Variability Support for Variability-Rich Software Ecosystems

Klaus Schmid
Software Systems Engineering, Institute of Computer Science,
University of Hildesheim, Germany
schmid@sse.uni-hildesheim.de

Please cite this publication as follows:

The corresponding BibTeX-entry is:
@INPROCEEDINGS{Schmid13c,
author = {Klaus Schmid},
title = {Variability Support for Variability-Rich Software Ecosystems},
booktitle = {Fourth Workshop on Product LinE Approaches in Software Engineering (PLEASE) at the International Conference on Software Engineering, 2013},
year = {2013},
pages = {4},
publisher = {IEEE}
}

Accepted for publication by IEEE. ©2013 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.
Variability Support for Variability-Rich Software Ecosystems

Klaus Schmid
Software Systems Engineering, Institute of Computer Science
University of Hildesheim
Email: schmid@sse.uni-hildesheim.de

Abstract—Lately, software ecosystems have generated a lot of attention as they are very important to modern software industry. Over the course of several research projects, we addressed the problem of variability-rich software ecosystems and their relation to software product lines in our research group. This paper summarizes some of the problems we identified and describes some solutions we created both on a conceptual level and implemented in a prototype tool environment.

Index Terms—Software Ecosystems, variability-Rich Ecosystems, Product Line Composition, Product Line Refinement, Default Logic, Supply Chains

I. INTRODUCTION

Traditionally, a software product line (SPL) is well-defined and bounded in terms of the products that will be derived from it [1]. In traditional software product line engineering a product line is also only the basis for software development in a single organization. Thus, product line engineering improvement approaches like the Families Evaluation Framework (FEF) typically focus only a single organization [2]. More recently, the notion of software ecosystems received considerable attention as it is of fundamental importance to modern software development practice [3], [4]. As opposed to the traditional notion of software product lines, which focuses on a single organization, software ecosystems aim at the integration of multiple different organizations.

As Bosch points out in [5], software ecosystems are somewhat related to software product lines due to the implication of software reuse. However, it is interesting to note that one of the key concept of product line engineering (variability management) is usually not addressed in ecosystems research [4]. Here, we go a step further and introduce the notion of a variability-rich ecosystem: an ecosystem, where variability in assets produced by one organisation strongly impacts assets produced in a different organization, i.e., organizations must be aware of the variability introduced in a different organization (and in the way this is resolved, if so).

A related term, that will be very useful in our discussion, is the platform. While the ecosystem concept focuses on the economic interactions among stakeholder organizations [3], the term platform denotes the common technical basis on which participants in the ecosystem rely. Depending on the economic structure of the ecosystem one or more of the stakeholder organizations may own this platform (e.g., Windows OS as a platform in the Windows ecosystem).

We discuss two types of software ecosystems in the next section and in Section III we illustrate the concepts with some examples. In Section IV we describe some problems we could identify in the context of variability-rich software ecosystems, while in Section V, we will describe our approach to dealing with these problems. Finally, in Section VI, we will conclude.

II. TYPES OF ECOSYSTEMS

We can differentiate between variability-rich ecosystems and non-variability-rich ecosystems. As we will emphasize the relation to software product lines, we will restrict our further discussion to the variability-rich case, although some parts of the discussion may be of general relevance. If we look closer at some real-world ecosystems, we can differentiate two types of (software) ecosystems based on platform ownership and control:

• Vertical ecosystems are based on the platform of a single (or small group of) partner(s). This "build-upon" relationship may be recursive in the sense that what one partner builds is used as a basis by another one.

• Horizontal ecosystems are created by combining assets from several organizations without an explicit builds-on-relation among the systems. Only the combination of the individual assets creates a relevant platform.

The Internet is an example of such an ecosystem. The Internet platform is created and run by the interplay of numerous stakeholders who all contribute to the underlying platform, without any central control.

No matter whether the platform creation in the ecosystem is predominantly done horizontally or vertically, further stakeholders will want to build upon it. Often, if we look at ecosystems, we even see combinations of horizontal and vertical ecosystems that may even be overlapping and more or less tightly coupled.

