Proceedings of the 2nd International Workshop on Formal Methods and Analysis in Software Product Line Engineering in conjunction with the 15th International Software Product Line Conference (SPLC 2011)

Munich, Germany
August 26, 2011

Editors:
Martin Becker
David Benavides
Martin Leucker
Rick Rabiser
Karina Villela
Peter Y.H. Wong
Organizers

Program Committee Chairs:
David Benavides (Univ. Seville, Spain)
Martin Leucker (Univ. Luebeck, Germany)

Organizing Committee Chairs
Martin Becker (Fraunhofer IESE, Germany)
Rick Rabiser (Univ. Linz, Austria)
Karina Villela (Fraunhofer IESE, Germany)
Peter Y.H. Wong (Fredhopper, Netherlands)

Program Committee

S. Apel, Univ. Passau, Germany
E. Bagheri, Athabasca Univ., Canada
D. Batory, Univ. of Texas, USA
A. Bauer, NICTA, Australia
M. Broy, Univ. Munich, Germany
D. Clarke, KU Leuven, Belgium
A. Classen, Univ. of Namur, Belgium
F. de Boer, CWI, Netherlands
A. Gruler, Siemens, Germany
K. Havelund, NASA JPL, USA
P. Heymans, Univ. of Namur, Belgium
K. Larsen, Aalborg Univ., Denmark
T. Männistö, Helsinki Univ. of Techn., Finland
M. Mendonca, Univ. of Waterloo, Canada
D. Muthig, Lufthansa Systems, Germany
B. O'Sullivan, Cork Constraint Computation Centre, Ireland
A. Ruiz-Cortés, Univ. Seville, Spain
G. Schneider, Univ. Gothenburg, Sweden
D. Schmidt, SEI, USA
J. White, Virginia Tech, USA
Program

09:00 – 09:15 Welcome and Opening

09:15 – 10:30 Keynote

Prof. Alexander Felfernig. "Intelligent Techniques for Software Product Line Engineering"

10:30 – 11:00 Coffee break

11:00 – 12:30 Short paper presentations and discussion

- Jesús García-Galán, Pablo Trinidad, José A. Galindo and Antonio Ruiz-Cortés. Tool supported error detection and explanations on feature models

- Peter Wong, Nikolay Diakov and Ina Schaefer. Modelling Software Product Lines using HATS Approach – A Fredhopper Case Study

- José A. Galindo, Fabricia Roos-Frantz, Jesús García-Galán and Antonio Ruiz-Cortés. Extracting Orthogonal Variability Models from Debian Repositories

12:30 – 14:00 Lunch break

14:00 – 15:30 HATS Tutorial

Dr. Rudolf Schlatte. "Modeling, Executing, and Visualizing Distributed Adaptable Object Oriented Systems using the ABS Tool Suite."

15:30 – 16:00 Coffee break

16:00 – 17:00 Discussion Session: Future research directions and the future of FMSPLE

17:00 – 17:15 Wrap-Up and Summary
Keynote by Prof. Alexander Felfernig

Title: Intelligent Techniques for Software Product Line Engineering

Abstract: Feature models are used for representing variability in software product lines. In order to successfully support the management of such models we are in the need of intelligent techniques which allow effective development, maintenance, and analysis processes. This talk will focus on an overview of intelligent techniques for feature model development and maintenance. An overview of existing model-based synthesis and analysis approaches will be given. Furthermore, the latest research results and important issues for future research will be presented.

Short Bio: Alexander Felfernig is professor of Applied Software Engineering at the Graz University of Technology (TU Graz). In his research he focuses on intelligent methods and algorithms improving the accessibility, development, and maintenance of complex knowledge bases. Furthermore, Alexander is interested in the application of AI techniques in the Software Engineering context, for example, the application of decision and recommendation technologies to make software requirements engineering processes more effective. For his research he received the Heinz-Zemanek Award from the Austrian Computer Society in 2009. Alexander is co-founder and director of ConfigWorks, a company focused on the development of knowledge-based recommendation technologies.

Web page: www.felfernig.eu
HATS Tutorial: Modeling, Executing, and Visualizing Distributed Adaptable Object-Oriented Systems using the ABS Tool Suite.

[Extended Abstract]

Rudolf Schlatte
University of Oslo
rudi@ifi.uio.no

ABSTRACT
The tutorial provides a “hands-on” demonstration of the features of Core ABS and Delta-oriented programming and their relation to software product line engineering, as developed in the EU HATS project. Light on theory, we will focus on understanding the language and concurrency model, followed by tool demonstrations and development of a model of a simple software product line.

Categories and Subject Descriptors
H.4 [Information Systems Applications]: Miscellaneous

Keywords
Software Product Lines, ABS, specification languages, Tutorial

1. INTRODUCTION
The ABS language [1] is being developed in the HATS project as an object-oriented, executable modeling language. Its minimal core, Java-like notation, safe multiprocessing semantics and distributed nature make it suitable for as a specification and modeling language for the working programmer.

