictory information?
- How many products are we able to build?
- How much does it cost to produce a certain product?

Answering the above questions can be a tedious and error-prone task, and it is infeasible to perform manually with large-scale models; so it requires to be automated.

Using tools for the Automated Analysis of Variability Models (AAVM, see Definition 11.1) is part of the life of software product line developers. As an example of such tools, we present in this chapter the FeAture Model Analyser (FaMa) Tool Suite. It is an ecosystem of tools that focuses on the most used variability modelling languages: feature models. It contains several software tools some of which are:

D. Benavides (✉) • P. Trinidad • A. Ruiz-Cortés • S. Segura
University of Seville, Seville, Spain
e-mail: benavides@us.es; ptrinidad@us.es; aruiz@us.es; sergiosegura@us.es

- FaMa Framework: a customizable analysis framework.
- FaMa Test Suite: a set of implementation-independent test cases to test feature model analysis tools such as those created by the FaMa Framework.
- FaMa Integrations: a set of developments to integrate FaMa Framework into other tools such as Moskitt Feature Modeler.
- BeTTy Framework: Betty [1] is a highly configurable framework supporting benchmarking and functional testing of variability model analysis tools.

The tool suite is under LGPL v3 licence and can be found at http://www.isa.us.es/fama. In this chapter we focus on FaMa Framework, the masterpiece in the puzzle of tools provided by the ecosystem.

### *1.1 Definitions and Examples*

**Definition 11.1.** Automated analysis of variability models

The Automated Analysis of Variability Models (AAVM) is about the automated extraction of information from variability models using automated mechanisms to assist decision-making in PL development or life cycle. This is a key activity in variability management because it allows checking and extracting information from the variability model which is central in the development of a PL.

**Example 11.1.** Example of analysis operations

Let's think about a feature model as a possible variability model. A feature model can be analysed automatically extracting valuable information from it. For instance, dead features could be automatically detected. Dead features are features that are represented in the model but can never be part of a concrete product because model internal inconsistencies. At the time of writing this book, there are 30 different analysis operations reported in the literature [2].

## 2 FaMa Framework

FaMa Framework (FW) is a product line of tools to analyse variability models. It mainly performs analysis operations transforming a variability model into a suitable logic, which is used to reason about the model using off-the-shelf logic solvers. FaMa FW is developed as a product line, which contains many variant features. For example, there exist different variability meta-models (a.k.a. feature model dialects) and file-formats, several analysis operations (a.k.a. *questions*) to be performed over them. These questions are answered using different *reasoners* or logic solvers, each of them performing better or worse for a kind of model, which is determined by the *reasoner selectors*. Since some analysis operations can take long time to solve, the framework provides for several *transformations* to improve the response time.

Figure 11.1 represents a feature model of the FaMa FW. It offers a wide variety of variant features, such as meta-models, logic reasoners, analysis operations,

