

Report from Dagstuhl Seminar 14172

Unifying Product and Software Configuration

Edited by

Krzysztof Czarnecki¹, Arnaud Hubaux², Ethan Jackson³,
Dietmar Jannach¹, and Tomi Männistö⁵

- 1 University of Waterloo, CA, kczarne@gsd.uwaterloo.ca
- 2 ASML – Veldhoven, NL, contact@ahubaux.com
- 3 Microsoft Research – Redmond, US
- 4 TU Dortmund, Germany, dietmar.jannach@tu-dortmund.de
- 5 University of Helsinki, FI

Abstract

Research on computer-supported configuration of customizable products and services is currently carried out in two main communities: one community is mainly focused on the configuration of hardware artifacts, the other one is interested in configurable software systems and software product lines. Despite the significant overlap in research interests, the fields have mainly evolved in isolation in different fields such as Artificial Intelligence, Constraint Programming and Software Engineering. Yet, the communities have produced results that are applicable across the communities. The trend of products becoming increasingly heterogeneous, i. e., consisting of hardware, software and services, is furthermore increasingly blurring the line between the configuration domains in practice.

This report documents the program and the outcomes of Dagstuhl Seminar 14172 “Unifying Product and Software Configuration”. The seminar gathered researchers and practitioners working on configuration problems. The seminar consisted of invited presentations and working group sessions covering various topics of software and product configuration including knowledge representation issues, automated reasoning and configuration management and had a particular focus on the industry perspective.

Seminar April 21–24, 2014 – <http://www.dagstuhl.de/14172>

1998 ACM Subject Classification I.2.4 Knowledge Representation Formalisms and Methods, D.2.13 Reusable Software, F.4.1 Mathematical Logic, Logic and constraint programming

Keywords and phrases Product Configuration, Software Product Lines, Configuration Management

Digital Object Identifier 10.4230/DagRep.4.4.20

1 Executive Summary


Krzysztof Czarnecki

Arnaud Hubaux

Ethan Jackson

Dietmar Jannach

Tomi Männistö

License  Creative Commons BY 3.0 Unported license

© Krzysztof Czarnecki, Arnaud Hubaux, Ethan Jackson, Dietmar Jannach, and Tomi Männistö

Customizable products are an integral part of most Business-to-Business (B2B) and Business-to-Consumer (B2C) markets. The fast-growing demand for mass-customization affects both tangible products (e. g., cars and mobile phones) and intangible products like software (e. g.,



Except where otherwise noted, content of this report is licensed under a Creative Commons BY 3.0 Unported license

Unifying Product and Software Configuration, *Dagstuhl Reports*, Vol. 4, Issue 4, pp. 20–35

Editors: Krzysztof Czarnecki, Arnaud Hubaux, Ethan Jackson, Dietmar Jannach, and Tomi Männistö



DAGSTUHL
REPORTS

Dagstuhl Reports
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

operating systems, Enterprise Resource Planning systems and mobile phones). To this end, companies use software *configurators* that provide automated support to tailor products to the requirements of specific customers or market segments. These configurators have been developed essentially in two threads of research: *Product Configuration* (PC) and *Software Configuration* (SC).

PC is the umbrella activity of assembling and customizing physical artefacts (e.g., cars or muesli) or services (e.g., insurances). Due to the inherent complexity of configuration problems, PC was one of the first large-scale application fields of artificial intelligence (AI), as it required both powerful knowledge-representation formalisms and efficient reasoning methods. The particular challenges of knowledge representation and reasoning in PC even led to the development of new AI techniques. Today, PC can be seen as one of the major fields in which AI-based technology found its way into industrial practice and is part of many industrial configuration systems.

Mostly independent of PC, the software engineering community was confronted with challenging configuration problems. A typical challenge is the design and implementation of software components that can be adapted and parameterized according to customer requirements and business or technical constraints. As in PC approaches, the goal is to save costs by assembling individualized systems from reusable components. These challenges are dealt with in different strands of software engineering, e.g. software product line engineering or self-adaptive systems.

Questions of knowledge representation and types of reasoning support have been investigated for many years in PC and SC. Interestingly, research in these two fields has been carried out so far mostly independently. Except in rare cases, researchers in both fields are often unaware of approaches that have been developed in the other community.

This fragmentation is observable in two particular dimensions: *knowledge representation* and *configuration reasoning*. Knowledge representation is concerned with the question of how to encode the domain knowledge, e.g., about the compatibility of different features of a configurable product, in a formal or machine processible way. Configuration reasoning covers various aspects of how to make inferences given a knowledge base (configuration model), specific user requirements or an existing configuration. Typical tasks include the automatic completion of a partial configuration or checking the consistency of a given configuration.

The seminar was organized around the following research questions:

- (RQ1) What classes of configuration problems exist?
- (RQ2) How are these problems modelled?
- (RQ3) What automated tasks are supported?
- (RQ4) How are these automated tasks implemented?

