HyVar: Scalable Hybrid Variability for Distributed Evolving Software Systems

HyVar

Scalable Hybrid Variability for Distributed Evolving Software Systems

Thomas Brox Røst[1], Christoph Seidl[2], Ingrid Chieh Yu[3], Ferruccio Damiani[4],

Einar Broch Johnsen[3], Cristina Chesta[5]

[1] Atbrox AS thomas@atbrox.com
[2] Technische Universität Braunschweig c.seidl@tu-braunschweig.de
[3] Universitetet i Oslo ingridcy@ifi.uio.no, einarj@ifi.uio.no
[4] Università di Torino damiani@di.unito.it
[5] Santer Reply SpA c.chesta@reply.it

Project start: 1/2/2015; **Project end:** 31/1/2018
Project Website: http://www.hyvar-project.eu/

Project consortium: Santer Reply SpA, Università di Torino, Technische Universität Braunschweig, Universitetet i Oslo, Atbrox AS, Magneti Marelli SpA

Summary of project objectives: (O1) To develop a Domain Specific Variability Language (DSVL) and tool chain to support software variability for highly distributed applications; (O2) to develop a cloud infrastructure that exploits software variability as described in the DSVL to track the software configurations deployed on remote devices and to enable (i) the collection of data from the devices to monitor their behavior; and (ii) secure and efficient customized updates; (O3) to develop a technology for over-the-air updates of distributed applications which enables continuous software evolution after deployment on complex remote devices that incorporate a system of systems; and (O4) to test HyVar's approach as described in the above objectives in an industry-led demonstrator to assess in quantifiable ways its benefits.

Summary of project results and current state: HyVar is approaching the end of the project and we are close to reaching all the objectives. In this paper we present the integrated tool chain, which combines formal reuse through Software Product Lines with common used industrial practices, and support the development and deployment of individualized software adaptations. We also describe the main benefits for the stakeholders involved.

Keywords: Software engineering, software maintenance, software evolution, software product lines, variability models, distributed software, over-the-air updates, OTA, data intensive systems, internet of things, cloud computing

1 Motivation and Approach

1.1 Software Evolution in the Automotive Domain

Evolution is a well-known problem in any software product family of non-trivial complexity. Over the lifespan of a product line, new features will be added, old features are deprecated, and separate code branches may be created to deal with spin-off products. As the code and customer base grows, the possible feature combinations used by various product instances soon becomes unwieldy.

In some domains, such as the automotive industry, all the possible variants of a car (make, model, base equipment, extras, etc.) can create a situation with a combinatorial explosion of feature combinations. This poses a challenge when maintaining the supporting software. Not only the new features must be catered for, but also all previous feature combinations for products that are still on the market and under support agreements. This can easily lead to bad software development practices, such as "clone and own", where code is copied between repository branches.

Regarding deployment of software updates in the automotive scenario, how does one ensure that a given car gets an update that is customized to its particular feature combination? Moreover, how can we also take, e.g., the driving characteristics of the car owner into account when updating the car software? Using traditional software development techniques, it is easy to end up with a software repository with lots of duplicated and overlapping code, where making any substantial change carries the risk of unforeseen down-the-line implications.

1.2 HyVar Solution

Within the HyVar project we take a two-fold approach towards tackling this problem, focusing on both the *development* and *deployment* aspect of maintaining complex software products.

On the development side, we have created a set of tools that use software product line (SPL) modeling techniques for a structured approach towards handling feature combinations [1, 2, 3]. These tools can either be adopted from scratch for new projects or be applied to existing projects with low adoption efforts. Moreover, our analysis tools can be applied to a project that is already suffering from previous clone-and-own development, so that differences and similarities of cloned products are automatically extracted and modeled [4]. This greatly simplifies the task of cleaning up from years of bad development practices.

On the deployment side, we introduce a cloud-based tool chain that complements our development tools [1]. The tool chain functions in an Internet of Things context, keeping tabs on a large number of connected devices and their feature configurations. A reconfiguration component detects whenever a software update must be applied,

either due to changes on the device (pull action) or changes on the software side (push action). Instead of taking the naïve approach of pushing the same monolithic update to each device, the tool chain will create a customized update for each device and only do the full recompile when it is deemed necessary. This greatly reduces the complexity and bandwidth required to do over-the-air updates for a device fleet. Also, as part of the supporting toolkit, we can do a static analysis of the resources required for running our tool chain on the cloud. For a realistic traffic pattern model, this removes a lot of the guesswork when doing cloud infrastructure cost estimation.

2 Application and Benefits

The approach has been applied to an automotive domain use, where the goals are to: 1. Derive a new product from an existing one or develop a Software Product Line from scratch; 2. Develop several software product lines with distributed teams keeping track of the dependencies; 3. Customize software deployment for a highly specific environment; 4. Derive a Software Product Line from existing products. For the stakeholders (e.g. car manufacturers and automotive software developers) in our automotive scenario there are several benefits associated with using our tool chain. These are described in detail below.