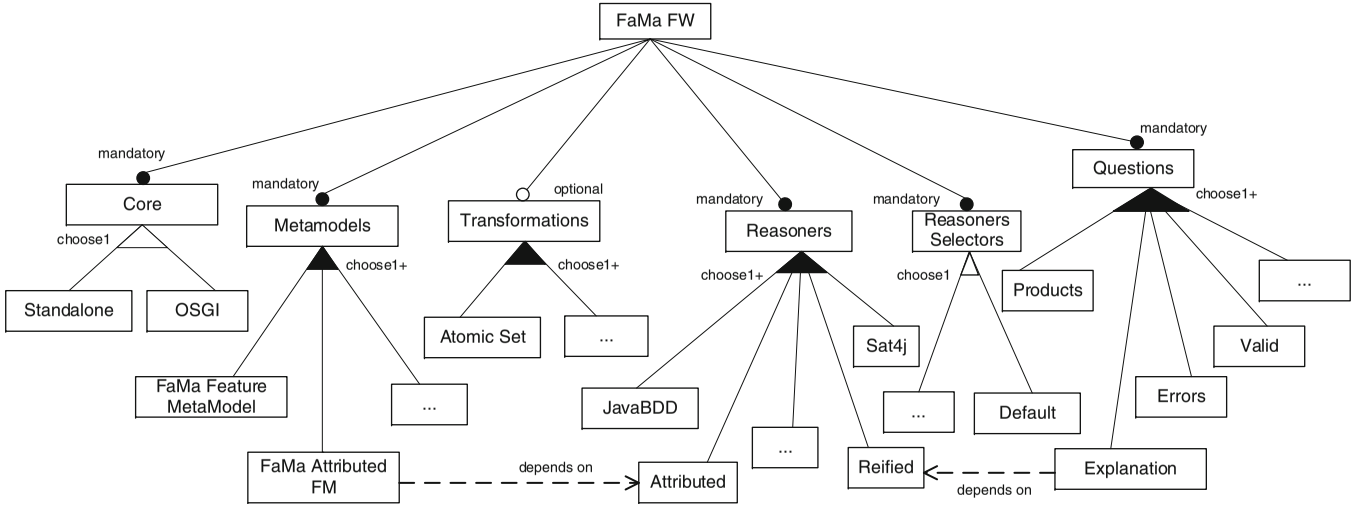


**Fig. 11.1** A feature model describing FaMa Framework variability

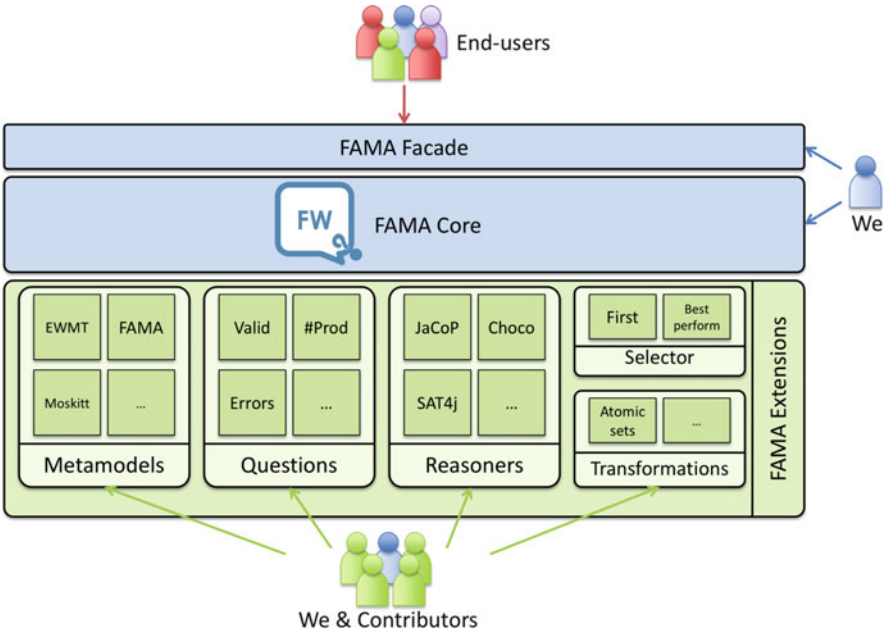


**Fig. 11.2** FaMa Framework architecture

transformations and reasoner selectors. A product in FaMa FW product line is therefore a selection of the variant features that are useful for a certain context, such as a CASE tool for PL, a product configuration tool or a reconfigurable smart home.

## *2.1 FaMa Architecture*

FaMa FW has a component-based architecture as shown in Fig. 11.2. Every variant feature is built as an independent component or *FaMa Extension*. These extensions can be classified as follows:

- Meta-models: support different FM representations and file formats.
- Reasoners: map a feature model into a concrete logic. This logical representation is used to solve analysis operations.
- Questions: interfaces that define the available analysis operations independently of the specific reasoners that answer them.
- Criteria selectors: select the most efficient reasoner for a particular question and a given feature model.
- Transformations: transform a feature model into an equivalent one in terms of products but easier to analyse in terms of performance.

A FaMa FW product is customised selecting a valid set of FaMa Extensions that comes together with the *FaMa Core*, which is the common and mandatory part of the architecture among different FaMa FW products. It mainly communicates meta-model and reasoner components through mappings, registers the available questions and use criteria selectors to search for the reasoners that may answer the question a user demands. All this process is performed independently of specific components, avoiding any kind of coupling among them. We can build our own customised FaMa FW product selecting the components we are interested in. We assume that the SPL developers, as end-users, are not interested in the internal aspects. To this purpose, FaMa FW just wants to analyse feature models with the best performance. We provide *FaMa Facade* as a transparent way to integrate FaMa FW into existing systems. It is a stable facade that hides the complexity of FaMa Core into a unique interface. It is designed to reduce the changes that are carried to users; so a new extension can be deployed while the façade remains unaffected.

FaMa FW aims to be integrated into existing CASE tools. Eclipse has become a standard in CASE tool development. FaMa Core and Extensions are OSGi-compliant [3] which is a requirement for being Eclipse-compliant. FaMa FW not only uses this technology as an alternative to support componentization but also provides for its own componentization technology that permits it to work as a standalone application wherever OSGi is not available.