The seminar was structured into three main blocks: *Problem characteristics*, *Knowledge representation* and *Reasoning and tools*. Each block consisted of a number of introductory presentations on the topic, which were given by researchers from different subfields and the seminar participants from industry. These talks then served as a basis for discussions on commonalities, differences and possible synergies. These discussions were made in small working groups in break-out sessions and the results then synthesized in plenary meetings. To make these break-out sessions more effective, the seminar participants were asked to fill out a detailed questionnaire before the seminar.

Overall, the seminar featured more than a dozen introductory talks from academia and from industry. In general, the interest from industry was particularly encouraging and the seminar was attended by representatives and speakers, e.g., from IBM, SAP, Microsoft, Siemens and BigLever. The evening sessions were used by several seminar participants to give additional “lightning” talks, to share recent research results and dive deeper into technical aspects.

2 Table of Contents

Executive Summary

<i>Krzysztof Czarnecki, Arnaud Hubaux, Ethan Jackson, Dietmar Jannach, and Tomi Männistö</i>	20
--	----

Overview of Talks

Selected knowledge representation aspects <i>Michel Aldanondo</i>	23
Configuration reasoning is hard in general, but can be made efficient by exploiting the hierarchical structure of configuration problems <i>Conrad Drescher</i>	23
Problem Characteristics of Industrial Product Configuration <i>Andreas Falkner and Albert Haag</i>	24
Towards combining performance optimisation and constraint satisfaction in software configuration <i>Holger H. Hoos</i>	26
Configuration in Industrial Product Families <i>Lothar Hotz</i>	26
High-Level Languages for Configuration Modeling and Analysis <i>Eunsuk Kang</i>	27
Product Line Engineering Meets Product Line Operations <i>Charles Krueger</i>	28
Some Verification Problems in Automotive Configuration <i>Wolfgang Kuchlin</i>	28
Boolean reasoning requires smart propositional encodings <i>Daniel Le Berre</i>	29
Configuration Evolution <i>Leonardo Gresta Paulino Murta</i>	29
Configuration in Variability-Rich Software Ecosystems <i>Klaus Schmid</i>	30
Performance Prediction in the Presence of Feature Interactions <i>Norbert Siegmund</i>	31
A minimal introduction to product configuration <i>Juha Tiihonen</i>	32
Challenges of topological variability <i>Andrzej Wasowski</i>	33
Strategically Optimizing Product Portfolios <i>Patrick Wischniewski</i>	33
Summary of Closing Discussion	34
Participants	35

3 Overview of Talks

3.1 Selected knowledge representation aspects

Michel Aldanondo (University of Toulouse, France)

License © Creative Commons BY 3.0 Unported license
© Michel Aldanondo

Main reference M. Aldanondo, E. Vareilles, “Configuration for mass customization: how to extend product configuration towards requirements and process configuration,” *Journal of Intelligent Manufacturing*, 19(5):521–535, 2008.

The talk deals first with the elements that need to be modeled. First, three views (functional, physical and process) are shown; then the multi-level modeling idea is introduced. The need of using both discrete and continuous types of variables and constraints is then discussed. In some situations, the need to add some location aspects (ports, location constraints, ...) to the traditional bill-of-material (physical view) is described. The fact that the model of the problem can change during configuration (adding variables or variables set) is discussed and also the need for distributed modelling. Then the talk presents two key modeling ideas: either modeling the problem or modelling the solution space. Classical constraint based approaches are recalled as a problem modelling solution while less known solutions automata illustrates the solution space modelling idea.

References

- 1 P. Pitiot, M. Aldanondo, E. Vareilles, Concurrent product configuration and process planning : Some optimization experimental results. *Computers in Industry*, Vol. 65, pp. 610–621, 2014
- 2 A. Felfernig, G.E. Friedrich, D. Jannach, UML as domain specific language for the construction of knowledge-based configuration systems. *International Journal of Software Engineering and knowledge Engineering*, 10(4), 449–469, 2000.

3.2 Configuration reasoning is hard in general, but can be made efficient by exploiting the hierarchical structure of configuration problems

Conrad Drescher (SAP AG – Walldorf, Germany)

License © Creative Commons BY 3.0 Unported license
© Conrad Drescher

In my talk I give an overview of reasoning problems arising in configuration and the most common solution approaches. Problems discussed include:

- checking the consistency of a configuration
- computing valid domains for user choices in interactive configuration; explaining why some option is not available
- finding a valid / optimal / updated configuration
- proving properties about configuration models (model equivalence, ...)

Except for the first one all of the problems are computationally hard in general, an observation that has important implications for scalability of configuration reasoning. I also discuss key differences between the most important approaches to reasoning such as constraint propagation solvers, conflict-driven clause learning solvers for Boolean problems, compilation of problems to finite graphs (BDDs/MDDs), mathematical programming and local search.

I then discuss two state-of-the-art approaches for calculating valid domains. One is based on storing a maximally condensed version of the complete search tree of the configuration

problem (i. e., as an MDD) [1]. The other is based by incrementally exploring portions of the search tree on the fly [2].

Finally I argue that configuration problems as typically encountered in practice differ from general constraint satisfaction problems in that they have a low tree-width. This fact can be exploited in both approaches for computing valid domains as CSP with low tree-width are known to admit search trees and hence also BDDs/MDDs of polynomial size [3].