ABS is divided into Core ABS and Full ABS. Core ABS contains the basic language features such as datatypes, objects, communication semantics etc. Full ABS extends Core ABS with notions of temporal and spatial variability, and in general serves as a test bed and implementation platform for new and experimental language features.

2. SOFTWARE PRODUCT LINE ENGINEERING USING ABS
A software product line, in ABS, consists of a set of products. Each product is characterized by the presence or absence of certain features, with features possibly depending on or precluding each other. Products are generated from a core model.

Full ABS adds delta and feature modeling to the Core ABS language. Deltas are compositional code changes operating at the level of classes and methods. Features are semantically cohesive, large-scale changes to a core product, expressed as a set of deltas that should be applied.

Product-line engineering in ABS consists of specifying products as sets of features against a core code base, then implementing and testing the deltas that implement the features.

3. TOOL SUPPORT
Tool support, language development and work on case studies go hand-in-hand in the HATS project. While still under development, the tools have reached a usable state and are being constantly maintained and updated as the project progresses.

There are multiple ABS editing environments. An Eclipse plugin, accessible from the HATS tools website [2], features the standard outliner, syntax highlighting, cross-referencing and execution / debugging support. A mode for the Emacs text editor implements similar features.

ABS can be executed via translation to Java and execution on the Java Virtual Machine, or via simulation in the rewriting logic engine Maude, which also defines the formal semantics of the language. The Eclipse plugin also supports sequence diagram generation, recording and replaying of pseudo-random execution traces, manual scheduling of processes and debugging support.

4. ACKNOWLEDGMENTS
ABS language development and the tutorial are funded by the EU FP7 project HATS (FP7-231620) (http://hats-project.eu/).

5. REFERENCES
Tool supported error detection and explanations on feature models

Jesus García-Galán
Dept. Computer Languages and Systems
University of Seville
Avda. Reina Mercedes s/n, 41012, Seville, Spain
jegalan@us.es

Pablo Trinidad
Dept. Computer Languages and Systems
University of Seville
Avda. Reina Mercedes s/n, 41012, Seville, Spain
ptrinidad@us.es

José A. Galindo
Dept. Computer Languages and Systems
University of Seville
Avda. Reina Mercedes s/n, 41012, Seville, Spain
jagalindo@us.es

Antonio Ruiz-Cortés
Dept. Computer Languages and Systems
University of Seville
Avda. Reina Mercedes s/n, 41012, Seville, Spain
aruiz@us.es

Automated analysis of feature models (FM) is a field of interest in recent years. Many operations over FMs have been proposed and developed, and many researchers and industrial companies have adopted FMs as a way to express variability. This last makes more necessary having support to detect, explain and fix errors on FMs. The notation of FMs makes very easy to express variability, but makes hard detecting errors and find their cause manually. And these errors may cause the model does not express the variability what we want of it. Therefore, we need support to detect errors and find their causes.

The contribution of this paper is a method to detect errors in FMs, based on the concept of observation. We also present implementations of this approach and of an approach to explain errors, in FaMa Framework [1] tool.

To detect FM errors, firstly we have to identify the different error types and what it means each of them. Void FM error means that the FM does not represent any product, dead feature error means that a feature of the FM does not appear in any product, false optional error means that an optional feature appears in every product that its parent feature also appears, and wrong cardinality error means that one or more values of a set relationship cardinality are not reachable.

We can check for these errors in a intuitive way. For instance, to detect if a FM has dead features, we can calculate every product and check if each feature appears in, at least, one product. But further, we propose a method based on observations, it means, FM configurations associated with a specific element (feature or cardinality). Each type of error has its type of observation associated too. With an algorithm, we calculate the set of observations of a FM. Then, for each observation, we check if FM has at least one product. If not, we have found an error. For instance, dead feature observation sets its feature as selected. If the FM with a dead feature observation is not valid, it means the feature we are checking is dead.

When we have found the errors, explanations tell us what is the cause of each error. An explanation is a set of relationships that originates one or more errors. Changing or removing these relationships we can fix a error. However, explanations by themselves do not provide information about how to change the relationship. For instance, if an explanation about a dead feature is a mandatory relationship, we can turn it into a optional relationship, but the explanation does not tell us directly.

We have implemented observations and explanations approaches in FaMa Framework, a tool for the automated analysis of FMs. The observations approach implemented is the previously mentioned, while the explanations approach implemented is the one described by Trinidad et al. [3] [4]. With these approaches, we have detected errors in SPLoT FM repository [2], and we have obtained explanations for them also.

1. ACKNOWLEDGEMENTS

This work has been partially supported by the European Commission (FEDER) and Spanish Government under CITI project SETI (TIN2009-07366), and by the Andalusian Government under ISABEL project (TIC-2533) and THEOS project (TIC-5906).

2. REFERENCES

http://www.splot-research.org/.
Diversity is prevalent in modern software systems in order to adapt to their application contexts [5]. Additionally, software systems must evolve to meet changing requirements over time. This may require substantial changes to the software and often results in quality regressions. After a change in a software system, typically some work is needed in order to regain the trust of its users. The “Highly Adaptable and Trustworthy Software using Formal Models” (HATS) project aims at developing a formal model-centric software product line (SPL) development methodology [1, 5] for engineering SPLs that are subject to anticipated (spatial variability) or unanticipated (temporal variability) changes.

