Introduction to the Track on Variability Modeling for Scalable Software Evolution

Ferruccio Damiani[1], Christoph Seidl[2(✉)], and Ingrid Chieh Yu[3]

[1] University of Torino, Turin, Italy
ferruccio.damiani@unito.it
[2] Technische Universität Braunschweig, Braunschweig, Germany
c.seidl@tu-braunschweig.de
[3] University of Oslo, Oslo, Norway
ingridcy@ifi.uio.no

Abstract. Information and communication technology today is increasingly integrated into the environment we live in, distributed on cars, appliances and smart infrastructures. The software running on these devices is increasingly individualized, adapted to the preferences and needs of the specific customer and must be able to evolve after deployment by means of software patches. Upgrades are becoming individualized; software patches used to upgrade the software are selected and adapted depending on the configuration and external constraints of the host device. The objective of the European project HyVar is to develop techniques and tools for fast and customizable software design, for the management of highly distributed applications, for continuous software evolution of remote devices, and scalable infrastructure to accommodate a large number of devices. The track *Variability Modeling for Scalable Software Evolution* aims to foster cooperation opportunities and create synergies between related research directions to address challenges stemming from software variability, evolution, and cloud technology for highly distributed applications in heterogeneous environments. This paper introduces the track and its individual contributions.

1 Context and Background

Software is an essential part of information and communication technology so that it is becoming increasingly integrated into our everyday environment, distributed on cars, appliances and a wide variety of devices. The struggle between the ideal fit of software resulting from individual development and the low cost of off-the-shelf software creates tension for developers and customers of software products alike. With the advent and rise of the Internet of Things (IoT) [2] and its devices (e.g., smartphones, tablets), the need for software that can be used on many similar yet slightly different devices and that can be individualized in

This paper contains an introduction to the ISoLA'16 track organized in the context of the EU H2020 project 644298 HyVar: Scalable Hybrid Variability for Distributed Evolving Software Systems (http://www.hyvar-project.eu).

T. Margaria and B. Steffen (Eds.): ISoLA 2016, Part II, LNCS 9953, pp. 423–432, 2016.
DOI: 10.1007/978-3-319-47169-3_35

functionality has further increased this tension towards favoring customizable software but still having to keep development costs at a reasonably low level. Furthermore, highly configurable software systems are a major asset in a wide range of areas from business software (e.g., SAP ERP[1] with its configurable modules for enterprise resource planning) to the transportation domain (e.g., cars with different on board electronics and specific integrated navigation systems desired by customers). Due to the sheer number of variants resulting from the configuration options, it is infeasible to develop, maintain or test all individual variations of the respective *software families* independent of one another and in isolation.

A Software Product Line (SPL) [18,21,22] is an approach to software reuse in the large where a set of related software systems is perceived as a software family consisting of a common core and variable parts often referred to as *features* [14]. A *product* or *variant* of the SPL is created by combining the common core with the functionality associated with a set of selected features. However, not all combinations of features form valid products, e.g., due to technical incompatibilities of the features' realization or due to business constraints that do not allow combining certain features. To define the principally valid constellations of features, a *variability model*, such as a *feature model* [14] is employed, which represents all valid *configurations* (sets of selected features) in a compact representation on a conceptual level. To create executable software system from this selection of conceptual features, a *variability realization mechanism* collects the realizations associated with each feature (e.g., source code or design models) and assembles them with the common core. Delta modeling [4,20] is a transformational variability realization mechanism that realizes variation of a software artifact by adding, modifying or removing parts of a software artifact in accordance with a feature's functionality, e.g., a feature might add certain methods to a class written in Java to realize additional functionality.

Modern software systems outgrow the scope of a traditional SPLs. When a software family consists of multiple SPLs, the software family may be managed by a Multi-software Product Line (MSPL) [10,12]. In a MSPL, several SPLs are composed in order to build a larger system of configurable components. These variable components need to be configured together to build a common system configuration but still depend on a common notation for a variability model. A Software Ecosystem (SECO) [5,26] is similar to an SPL or even a MSPL in the sense that it also manages a set of closely related software systems. However, a SECO is different from an SPL, in the sense that it does not have a variability model as central configuration knowledge and that multiple independent developers create and maintain the variable parts of the SECO.