More generally, for the above mentioned reasoning problems low tree-width can in many cases be exploited to devise efficient algorithms.

References

- 1 J. Amilhastre, H. Fargier, P. Marquis, “Consistency Restoration and Explanations in Dynamic CSP – Application to Configuration”, *Artificial Intelligence*, 2002
- 2 C. Bessiere, H. Fargier, C. Lecoutre, “Global Inverse Consistency for Interactive Constraint Satisfaction”, *Proceedings of CP*, 2013
- 3 P. Jegou and C. Terrioux, “Hybrid Backtracking Bounded by Tree-Decomposition of Constraint Networks”, *Artificial Intelligence*, 2003

3.3 Problem Characteristics of Industrial Product Configuration

Andreas Falkner (Siemens AG, Austria), Albert Haag (SAP AG – Walldorf, Germany)

License  Creative Commons BY 3.0 Unported license
© Andreas Falkner and Albert Haag

A “product” can be anything a company offers for sale, both tangible (manufacturable) goods and intangible ones such as services, software, or projects. The product can also be an upgrade of something a customer already has. A configurable product is one that is not specified solely by its product designation. Product configuration is a step in a business process to establish a complete and correct specification of a configurable product. A predominant business process is sales, including associated manufacturing and/or assembly. Sales configuration is not meaningful without being able to give and guarantee both a price and an availability date. It also affects the entire logistics supply chain. For example, production pre-planning will also need to be based on the history of sold configurations. After a sale is completed it may be necessary to configure the specific product instance sold to make it operational (such as an airplane or computer server). This after-sales configuration poses distinct challenges and will tend to be product specific. Various terms are in use to characterize different ways of dealing with sales configuration. Some examples are Pick-to-Order, Assemble-to-Order, Make-to-Order, and Engineer-to-Order. These aim mainly to distinguish two business relevant aspects:

- to what degree the product is assembled at the company’s site and the customer’s site respectively
- to what degree the product has been standardized. Standardization enables a streamlined fulfillment process. In the worst case, manual intervention by engineers is necessary in non-standard cases. (However, note that product design is considered outside the scope of configuration.)

Sales configuration poses three somewhat different tasks:

- High-level configuration is the interactive configuration dialog with the customer/sales person. Besides needing an underlying sales model for the product, this will depend on sales organization data and an availability date.

- Completion derives additional properties/components needed for manufacturing and at the same time ensures that the configuration can be manufactured as ordered for the given date and at what cost. This is usually non-interactive and will depend on a manufacturing (engineering) model of the product, and the particular manufacturing plant(s) as well as the effective date of manufacture.
- Low-level configuration (or Bill-of-Materials (BoM) explosion) is selecting the actual parts (BoMs) and operations (route sheets/routings) needed for manufacturing and reserving corresponding resources. Again, this process depends on plant and effective date.

It is a current topic of research whether all three tasks might be handled using a uniform approach (such as SAT-solving). Practical experience in the last decades suggests separation. This is also vindicated by the fact that different parts of the organization have responsibility for the different tasks. The first task is the responsibility of the sales organization. They will want to influence the model to some degree. The last task is fulfillment and mainly the responsibility of the manufacturing engineers. The completion task joins the two.

Often high-level configuration must primarily provide decision support rather than just solving under given constraints. A sales configuration will typically be under-constrained with respect to actual hard constraints, but poses an embedded multi-criteria optimization problem in determining a best configuration. Optimality is not expressed explicitly, but in the form of soft constraints or desirable properties that should be fulfilled in a good solution. When adding these soft constraints the problem becomes over-constrained. Which properties to forego is ultimately a decision of the user. The requirements for a user interface (UI) depend on the type of user. In a B2C scenario the user is non-expert and requires very explicit visualization of alternatives/aid in resolving inconsistencies. In a B2B or in-house scenario the users are more expert and it may be sufficient to simply alert them to inconsistencies and incompleteness.

Problems can occur in the context of upgrades or after-sales configuration that may require dedicated CSP or SAT-solving approaches. The complexity of the first two sales configuration tasks is determined mainly by how the product is represented at that level. The easiest way would be as a single component with a manageable number of specifiable characteristics (attributes). An example of a product presented this way is a car (50-70 characteristics presented in the high-level configuration, 150-300 in completion, tens of thousands of components in low-level configuration). At the other end of the spectrum is a complex multi-level system with an a priori indeterminate number of sub-components. Examples of this would be elevators, busses, or computer servers.

Since a sales contract is legally binding, care must be taken that a configuration that is accepted is complete and correct and can be delivered at the promised date and price. Thus tools for model verification, testing and debugging are very important. Limiting the complexity of the product is essential in achieving this at reasonable cost. An exact methodology for measuring the complexity of a configurable product project from the business perspective is still lacking.