2.1 Software Product Line development using the HyVar Tool Chain

The existing product can be used as it is. Using delta modeling techniques to transform statecharts and/or source code, it is possible to start from an existing product.

New features are fully implemented and recorded in statecharts. The configuration differences between product branches are highly visible and explicit and much easier to communicate within the company. Moreover, the executable programs can be created directly from the statechart editor.

Living models. Since new features are both modeled and built from statecharts, there is a direct connection between the model and the final product.

Reduced code duplication and development time. Since code is generated from models, the amount of code duplication is reduced. The copy/paste approach towards software development is no longer needed.

2.2 Reducing risk in distributed software development projects

All interfaces are defined through MSPL feature model interfaces. This encourages both better encapsulation and a more structured approach towards feature interfaces and simplifies the distributed development.

Independent software product line. By the use of feature model interfaces of the HyVar tool chain, the new functionality can be planned and constructed independently from the rest of the software product line,

Feature encapsulation helps evolution. Better encapsulation makes it explicit which parts of the software system can be changed without introducing errors in the existing functionality.

Early detection of specification errors. Using feature model interfaces, it is possible to guarantee that only intended configurations can be created.

2.3 Personalized deployment from the cloud

Cloud-based infrastructure. One of the major benefits of using cloud services is that you only pay for the resources you use. This means that there is no need for an up-front data center investment and that the costs scale along with the number of users as the company grows. For customers who are wary of using public clouds, private cloud installations are also possible.

Simulation model of cloud resource requirements. With the traffic data collected by a car manufacturer stakeholder, it is possible to simulate the resources needed for the tool chain cloud infrastructure. This makes it possible to estimate the costs required for deploying the tool chain for a given fleet of cars even if there are certain peak periods.

Context changes can be reflected in the software configuration. Using validity formulas, context constraints and the HyVarRec reconfigurator, the software can be customized for a highly specific environment. This means that it is possible to build software updates that take the driving style and preferences of individual drivers into account.

2.4 Reducing code duplication in SPLs with variability mining

Automated analysis of existing software products. The differences and similarities of existing, cloned products can be analyzed almost completely automatically. From this, it is possible to generate feature models, mappings and delta modules that later can be used when adding new features or spinning off new product lines. The effort compared to doing this manually is greatly reduced.

Generated software product line elements. From the results of the analyses, the variability mining also generates suitable elements, such as delta modules, a technical feature model and, if needed, even an entire suitable delta language, e.g., for elements written in domain-specific languages. This overall greatly reduces the manual effort to set up an SPL from cloned variants.

Controlled restructuring. The process is not fully autonomous, meaning that developers have a lot of control over things such as feature naming and how the mined products should be restructured. They also have the opportunity to guide the restructuring through doing an iterative feedback/adaption process with the variability mining technology.

Increased abstraction between features and code. As the feature models are refined, so is the abstraction to the underlying code. This has long-term benefits in terms of both planning, discussing and working with a product as a combination of evolved features rather than just lines of code.

Conclusions

We have presented an innovative solution to the software reuse problem, integrating SPL engineering principles with existing tools and commonly used industrial practices. The HyVar approach supports the development and deployment of individualized software adaptations and realizes the concept of Hybrid Variability. The methodology and tool chain has been applied in a scenario from the automotive domain, and seems promising also for other emerging scenarios, such as Internet of Things (IoT) and Cyber-Physical Systems (CPS), characterized by a huge number of remote devices, each of whom has its own hardware configuration, runs a customizable distributed software application and needs to evolve in order to fix or prevent misbehavior, to adapt to environmental changes, accomplish new regulations, satisfy new user requests or meet new market expectations.

References

1. Cristina Chesta, Ferruccio Damiani, Liudmila Dobriakova, Marco Guernieri, Simone Martini, Michael Nieke, Vítor Rodrigues, and Sven Schuster. A Toolchain for Delta-Oriented Modeling of Software Product Lines, pages 497–511. Springer International Publishing, Cham, 2016.
2. Michael Nieke, Gil Engel, and Christoph Seidl. DarwinSPL: An integrated tool suite for modeling evolving context-aware software product lines. In Proceedings of the Eleventh International Workshop on Variability Modelling of Software-intensive Systems, VAMOS ’17, pages 92–99, New York, NY, USA, 2017. ACM.
3. Ferruccio Damiani, Michael Lienhardt, and Luca Paolini. A formal model for multi SPLs. In 7th IPM International Conference FSEN, Accepted for pubblication in Lecture Notes in Computer Science. Springer, Germany, 2017.
4. D. Wille, S. Schulze, C. Seidl, and I. Schaefer. “Custom-Tailored Variability Mining for Block-Based Languages”. In: Proc. of the Intl. Conference on Software Analysis, Evolution, and Reengineering (SANER). SANER ’16. IEEE, 2016.
5. Jacopo Mauro, Michael Nieke, Christoph Seidl, and Ingrid Chieh Yu. Context aware reconfiguration in software product lines. In Proceedings of the Tenth International Workshop on Variability Modelling of Software-intensive Systems, VaMoS ’16, pages 41–48, New York, NY, USA, 2016. ACM.