SPLs, MSPLs and SECOs are subject to change over the course of time when their products have to adapt to altered or new requirements. This procedure is called *software evolution* [16] and poses a major challenge for SPLs, MSPLs and SECOs as not only single software systems but entire families of software systems have to be evolved [23,24]. Software evolution is especially difficult for SECOs

[1] http://go.sap.com/product/enterprise-management/erp.html.

where independent developers release new features or versions thereof in unsynchronized intervals and, possibly, without explicitly notifying other developers or users so that awareness of the current state of evolution of a SECO becomes a further challenge. As both configuration and evolution are sources of variability within the set of related software systems, it is also customary to denote them as *variability in space* and *variability in time*, respectively [18].

Due to the level of maturity of cloud technology and the wide variety of offered services, SPLs, MSPLs and SECOs become heavily based on cloud technology. Features may be realized as webservices [1] accessible by customers over the web and end-users may contribute features to shared platforms for various domains similar to apps for smartphones. In the automotive domain, utilizing the web for over-the-air update of entire products as well as individual features receives increasing attention.[2]

This combination of challenges stemming from configuration, evolution and cloud technology is at the center of the research conducted within the European Union H2020 project HyVar. To foster opportunities for cooperation on the topics of HyVar and to capitalize on synergies of related research directions, we organized the special track *Variability Modeling for Scalable Software Evolution* at the *International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA)*. This paper introduces the track and provides and overview of its sessions and their respective contributions.

2 The European Union H2020 Project HyVar

The EU H2020 project *HyVar* plans to integrate and enhance state-of-the-art techniques for the management of complex software systems from software product lines with cutting edge technology for over-the-air software upgrades and scalable cloud solutions from European industry to support highly individualized and reconfigurable distributed applications. HyVar's objectives are:

1. To develop a Domain Specific Variability Language (DSVL) and a tool chain to support software variability of highly distributed applications in heterogeneous environments, which allows developers to encompass unanticipated evolution as a standard feature of software systems in production.
2. To develop a cloud infrastructure that exploits the software variability supported by the DSVL and a tool chain to track the exact software configurations deployed on remote devices to enable the collection of data from the devices to monitor their behavior and perform statistical analyses.
3. To develop a technology for supporting over-the-air updates of distributed applications in heterogeneous environments and enabling continuous software evolution after deployment on complex remote devices that incorporate a system of systems.
4. To test HyVar's approach as described in the above objectives in an industry-led demonstrator in the automotive domain to assess in quantifiable ways the benefits of the approach.

[2] http://www.wired.com/2014/02/teslas-air-fix-best-example-yet-internet-things/.

HyVar aims to create a development framework for continuous and highly individualized evolution of distributed software applications, which can be integrated into existing software development processes. The framework, which is currently under development, will consist of advanced methods and tools that support

- modeling of both variability in space and time in all phases of the software lifecycle,
- scalable, elastic solutions to accommodate numerous individualized application instances, and
- secure and efficient over-the-air software update on remote devices.

This framework will be realized by combining variability modeling from SPL engineering with formal methods and software upgrades for distributed applications. HyVar goes beyond the state-of-the-art in devising and assessing the feasibility of the notion of *hybrid variability*, i.e., the automatic generation and deployment of software updates by relying on both

1. the variability model that describes the possible software variants that may be deployed to a remote device; and
2. the sensor data collected from that device.

The selection of features (and parameters) that will trigger the automatic generation and deployment of the most appropriate upgrades to a specific remote device depends on sensor data from that device (e.g., its location, and/or other things).

3 Track Papers

The ISoLA track *Variability Modeling for Scalable Software Evolution*, organized in the context of the EU H2020 project HyVar, is aimed at disseminating the results of the HyVar project and at promoting fruitful collaborations on its topics between researchers from academia and industry. Topics of special interest within the track are:

- Mobility, mobile and cloud.
- Methodologies, languages and tools.
- Research on variability retrieval, reconfiguration and refactoring.

The track consists of papers that directly relate to the core challenges of HyVar written by the consortium members [6,9,13,17,27] as well as papers addressing issues extending beyond the scope of HyVar written by other researchers [7,11,15,19,25] that contribute their insights to a joint effort of combining highly configurable software with customizable upgrades and flexible cloud technology for over-the-air distribution.

3.1 Keynote

Hähnle and Muschevici [11] outline an approach for formal modeling, simulation, and analysis of railway systems together with their requirements and interoperability constraints. The approach is based on the Abstract Behavioral Specification (ABS) language, which permits precise, executable specifications as basis for efficient code generation. ABS permits to trace system updates from requirements down to the implementation via delta modeling, thus allowing to analyse functional and non-functional properties that may have have changed. The specification of system updates in terms of SPL in ABS permits traceability of features down to code. Together with the analysis tool suite of ABS, this forms a feasible technological basis for an incremental verification and certification process.