3.4 Towards combining performance optimisation and constraint satisfaction in software configuration

Holger H. Hoos (University of British Columbia – Vancouver, Canada)

License © Creative Commons BY 3.0 Unported license
© Holger H. Hoos

Joint work of Hoos, Holger H.; Hutter, Frank; Leyton-Brown, Kevin
Main reference H. H. Hoos, “Programming by Optimization,” *Communications of the ACM*, 55(2):70–80, February 2012.

URL <http://dx.doi.org/10.1145/2076450.2076469>

My group works on automatically configuring software for the purpose of performance optimisation. This is very widely applicable in industry (e. g., mixed integer programming – CPLEX, scheduling, SAT-based hardware and software verification, machine learning) and academia. There are several classes of techniques available for carrying out these configuration tasks, the best of which have been demonstrated to work on configuration problems involving up to about 750 parameters. I believe that sequential model-based techniques, like our own SMAC procedure, are particularly promising, especially in cases where performance evaluations are costly and therefore few can be completed over the course of the configuration process. I see very significant potential for these kinds of techniques to fundamentally change the way performance-critical software will be designed, towards much more configurable systems than used currently. This is at root of the Programming by Optimisation (PbO) software design paradigm developed and promoted by my group (see main reference provided above).

There is a different notion of configuration problems where the focus is on finding configurations that satisfy potentially complex constraints. These can be tackled with SAT and CSP solvers, which should be automatically configured using the previously mentioned techniques to perform well on the specific configuration problems they are being used on.

I see interesting potential in combining the two aspects of configuration mentioned above: performance optimisation within a potentially large space of configurations satisfying given constraints.

3.5 Configuration in Industrial Product Families

Lothar Hotz (HITeC e. V. / Universität Hamburg, Germany)

License © Creative Commons BY 3.0 Unported license
© Lothar Hotz

The software product line (SPL) approach provides a general reference process for supporting reuse of software components. This process is divided into domain engineering and application engineering. In domain engineering reusable components are developed and implemented that can be used in multiple applications. During application engineering these components are selected, configured, and composed to form a particular application. However, SPL provides a general schema, how the engineering subtasks can be resolved is matter of research.

Knowledge-based configuration as a field of Artificial Intelligence provides modeling languages and reasoning tools that enable the task of composing a system from components [1]. As such, knowledge-based configuration supplies technologies that support the task of application engineering.

The ConIPF methodology [2] demonstrates a successful application of knowledge-based configuration technologies for solving the application engineering task of SPL. The ConIPF methodology enhances the reference process by configuration activities such as development of a configuration model and running the configuration process. As a difference to hardware configuration, the ConIPF methodology adds activities to the process that create (compile, link, test) configured software (sub-)systems during the configuration process. The ConIPF methodology was applied to construct software-intensive systems in the field of car manufacturing.

References

- 1 Felfernig, A., Hotz, L., Bagley, C., Tiihonen, J. (Eds.), Knowledge-based Configuration – From Research to Business Cases. Morgan Kaufmann Publishers, 2014.
- 2 Hotz, L., Wolter, K., Krebs, T., Deelstra, S., Sinnema, M., Nijhuis, J., MacGregor, J., 2006. Configuration in Industrial Product Families – The ConIPF Methodology. IOS Press, Berlin.

3.6 High-Level Languages for Configuration Modeling and Analysis

Eunsuk Kang (MIT – Cambridge, USA)

License © Creative Commons BY 3.0 Unported license
 © Eunsuk Kang
URL <http://people.csail.mit.edu/eskang/talks/dagstuhl-configuration.pdf>

In this talk, I will introduce two modeling languages, Alloy (developed at MIT) and Formula (Microsoft Research), and describe how they can be used to model and analyze a variety of configuration problems.

Alloy is an expressive modeling language based on first-order relational logic [1]. Originally designed for describing complex structures that arise in software systems, it has been applied to a variety of applications, including requirements analysis, program verification, policy modeling, and security protocols. It has also been used to solve different types of configuration problems, including product lines, feature models, multi-objective optimization, and configuration synthesis. The analysis in Alloy is done by translating an original FOL formula to an equisatisfiable SAT formula, which is then handed off to a third-party SAT solver. When the solver finds an instance, it is translated back to a high-level representation in the original model. If no instance is found, a minimal unsatisfiable core is generated to highlight the parts of the model are contradictory; this feature can be used to produce an explanation for configuration problems.

FORMULA is a modeling language developed at Microsoft Research for model-driven architecture development [2]. Based on logic programming (stratified horn clauses), FORMULA provides expressive constructs for modeling and composing domain abstractions. Its analysis is done by translation to the Z3 SMT solver, which is capable of handling a variety of theories (arithmetic, arrays, etc.). FORMULA has been used in a number of applications, including exploring the design space of automobile architectures [3].

References

- 1 Official page for Alloy (<http://alloy.mit.edu>)
- 2 Official page for FORMULA (<http://research.microsoft.com/en-us/projects/formula>)
- 3 Eunsuk Kang, Ethan K. Jackson, Wolfram Schulte: An Approach for Effective Design Space Exploration. Monterey Workshop 2010:33–54.