## 3 Examples and Recommended Areas of Practice

FaMa FW has been designed to be an important piece of third-party products. It provides for an API offered as an OSGi bundle and a Java library distribution. At date, it has been used in four kinds of products:

- Feature Modeling Tools: Incorporating analysis capabilities to visual editors and other CASE tools that use VMs somehow. Moskitt Feature Modeller [4], an Eclipse-based visual editor of feature models is an example of it.
- Product Configurators: Providing product configuration capabilities to validate feature selections and to assist end user by propagating user decision and suggesting corrections for invalid configurations. These products usually use a fixed feature model. FaMa Debian Package and ISA Packager use FaMa FW for this purpose.
- Dynamic Systems Reconfiguration: Restoring and reconfiguring dynamic systems such as smart homes [5] and TV broadcasting systems [6] whenever errors happen or system preferences change. FaMa Lite for Smart Homes supports the decision of the features to activate or deactivate services whenever a new feature is deployed or an existing one fails.
- Fast Prototyping Framework: Building tools for the development of new variant features for FaMa FW PL. We have developed the BeTTy Framework [1] and

the FaMa Test Suite and FaMa Random Generator to test the functionality and performance of FaMa FW reasoners and FaMa SDK, an environment to develop new variant features.

For a first contact to FaMa FW, it is recommended to use its console interface. It permits a complete interaction with all the elements in the infrastructure and may be used to evaluate its capabilities. Next, we show an example of how to specify an FM using our file-format and how to use FaMa FW from the console and a Java application as a library.

## 3.1 An Example Input Feature Model for FaMa FW

FaMa FW supports several feature meta-models and also offers a plain-text format to represent all the kinds of relationships that can be found in the bibliography. The example below describes the FM in Fig. 11.1 in a textual format.

Notice that a FM is divided into hierarchical relationships and cross-tree constraints. Any relationship follows the next syntax:

`Parent: [min_card,max_card] {Child1 Child2 ...};`

This cardinality-based relationship allows the definition of mandatory ([1,1]) and optional ([0,1]) relationships for a one-child relationship and alternative ([1,1]), or ([1,N]) and set relationships for a multiple-child relationship. Constraints can represent *require* and *exclude* constraints and any other constraint that can be represented in terms of a Boolean constraint. Although there is a full support for extended feature models, allowing working with attributes, this example avoids them for the sake of simplicity.

**Example 11.2.** A definition of a FM using FaMa plain-text format

```
%Relationships

FaMaFW: Core Metamodels [Transformations] Reasoners
        ReasonerSelectors Questions;
Core: [1,1]{Standalone OSGi};
Metamodels: [1,3]{FaMaFeatureMetamodel FaMaAttributedFM
AnotherFM};
Transformations: [1,2]{AtomicSet Attributed2Basic};
Reasoners: [1,5]{JavaBDD Sat4j Choco Reified Attributed};
ReasonerSelectors: [1,2]{Default OptimalSelector};
Questions: [1,23]{Valid Products Errors ValidProduct Explanation
[...]};

%Constraints
FaMaAttributedFM REQUIRES Attributed;
Explanation REQUIRES Reified;
```

## 3.2 Using FaMa Console

FaMa console is a recommended way to give the first steps to learn the scope and capabilities AAVM provides. You only need to download the last available distribution and execute the main Java jar file and the console will be automatically launched. Example 11.3 shows an example of interaction that loads the FM in Example 11.2, validates it and counts the number of products the FM describes.

**Example 11.3.** Interacting with FAMA FW through its console

```
C:>java -jar FaMaSDK-1.1.0.jar
Welcome to FaMa shell
$>load fama-fm.fama
Loading model...
Loaded!!
$>valid
Model is valid
$>#products
Number of products: 87240
```

## 3.3 Using FaMa Façade

FaMa FW is also a Java library that can be integrated in third-party tools through *FaMa Façade*. It hides FaMa FW insides offering a simple facade to interact in a question–answer manner. Since a change on it will make all the coupled products change, the facade must provide a stable set of interfaces that mainly consists of question interfaces and a feature meta-model at choice. One of the advantages of using this facade is that new versions of reasoners, reasoner selectors and meta-models may reach end-users with no adaptation on their applications since there is no need to couple to them.