3.2 Session 1: Mobility, Mobile and Cloud

Mobile systems, such as smartphones or the electronic devices found in cars, pose extensive challenges on software configuration and evolution. For one, this class of systems is usually characterized by a large degree of hardware heterogeneity that has to be addressed through software configurability for drivers and applications building upon the respective hardware, e.g., smarthpones by various vendors may have a different screen size and cars may have different engines or multimedia systems due to customer preferences, which the respective software has to respect. Moreover, this class of systems poses significant challenges in software evolution not only to frequent changes (e.g., updates to apps in smartphones) but also be the means of transporting respective software updates without having to call devices to service stations (e.g., performing updates to a car's software only in the garage). A cloud infrastructure is suitable for providing solutions to these challenges as mass-customized products may be supplied by scalable online servers and individualized updates from a version installed for one user's configuration to a newer version may be assembled and provided via over-the-air updates. The contributions of Session 1 deal with variability-aware design of services, designing and analyzing the architecture of highly configurable software systems as well as the conscious choice of cloud technology regarding the tradeoff between cost and performance, e.g., to create scalable cloud infrastructures for assembling and deploying updates of configurable software systems.

Ter Beek et al. [25] present a variability-based design of services for smart transportation systems. The addressed research topic is at the intersection of SPL engineering and machine learning. The key idea is to guide the design and implementation of an SPL by measuring the effectiveness of certain features in practice. The considered application domain is bike sharing, where the features correspond to various user-assisting prediction services that improve over time through machine learning. The investigation is based on concrete experiments with data concerning the bike-sharing system of the city of Pisa.

Khalilov et al. [15] model and optimize automotive electric/electronic architectures by making the variability-aware modeling language Clafer [3] more

accessible to practitioners. At present, software architectures, e.g., as used in the automotive domain, are nearing the point where the size and complexity of the design prevent software architects from making assessments on the impact of proposed changes. Tools of the variability-aware modeling language Clafer may, principally, be used to evaluate effects of design decisions but the Clafer language offers no dedicated support for modeling architectures. The authors present a Domain-Specific Language (DSL) on top of Clafer to model architectures, which embodies a reference architecture model and ensures that practitioners apply it adequately so that typical errors in the specification process can be caught early.

Johnsen et al. [13] compare different deployments on Amazon Web Services (AWSs) using model-based predictions. With the plethora of cloud services offered on the market today, it is challenging for a user to select a solution which best balances performance and incurred cost for a particular application. This paper builds upon ABS and Yet Another Resource Negotiator (YARN) by showing how the ABS-YARN framework enables users to assess the impact of different deployment decisions on the performance, operational cost, and workload completion of their software. Several workload scenarios are used to compare the cost-performance tradeoffs between different AWS on-demand resource purchasing options. The presented AWS instance study is based on MapReduce benchmarks of varied length, time requirements and distribution. Based on the simulation results, one may identify non-trivial tradeoffs early at the design phase of a software development process.

3.3 Session 2: Methodologies, Languages and Tools

To successfully employ, foster and maintain an SPL or a SECO of highly configurable software systems, a variety of methodologies, languages and tools are required. For both SPLs and SECOs, a tool chain is required that allows structured reuse within a family of related software systems by supporting the specification of a variability model, the derivation of variants according to selected configurations as well as the assembly and distribution of updates for the variants. For SECOs, a major challenge is to obtain on overview of the ongoing development efforts of loosely coupled contributors of extensions. For deployed variants running as individual software systems, runtime monitoring is a prerequisite to allow runtime adaptation for dynamic reconfiguration. The contributions of Session 2 introduce methodologies, languages and tools to cope with these challenges.

Chesta et al. [6] present a tool chain for delta-oriented modeling of SPLs. The paper addresses the challenge associated with large-scale reuse of software intensive systems by proposing an architecture and tool chain for customizing and managing variability modeling, derivation of product variants and scalable software repositories for distributed applications. The authors present a component-based architecture including a model variant generator, a state-diagram code generator, a source-code packager and a cross compiler as the main components. The paper demonstrates the overall approach through an industrial use case taken form the automotive domain.