3.7 Product Line Engineering Meets Product Line Operations

Charles Krueger (BigLever, Austin, USA)

License © Creative Commons BY 3.0 Unported license
© Charles Krueger

URL http://www.biglever.com/newsletters/Edge_PLE_Envelope_Part2.html

The complexity of managing the variability for a family of similar products or systems is not limited to product line engineering (PLE) organizations. Other organizations that can spend inordinate amounts of time and effort dealing with product feature diversity include manufacturing and supply chains in automotive, certification and compliance documentation in aerospace and defense, portfolio planning in highly competitive markets, web system deployments in e-commerce, and sales automation for complex configurable systems.

Although it became clear to many successful PLE organizations that alignment of PLE with their existing business operations was crucial, the idea of consolidating the variant management and configuration disciplines across engineering and operations groups is an emerging idea at the edge of the applied research envelope. Some of the industry's most innovative product line enterprises are now leveraging their PLE competence to create highly efficient Product Line Operations. We refer to this convergence as Product Line Engineering and Operations, or PLE&O.

PLE&O is more than just a new approach for aligning PLE with business operations, it is a generational step forward in the evolution of product line paradigms. PLE&O extends PLE with fundamental new perspective and methodology, with consolidated Feature Ontology and configuration automation.

3.8 Some Verification Problems in Automotive Configuration

Wolfgang Kuchlin (Universität Tübingen, Germany)

License © Creative Commons BY 3.0 Unported license
© Wolfgang Kuchlin

Joint work of Kuchlin, Wolfgang; Sinz, Carsten; Zengler, Christoph; Walter, Rouven
Main reference C. Sinz, A. Kaiser, W. Kuchlin, "Formal methods for the validation of automotive product configuration data," *AI EDAM: Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 17(1):75–97, 2003.

URL <http://dx.doi.org/10.1017/S0890060403171065>

Automotive production is based on product configuration with very high variance, especially for German premium car manufacturers. Vehicle configuration is structured in two levels. High-level configuration (HLC) is concerned with the configuration of customer car orders from sales options such as motors, seats, etc. Low-level configuration (LLC) is concerned with the selection of the necessary parts for a car order from the bill-of-materials (BOM), which is the list of all parts necessary for an entire line of cars. Documentation of both HLC and LLC is usually based on Boolean logic. For a number of years, we have successfully shipped verification systems based on SAT-solving to the automotive industry.

Some verification issues concerned with HLC are the computation of options which are necessary, possible, or impossible, for every car order. Verification of LLC is concerned with the computation of BOM materials (parts or software) which can never be used in any order, which would be missing for some orders, or which would be multiply selected for some orders.

More recent issues include e. g., model counting the number of car orders in the HLC, the explanation of proof results, or the reconfiguration of orders.

References

- 1 Wolfgang Küchlin and Carsten Sinz. Proving consistency assertions for automotive product data management. *J. Automated Reasoning*, 24(1–2):145–163, February 2000. (Special issue: Satisfiability in the Year 2000).
- 2 Andreas Kübler, Christoph Zengler, and Wolfgang Küchlin. Model counting in product configuration. In Inês Lynce and Ralf Treinen, editors, *Proc. First Int’l Workshop on Logics for Component Configuration (LoCoCo)*, volume 29 of *EPTCS*, pp. 44–53, 2010.
- 3 Carsten Sinz, Andreas Kaiser, and Wolfgang Küchlin. Formal methods for the validation of automotive product configuration data. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 17(1):75–97, January 2003. Special issue on configuration.
- 4 Christoph Zengler and Wolfgang Küchlin. Boolean quantifier elimination for automotive configuration – a case study. In Charles Pecheur and Michael Dierkes, editors, *Formal Methods for Industrial Critical Systems – 18th Int’l Workshop, FMICS 2013*, volume 8187 of *LNCS*, pp. 48–62. Springer, 2013.

3.9 Boolean reasoning requires smart propositional encodings


Daniel Le Berre (Artois University, Lens, France)

License  Creative Commons BY 3.0 Unported license
© Daniel Le Berre

Boolean reasoning has been used both in product and software configuration, with both success and failure stories. Encoding a problem into a Boolean satisfaction or optimization problem requires a lot of expertize: there are numerous ways to translate high level constraints into clauses, and intermediate solutions based on custom constraint propagators do exist. A great encoding is typically not a Boolean model of the initial problem but a way to describe a problem specific propagator for a Boolean engine. The choice of the Boolean input language (SAT, MAXSAT, Pseudo-Boolean Optimization) as well as the Boolean engine used may also have a deep impact on performances. It is thus important to model those problems into a high level input language such as ASP (Answer Set Programming), Alloy, MiniZinc, Copris/Scarab and to reuse state-of-the-art translators to produce Boolean formulas.

3.10 Configuration Evolution

Leonardo Gresta Paulino Murta (Federal University Fluminense – Niteroi, Brazil)

License  Creative Commons BY 3.0 Unported license
© Leonardo Gresta Paulino Murta

Both Product and Software Configuration can and do evolve over time. However, low attention is being provided to this problem. The discipline of Configuration Management can help to shed some light on this subject. In this talk, we introduce some basic Configuration Management concepts, discuss why general purpose Version Control Systems provide poor support for controlling evolution of more elaborate artifacts, and discuss some challenges of versioning Product and Software Configurations.