The HATS approach is designed as a formal methodology for SPL development [1]. The HATS methodology is a combination of the ABS language, a set of well-defined techniques and tool suite for ABS, and a formal methodology to bind them to specific steps in a SPL development process. ABS comprises a core language with specialised language extensions, each focusing on a particular aspect of modeling SPLs, while respecting the separation of concerns principle and encouraging reuse. For example, the Micro Textual Variability Language, based on Classen et al.’s TVL [3], expresses the variability of SPL at the level of feature models, while the Delta Modeling Language [4] captures variability of SPLs at the level of object behavior during the family engineering phase of the SPL engineering process.

The ABS tool suite [2] includes an ABS compiler front end, which takes a complete ABS model of the SPL as input, checks the model for syntax and semantic errors and translates it into an internal representation. The front end supports automatic product generation where variability of the SPL is resolved by applying a sequence of delta modules containing modifications to a core ABS model at compile time. Variability resolution is one of the main activities during the application engineering phase of the SPL. Different back ends translate the internal representation into Maude or Java, allowing ABS models to be executed and analyzed. The tool suite also includes a plug-in for the Eclipse IDE ([www.eclipse.org](http://www.eclipse.org)). The plugin provides an Eclipse perspective to SPL developers for navigating, editing, visualizing, and type checking ABS models, and an integration with the back ends, so that ABS models can be executed or simulated directly from the IDE.

We evaluate the HATS approach with respect to an industrial SPL case study based on the Fredhopper Access Server (FAS), a distributed web-based software system for Internet search and merchandising, developed by Fredhopper B.V. ([www.fredhopper.com](http://www.fredhopper.com)). In particular, we consider the replication system product line; the replication system ensures data consistency across the FAS deployment. We use this case study to evaluate the HATS approach with respect to the following criteria:

**Expressiveness** We evaluate the ABS language with respect to its practical language expressiveness. We investigate from the user’s perspective how readily and concisely ABS allows users to express program structures and behavior, and its capability to capture both spatial and temporal variabilities in SPLs.

**Scalability** We evaluate the ABS language with respect to the size and the complexity of the modeled system. It is important to provide mechanisms at the language level that permit separations of concerns, reuse and scalable analysis in SPLs.

**Usability** We evaluate the HATS approach with respect to its overall usability, focusing on the ease of adoption and learnability, and into account the tool support as well as the language’s syntax and semantics.

1. **REFERENCES**


Software Product-Line (SPL) engineering is about management of a set of similar software products from a set of reusable core assets [2]. Variability Models (VM) are used to describe the variability present in the different stages of an SPL. The automated analysis of variability models involves a large number of techniques and tools. Currently, there are a variety of commercial and open-source tools to automate the reasoning on VMs such as FaMa\textsuperscript{1}, FaMa-OVM\textsuperscript{2}, pure::variants\textsuperscript{3}, SEGOS\textsuperscript{4} are some of them.

Although there are several kinds of variability models, the majority of the research works on analysis of these models has focused on Feature Models (FM) and Orthogonal Variability Models (OVM). The former focuses on the domain and variability present in an SPL and the latter in the variability.

A well known problem in SPL community is the lack of real models available to researchers [1]. There exists references to real VMs, but those models are not public, probably this is mainly caused by the apprehension of the industry to reveal their business models. Real models are needed for providing an idea of program’s behaviour in a real scenario [5].

As other researchers[4], we looked for real variability models in the opensource community. We found that Debian based distributions define the variability model of their product line [3] in a description file. We propose using Debian repositories as a good source of realistic and complex OVM. These can be used as motivations input problems for those tools dealing with the automation of OVM models.

In OVM a variation point (VP) documents what can vary from one product to another, and a variant (V) documents how this variation point can vary. Every line of the Debian description file describes an attribute or a relation of a package. A package definition is composed of a set of attributes and their relations with other packages. In our work we describe a mapping to obtain an OVM from the the variability present in Debian based distributions. The models obtained using our proposal could have up to 28,000 variability elements (Ubuntu 8.04) and more depending on the distribution used.

Materials. In www.isa.us.es/materials/fmsple you can find the OVM model (in FaMa-OVM format) of the Ubuntu 8.04 distribution, and also a eclipse workspace with the sources, libraries and the implementation of the algorithm.

Acknowledgements
This work has been partially supported by the European Commission (FEDER) and Spanish Government under CYT projects SETI (TIN2009-07366), by the Andalusian Government under ISABEL project (TIC-2533) and THEOS (TIC-5906). And by Evangelischer Entwicklungsdienst e.V. (EED).

1. REFERENCES

---

\textsuperscript{1}www.isa.us.es/fama
\textsuperscript{2}www.isa.us.es/famaovm
\textsuperscript{3}www.pure-systems.com
\textsuperscript{4}www.software-productline.com/SEGOS-VM-Tool