Stănciulescu et al. [7] present a technology-neutral role-based collaboration model for Software Ecosystems (SECOs). Due to the independent development efforts and the lack of a central steering mechanism in SECOs, largely similar features may be developed multiple times by different developers, which increases effort and creates redundancy. The authors present remedy to this problem by contributing a role-based collaboration model for SECOs to make such implicit similarities explicit and to raise awareness among developers during their ongoing efforts, which fosters overview of the software ecosystem, analyses of duplicated development efforts and information of ongoing development efforts.

Rosà et al. [19] applies the DiSL framework for runtime monitoring on the Java Virtual Machine (JVM). DiSL is an aspect-oriented programming system specialized for dynamic program analysis which offers additional join points, pointcuts and advices. DiSL also uses partial evaluation in weaving and adaptive runtime instrumentation. The paper explains how runtime adaptation can be used for runtime monitoring and demonstrates the approach on an example using stationary field analysis. Benchmarks from the Da Capo suite[3] are used to evaluate the approach.

3.4 Session 3: Variability Retrieval, Reconfiguration and Refactoring

Choosing SPLs as a reuse strategy yields benefits in terms of reduced effort for creating products and increased product quality. However, adopting and maintaining an SPL strategy requires maintenance effort: *(i)* For adopting an SPL strategy, the product line has to be created, e.g., by retrieving it from a set of similar products that resulted from copying and then modifying an individual software system; *(ii)* For maintaining an SPL strategy, individual customer concerns regarding product functionality have to be addressed through flexible means of reconfiguration. Through the course of adapting an SPL to new or altered requirements, specific properties of the SPL have to be maintained, e.g., through refactoring. The contributions to Session 3 deal with retrieval, reconfiguration and refactoring of SPLs.

Wille et al. [27] identify variability in object-oriented code using model-based code mining. Companies often employ Object-Oriented Programming (OOP) languages to create variants of their existing software by copying and modifying individual products to changed requirements. While these so-called *clone-and-own* approaches allow to save money in short-term, they expose the company to severe risks regarding long-term evolution and product quality. The authors introduce a model-based approach to identify variability information for OOP code, which allows companies to better understand and manage variability between their variants. This information allows to improve maintenance of the variants and to transition from single variant development to the more elaborate reuse strategy of an SPL.

[3] http://www.dacapobench.org/.

Nieke et al. [17] incorporate user-preferences into the reconfiguration process for products of an SPL by presenting user profiles for context-aware reconfiguration in SPLs. Although user customization has a growing importance in software systems and is a vital sales argument, SPLs currently only allow user customization at deploy-time. The authors extend the notion of context-aware SPLs by means of user profiles, containing a linearly ordered set of preferences with priorities. Furthermore, they present a reconfiguration engine that checks the validity of the current configuration and, if necessary, reconfigures the SPL while trying to fulfill the preferences of the active user profile to provide the most suitable configuration.

Damiani and Lienhardt [9] refactor delta-oriented product lines to enforce guidelines for efficient type-checking. Ensuring type safety in an SPL (i.e., ensuring that all programs of the SPL are well-typed) is a computationally expensive task. Recently, five guidelines to address the complexity of type checking delta-oriented SPLs have been proposed by the same authors [8]. In this paper, the authors present algorithms to refactor delta-oriented SPLs to follow the five guidelines. Individual steps, complexity and correctness of the refactoring algorithms are stated.

References

1. Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., Zaharia, M.: A view of cloud computing. Commun. ACM **53**(4), 50–58 (2010)
2. Atzori, L., Lera, A., Morabito, G.: The internet of things: a survey. Comput. Netw. **54**(15), 2787–2805 (2010)
3. Bak, K., Diskin, Z., Antkiewicz, M., Czarnecki, K., Wasowski, A.: Clafer: unifying class and feature modeling. In: Software and Systems Modeling, pp. 1–35 (2014)
4. Bettini, L., Damiani, F., Schaefer, I.: Compositional type checking of delta-oriented software product lines. Acta Informatica **50**, 77–122 (2013). doi:10.1007/s00236-012-0173-z
5. Bosch, J.: From software product lines to software ecosystems. In: Proceedings of the 13th International Software Product Line Conference, SPLC (2009)
6. Chesta, C., Damiani, F., Dobriakova, L., Guernieri, M., Martini, S., Nieke, M., Rodrigues, V., Schuster, S.: A toolchain for delta-oriented modeling of software product lines. In: Proceedings of the International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA), ISoLA 2016, Heidelberg (2016)
7. Stănciulescu, Ş., Rabiser, D., Seidl, C.: A technology-neutral role-based collaboration model for software ecosystems. In: Proceedings of the International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA), ISoLA 2016, Heidelberg (2016)
8. Damiani, F., Lienhardt, M.: On type checking delta-oriented product lines. In: Ábrahám, E., Huisman, M. (eds.) IFM 2016. LNCS, vol. 9681, pp. 47–62. Springer, Heidelberg (2016). doi:10.1007/978-3-319-33693-0_4
9. Damiani, F., Lienhardt, M.: Refactoring delta oriented product lines to enforce guidelines for efficient type-checking. In: Proceedings of the International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA), ISoLA 2016, Heidelberg (2016)