3.11 Configuration in Variability-Rich Software Ecosystems

Klaus Schmid (Universität Hildesheim, Germany)

License © Creative Commons BY 3.0 Unported license
© Klaus Schmid

Joint work of Schmid, Klaus; Eichelberger, Holger; El-Sharkawy, Sascha; Kroeher, Christian; Brummermann, Hendrik; Keunecke, Markus

Main reference K. Schmid, “Variability Support for Variability-Rich Software Ecosystems,” in Proc. of the 4th Int’l Workshop on Product Line Approaches in Software Engineering (PLEASE’13), pp. 5–8, IEEE, 2013.

URL <http://dx.doi.org/10.1109/PLEASE.2013.6608654>

While software product lines are a way to support a single organization to create a range of products from common assets with variability, an ecosystem involves a number of organizations that produce software and services, which mutually enrich each other. In an ecosystem the final “system” is only created at a later point, when the decision is made which software from which organizations will be combined to form the final system. Of course, both situations may happen simultaneously: each – or at least some – organization may employ a product line approach. This is what we call a *variability-rich software ecosystem* [1].

In such an ecosystem it is important to describe the composition of the individual parts as well as the variability of the individual product lines, both of which can be seen as forms of configuration. Moreover, the situation of an ecosystem leads to additional requirements for configuring each individual product line. Examples for this are that some sort of default modeling should be supported [2] so that each composition results in a complete configuration, some sort of modularization should be supported (e. g., like in CVL [3], using interfaces), and so forth. Besides the demands such a situation creates for the configuration itself, it also creates demands on the way the instantiation is performed (e. g., support for partial instantiation).

As a reaction to these demands, we created the EASy-Producer tool [4]. This supports the configuration and instantiation of variability-rich ecosystems. As part of this effort a specific variability modeling language (IVML) and a variability instantiation languages (VIL) were developed and implemented.

References

- 1 K. Schmid. Variability Support for Variability-Rich Software Ecosystems, 4th International Workshop on Product Line Approaches in Software Engineering (PLEASE) at the International Conference on Software Engineering (ICSE), 5–8, 2013.
- 2 H. Brummermann, M. Keunecke, K. Schmid. Formalizing distributed evolution of variability in information system ecosystems. Proceedings of the 6th Workshop on Variability Modeling of Software-Intensive Systems (VaMoS ’12), ACM, 11–19, 2012.
- 3 CVL. Common variability language, 2012. <http://www.omgwiki.org/variability/doku.php?id=start&rev=1351084099>, Online accessed: 25.06.2014.
- 4 K. Schmid and E. Almeida. Product Line Engineering. IEEE Software, Vol. 30, No. 4, 24–30, 2013.

3.12 Performance Prediction in the Presence of Feature Interactions

Norbert Siegmund (*Universität Passau, Germany*)

License © Creative Commons BY 3.0 Unported license
© Norbert Siegmund

Joint work of Norbert Siegmund, Sergiy Kolesnikov, Christian Kästner, Sven Apel, Don Batory, Marko Rosenmüller, Gunter, Saake

Main reference N. Siegmund, S. S. Kolesnikov, C. Kästner, S. Apel, D. Batory, M. Rosenmüller, G. Saake, “Predicting Performance via Automated Feature-Interaction Detection,” in Proc. of the 34th Int’l Conf. on Software Engineering (ICSE’12), pp. 167–177, IEEE, 2012.

URL <http://dx.doi.org/10.1109/ICSE.2012.6227196>

Customizable programs and program families provide user-selectable features allowing users to tailor the programs to the application scenario. Beside functional requirements, users are often interested in non-functional requirements, such as a binary-size limit, a minimized energy consumption, and a maximum response time.

In our work, we aim at predicting a configuration’s non-functional properties for a specific workload based on the user-selected features [2, 3]. To this end, we quantify the influence of each selected feature on a non-functional property to compute the properties of a specific configuration. Here, we concentrate on performance only.

Unfortunately, the accuracy of performance predictions may be low when considering features only in isolation, because inaccurate predictions. many factors influence performance. Usually, a property is program-wide: it emerges from the presence and interplay of multiple features. For example, database performance depends on whether a search index or encryption is used and how both features interplay. If we knew how the combined presence of two features influences performance, we could predict a configuration’s performance more accurately. Two features interact (i. e., cause a performance interaction) if their simultaneous presence in a configuration leads to an unexpected performance, whereas their individual presences do not.

We improve the accuracy of predictions in two steps: (i) We detect which features interact and (ii) we measure to what extent they interact. In our approach, we aim at finding the sweet spot between prediction accuracy, measurement effort, and generality in terms of being independent of the application domain and the implementation technique. The distinguishing property of our approach is that we neither require domain knowledge, source code, nor complex program-analysis methods, and our approach is not limited to special implementation techniques, programming languages, or domains.