**Example 11.4.** Java code to analyse the previous FM

```
// Instantiating FAMA Framework facade
QuestionTrader qt = new QuestionTrader();

// Loads a FM from a file
VariabilityModel fm = qt.openFile("fama-fm.fama");
qt.setVariabilityModel(fm);

// Validates the FM and then counts its products
ValidQuestion vq = (ValidQuestion)
                        qt.createQuestion("Valid");
qt.ask(vq);
if (vq.isValid()) {
```

```
    NumberOfProductsQuestion npq =
              (NumberOfProductsQuestion) qt.
                        createQuestion("#Products");
    qt.ask(npq);
    System.out.println("The number of products is: "
              + npq.getNumberOfProducts());
  } else {
    System.out.println("Your feature model is not
valid");
  }
```

The façade is also available as an OSGi service providing the same interface the facade does.

## 4 Results and Lessons Learned

After 4 years of development, 12 FaMa Extension projects and 7 products that are used by 20 institutions, we have learned many things regarding PL development [7]. Those conclusions that have surprised us most are summarised next and we expect they serve to build other SPLs:

- *PL is a growth concept rather than specific architectures*: Many people explore books for finding the silver-bullet architecture for PL. We do not have to search for brand new architectures for PL, but using whole-life architectures and apply innovative PL management procedures.
- *Analysing core and variant helped on defining a stable core*: We built FaMa FW aiming that transferring new results in AAVM does not mean delivering new and incompatible versions every month. A thorough study in AAVM commonalities and waiting for the right moment when we had all the needed background to make stable decisions helped on defining stable interfaces that have suffered no changes since their definition.
- *Deploying brand new features transparently to end-users is possible*: In our context there are much functionality that just improves the performance of our tool such as reasoners, transformations and reasoner selectors. We have defined a solution that delivers new features to end-users while their existing systems suffer no change at all.
- *Open-source and SPL are compatible concepts*: open-source helps on disseminating FaMa FW. Managing a SPL is complex; even more if third party contributors are involved in the project. Using an adequate architecture reduces coupling among projects and allow reducing the collateral effects new projects might produce. However, critical parts such as the core and facade must be under the control of only one party to ensure the maintainability of the SPL.
- *Using tools instead of processes*: dealing with a large amount of components is not a trivial task. Incorporating Maven and SVN have reduced a large amount of

errors we had in the beginning to synchronise project versions by hand. We have also used the own FaMa FW to build tools to manage our SPL such as FaMa Benchmark to compare the performance of our reasoners, FaMa Test Suite to test every reasoner prior a delivery and FaMa itself to analyse dependences among variant features.

## 5 Outlook

The area of automated analysis has recently reached 20 years of existence [2]. Although the area is mature enough and technology transfer for production is ready (FaMa is an example on this direction), some challenges remain open and will be explored and leveraged in the following years. One of them is the introduction and exploitation of attributes inside variability models such as cost, time, versions, etc. This will bring a new generation of automated analysis tools. FaMa already includes some of those features but some investigation and practical use cases are need to complement the current state of the practice.

We will see in the future also how analysis tools will be integrated into product line development ecosystems: the research community has mainly built variability model analysis tools as stand-alone prototypes that have hardly been integrated as part of large-scope CASE tools. FaMa has also made some steps forward but more will come in the future.

A key issue in variability model analysis tools is performance. The community is investigating this but new results will be released in the following years. The BeTTy framework is one of our main contributions in this direction. Refer to FaMa's web page at http://www.isa.us.es/fama for any further information.

## References


1. BeTTy framework. http://www.isa.us.es/betty/
2. Benavides, D., Segura, S., Ruiz-Cortés, A.: Automated analysis of feature models 20 years later: a literature review. Inf. Syst. **35**(6), 615–636 (2010)
3. OSGi Service Platform, Core Specification. OSG Alliance – OSGi Specification
4. Moskitt Feature Modeler. http://www.pros.upv.es/mfm/
5. Cetina, C., Pelechano, V., Trinidad, P., Ruiz-Cortés, A.: An architectural discussion on DSPL. In: SPLC (2), pp. 59–68. Lero International Science Centre, University of Limerick, Ireland (2008)
6. Trinidad, P., Ruiz-Cortés, A., Peña, J., Benavides, D.: Mapping feature models onto component models to build dynamic software product lines. In: SPLC (2), pp. 51–56. Kindai Kagaku Sha Co. Ltd., Tokyo, Japan (2007)
7. Trinidad, P., Muller, C., García-Galán, J., Ruiz-Cortés, A.: Building industry-ready tools: FAMA Framework and ADA. In: WASDeTT-3 (2010)