10. Damiani, F., Schaefer, I., Winkelmann, T.: Delta-oriented multi software product lines. In: 18th International Software Product Line Conference, SPLC 2014, pp. 232–236 (2014)
11. Hähnle, R., Muschevici, R.: Towards incremental validation of railway systems. In: Proceedings of the International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA), ISoLA 2016, Heidelberg (2016)
12. Holl, G., Grünbacher, P., Rabiser, R.: A systematic review and an expert survey on capabilities supporting multi product lines. Inf. Soft. Technol. **54**, 828–852 (2012)
13. Johnsen, E.B., Lin, J.-C., Yu, I.C.: Comparing AWS deployments using model-based predictions. In: Proceedings of the International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA), ISoLA 2016, Heidelberg (2016)
14. Kang, K., Cohen, S., Hess, J., Novak, W., Peterson, A.: Feature-oriented domain analysis (FODA) feasibility study. Technical report, DTIC document (1990)
15. Khalilov, E., Ross, J., Antkiewicz, M., Markus Völter, K.C.: Modeling and optimizing automotive electric/electronic (E/E) architectures: towards makingclafer accessible to practitioners. In: Proceedings of the International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA), ISoLA 2016, Heidelberg (2016)
16. Lehman, M.M.: Programs, life cycles, and laws of software evolution. In: Proceedings of the IEEE (1980)
17. Nieke, M., Mauro, J., Seidl, C., Yu, I.C.: User profiles for context-aware reconfiguration in software product lines. In: Proceedings of the International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA), ISoLA 2016, Heidelberg (2016)
18. Pohl, K., Böckle, G., van der Linden, F.J.: Software Product Line Engineering - Foundations Principles and Techniques. Springer, Berlin/Heidelberg (2005)
19. Rosà, A., Zheng, Y., Sun, H., Javed, O., Binder, W.: Adaptable runtime monitoring for the java virtual machine. In: Proceedings of the International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA), ISoLA 2016, Heidelberg (2016)
20. Schaefer, I., Bettini, L., Bono, V., Damiani, F., Tanzarella, N.: Delta-oriented programming of software product lines. In: Bosch, J., Lee, J. (eds.) Software Product Lines: Going Beyond. LNCS, vol. 6287, pp. 77–91. Springer, Heidelberg (2010)
21. Schaefer, I., Rabiser, R., Clarke, D., Bettini, L., Benavides, D., Botterweck, G., Pathak, A., Trujillo, S., Villela, K.: Software diversity: state of the art and perspectives. STTT **14**(5), 477–495 (2012)
22. Schmid, K., Santana de Almeida, E.: Product line engineering. IEEE Softw. **4**, 24–30 (2013)
23. Seidl, C., Schaefer, I., Aßmann, U.: Integrated management of variability in space and time in software families. In Proceedings of the 18th International Software Product Line Conference (SPLC), SPLC 2014 (2014)
24. Svahnberg, M., Bosch, J.: Evolution in software product lines. J. Softw. Maint. Res. Pract. **11**(6), 391–422 (1999)
25. ter Beek, M., Fantechi, A., Gnesi, S., Semini, L.: Variability-based design of services for smart transportation systems. In: Proceedings of the International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA), ISoLA 2016, Heidelberg (2016)
26. van den Berk, I., Jansen, S., Luinenburg, L., Ecosystems, S.: A software ecosystem strategy assessment model. In: Proceedings of the Fourth European Conference on Software Architecture: Companion Volume, pp. 127–134. ACM (2010)

27. Wille, D., Tiede, M., Schulze, S., Seidl, C., Schaefer, I.: Identifying variability in object-oriented code using model-based code mining. In: Proceedings of the International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA), ISoLA 2016, Heidelberg (2016)