Our key idea to determine which features interact is the following: We measure each feature twice. In the first run, we try to measure the performance influence of the feature in isolation by measuring the variant that has the smallest number of additionally selected features. The second run, aims at maximizing the number of features such that all possible interactions that may influence on performance materialize in the measurement. If the influence of the feature in isolation differs with the influence when combined with other features, we know that this feature interacts. In the second step, we perform several sampling heuristics, such as pair-wise sampling, to determine the actual combinations of interacting features that cause interactions.

Our evaluation is based on six real-world case studies from varying domains (e. g., databases, encoding libraries, and web servers) using different configuration techniques. Our experiments show an average prediction accuracy of 95 percent, which is a 15 percent improvement over an approach that takes no interactions into account [1].

References

- 1 N. Siegmund, S. Kolesnikov, C. Kästner, S. Apel, D. Batory, M. Rosenmüller, and G. Saake. Predicting Performance via Automated Feature-Interaction Detection. In *Proc. ICSE*, pp. 167–177. IEEE, 2012.
- 2 N. Siegmund, M. Rosenmüller, C. Kästner, P. Giarrusso, S. Apel, and S. Kolesnikov. Scalable Prediction of Non-functional Properties in Software Product Lines. In *Proc. SPLC*, pp. 160–169. IEEE, 2011.
- 3 N. Siegmund, M. Rosenmüller, C. Kästner, P. Giarrusso, S. Apel, and S. Kolesnikov. Scalable Prediction of Non-functional Properties in Software Product Lines: Footprint and Memory Consumption. *Information and Software Technology*, 55(3):491–507, 2013.

3.13 A minimal introduction to product configuration

Juha Tiihonen (Aalto University, Finland)

License  Creative Commons BY 3.0 Unported license
© Juha Tiihonen

A minimal history of product configuration includes rule-based configurators, early model-based configurators, mainstream configuration environments and mass customization toolkits [4]. The main ideas of product configuration modeling [2, 5]) cover connection-based, resource-based, structure-based, and function-based approaches. The related concepts are treated uniformly in an object oriented manner with the availability of taxonomic hierarchies with refinement, abstraction, and applicability of attributes. An integral part of product configuration modeling is supporting the variable compositional structure of components and functions/features. Common feature modeling concepts of basic, cardinality based and extended feature models including “complex” constraints [1] can therefore be easily mapped to concepts of the product configuration community. A demonstration with product configuration system WeCoTin [6] showed how the example model of [1] can be modeled and configured. Research challenges include personalized configuration, unification of configuration and feature models, community-based configuration, standardized configuration knowledge representations, intelligent user interfaces for configuration knowledge acquisition, intelligent testing and debugging, unobtrusive preference elicitation, and processes for intelligent systems development [3]. It is concluded that product configuration has a long and successful history and product configurators are applied relatively widely. Product configuration modeling techniques can be directly applied for representing many if not most feature models. It seems that many aspects of variability modeling from the product configuration community could be carried from product configuration community to software configuration community. However, management of variability is just one aspect of software product family modeling.

References

- 1 Benavides, D., Segura, S., & Ruiz-Cortes, A. (2010). Automated analysis of feature models 20 years later: A literature review. *Information Systems*, 35(6), 615-636. DOI: 10.1016/j.is.2010.01.001
- 2 Felfernig, A. (2007). Standardized configuration knowledge representations as technological foundation for mass customization. *Engineering Management, IEEE Transactions On*, 54(1), 41–56. DOI: 10.1109/TEM.2006.889066
- 3 Felfernig, A., Hotz, L., Bagley, C., & Tiihonen, J. (2014). Chapter 15 – Configuration-Related Research Challenges. In A. Felfernig, L. Hotz, C. Bagley & J. Tiihonen

- (Eds.), Knowledge-Based Configuration (pp. 191–195). Boston: Morgan Kaufmann. DOI: 10.1016/B978-0-12-415817-7.00015-3
- 4 Hotz, L., Felfernig, A., Günter, A., & Tiihonen, J. (2014). Chapter 2 – A Short History of Configuration Technologies. In A. Felfernig, L. Hotz, C. Bagley & J. Tiihonen (Eds.), Knowledge-Based Configuration (pp. 9–19). Boston: Morgan Kaufmann. DOI: 10.1016/B978-0-12-415817-7.00002-5
 - 5 Soininen, T., Tiihonen, J., Männistö, T., & Sulonen, R. (1998). Towards a general ontology of configuration. *AI EDAM*, 12(4), 357–372.
 - 6 Tiihonen, J., Heiskala, M., Anderson, A., & Soininen, T. (2013). WeCoTin – A practical logic-based sales configurator. *AI Communications*, 26(1), 99–131. DOI: 10.3233/AIC-2012-0547

3.14 Challenges of topological variability

Andrzej Wasowski (IT University of Copenhagen, Denmark)

License © Creative Commons BY 3.0 Unported license
© Andrzej Wasowski

Main reference T. Berger, S. Stanculescu, O. Ogaard, O. Haugen, B. Larsen, A. Wasowski, “To Connect or Not to Connect: Experiences from Modeling Topological Variability,” to appear in the Proc. of the 18th Int’l Software Product Line Conf. (SPLC’14).

Classic variability models are non-structural. Both feature models and decision models focus on capturing sets of parameters, their names and dependencies between them. These are then used to configure the piecemeal of software in question. One well known publicly available example is the Linux kernel, having a configurator driven by a simple variability model.

In installation engineering, a bit differently than in software, there is need for modelling component types, and their connections (topologies). Configurators derived from such models are used by installation engineers to design specifications for particular deployments. I present the problem using the example of fire alarm systems of a Norwegian vendor, Autronica. The presentation explains the shortcomings of variability models known from software product lines area for specifications of such systems.

This work has been funded by the ARTEMIS project VARIES on variability in safety critical systems.

3.15 Strategically Optimizing Product Portfolios

Patrick Wischnewski (Logic4Business – Saarbrücken, Germany)

License © Creative Commons BY 3.0 Unported license
© Patrick Wischnewski

The steadily increasing number of product variants is leading to a steady increase in costs and time expenditures in development, production and sales. Even more, designing variants with respect to the actual market situation in order to exactly meet the customer’s demand requires optimization methods that enable the manufactures to strategically optimize their product portfolio.

Because of the discrete structure of the products, the respective optimization procedures are expensive from a computational point of view. Therefore, in order to successfully develop procedures that efficiently perform strategic optimizations for industrial size problems, two

directions of research are necessary. The first direction is towards efficient optimization procedures. The second direction is towards efficient encodings of the problem of exactly matching the properties of the optimization procedure.

In several industrial projects we have successfully used SAT-based optimization procedures. In these projects we observed that finding the right encoding for a product portfolio and adjusting the optimization procedure respectively was the key for efficiently performing strategic optimizations in terms of the product portfolio.

4 Summary of Closing Discussion

The closing discussion of the seminar focused on identifying future research directions in product and software configuration.

During the discussion, the idea of easy-to-use but expressive configuration languages emerged as the main vision for the future. These two language properties were identified as at odds with each other, making achieving this vision challenging. The participants pondered whether such languages would be generic or domain-specific and questioned how much representational adequacy generic languages could achieve.

In addition to support for ease of modeling, such languages should also provide efficient reasoning capabilities without burdening the user. The discussion explored the idea of targeting a range of solvers using intelligent solver-selection logic and translators determining the most efficient problem encodings. The languages should also support a smooth transition between different computational classes, without the need to reformulate existing configuration models.

A challenge posed by the expressive-language vision is the ability to quickly and reliably classify a given problem based on its characteristics and identify the most appropriate solvers and encodings. The participants agreed about the need to create a body of knowledge classifying configuration problems and the most appropriate solving techniques for each class.

The participants also recognized usable configurators as an additional important research direction. Existing works on this topic are very sparse; however, tool builders and problem modelers require clear guidance on how to design effective interactions with users. This direction requires multi-disciplinary efforts, including human-machine interface, cognitive science, and experimental research.

Finally, the participants postulated that achieving progress in the field would require creating widely accessible collections of benchmark problems. They also identified the diversity of configuration languages and tasks as main challenges in creating such benchmarks. Standard formats such as DIMACS in the SAT community greatly simplify creating benchmarks. However, other communities offer some positive experiences in addressing these challenges. For example, the CSP community has created the MiniZinc language, which is used as a frontend for wide range of solvers.

Participants

- Michel Aldanondo
University of Toulouse, FR
- Danilo Beuche
pure-systems GmbH –
Magdeburg, DE
- Nikolaj Bjorner
Microsoft Res. – Redmond, US
- Krzysztof Czarnecki
University of Waterloo, CA
- Conrad Drescher
SAP AG – Walldorf, DE
- Andreas Falkner
Siemens AG – Wien, AT
- Jianmei Guo
University of Waterloo, CA
- Albert Haag
SAP AG – Walldorf, DE
- Holger H. Hoos
University of British Columbia –
Vancouver, CA
- Lothar Hotz
HITeC e. V. /
Universität Hamburg, DE
- Arnaud Hubaux
ASML – Veldhoven, NL
- Dietmar Jannach
TU Dortmund, DE
- Christian Kästner
Carnegie Mellon University, US
- Eunsuk Kang
MIT – Cambridge, US
- Charles Krueger
BigLever – Austin, US
- Wolfgang Küchlin
Universität Tübingen, DE
- Daniel Le Berre
Artois University – Lens, FR
- Tomi Männistö
University of Helsinki, FI
- Leonardo G. P. Murta
Federal University Fluminense –
Niteroi, BR
- Klaus Schmid
Universität Hildesheim, DE
- Norbert Siegmund
Universität Passau, DE
- Markus Stumptner
University of South Australia
Adelaide, AU
- Juha Tiihonen
Aalto University, FI
- Andrzej Wasowski
IT Univ. of Copenhagen, DK
- Patrick Wischniewski
Logic4Business –
Saarbrücken, DE
- Ed Zulkoski
University of Waterloo, CA