If we consider that we focus in this paper solely on variability-rich software ecosystems, the important question is Will software that build on the platform work with all variants of the platform? In particular, it may happen that some software requires a specific variant as a basic platform (e.g., software that requires the windows server edition to run).
III. Examples

In the past, we discussed several examples of ecosystems in different publications, with which we got significant practical experience [6], [7]. Here, we will briefly outline two of those, because they contributed most to the concepts underlying our research on ecosystems and the solutions we derived.

A. Single Platform Ecosystem

In the past, we described an ecosystem for University Management Systems and discussed in particularly the problems of variability management and of variability evolution [6]. According to our definitions above, it is a prototypical example of a variability-rich vertical software ecosystem. It is variability-rich as the company provides University Information systems and adapts them to the specific demands of a wide range of different customers. The variation is due to different categories of universities, different regional laws, etc. At the same time it is a vertical ecosystem as further organizations provide specialized add-ons that can integrate with this system. The universities also provide special extensions for their own use. Thus, there are multiple stakeholder organizations who need to interact to achieve the desired results. Stakeholders who build on top must take the configuration of the underlying platform into account and may even introduce further variability. Thus, it is a variability-intensive ecosystem.

B. Multi Platform Ecosystem

A slightly more complex case of an ecosystem has been described in [7] and is at the core of the EU-project INDENICA. In this case no single company controls the ecosystem or provides the basis for it. Rather the platform is only created by the composition of several sub-platforms. This makes it similar to a horizontal ecosystem, however, at the same time further partners in the ecosystem may build on top of the ecosystem. In the INDENICA-case study the basic platform is composed of three platforms that are respectively relevant for warehouse management, yard management and telecommunications. Only all three together provide the desired range of logistics capabilities, thus the basic platform is created as a horizontal ecosystem, while further partners may build on the integrated platform, yielding a vertical ecosystem. Again the different ecosystem platforms may be instantiated to create special instances, e.g., for a specific installation.

In summary, the examples given above (and also other industrial cooperations) show that composition plays an important role in software ecosystems. We addressed this also earlier, when we analyzed the principles underlying existing software ecosystems [8]. There we also identified some practices for variability management in practical ecosystems.

Besides composition, it is important that some stakeholders in an ecosystem will only be able to build on a subset of the variants of a variability-rich platform. In order to describe this, we introduce what we call a specialized platform. It can often be described as a form of (partial) instantiation of the platform. We call this specialization akin to specialization in [9], although we lift the concept to ecosystems.

The Easy-Producer toolset has been developed and evaluated especially in the context of the second example. However, we currently evaluate it also with further industrial partners.

IV. How to Support Variability-Rich Ecosystems

As discussed in the previous section, key operations on ecosystems that need to be supported are composition and refinement. These are not necessarily the only operations, but in our work we found we identified the importance to deal with them time and time again. As we will discuss below, while these concepts are also well-known in product line engineering, the ecosystem context changes the problem and leads to the need to address them in a different way.

Work on product lines that addresses composition is often called multi software product lines. This can either be with a homogeneous mechanism for variability management [10] or can even be extended to a heterogeneous approach [11], where product lines using different variability modeling mechanisms are combined. In both cases, however, so far the assumption is that the composition operates on monolithic product lines, while we extend this to the composition of ecosystems. Existing approaches provide rather complex mechanisms to automate the composition as much as possible and often rely on heuristics like "if two names are identical this relates to the same underlying variability". Such assumptions are rather problematic in the context of ecosystems, in particular, if evolution occurs, a change in one participating product line may easily break such an assumption.

For ecosystems it is typical that each participating product line has its own variability model [8]. Regarding granularity, we found in [8] that the implementation units with independent variability models are often very small, e.g., individual features. An important distinction of our approach to typical multi-product line methods is that each of the elements that can be composed, can in turn be a complete ecosystem. Thus, arbitrary combinations of specialization and composition are supported to map the complexity of real-world situations.

The composition of such independent variable units may ideally be used horizontally to create a (horizontal) platform as well as vertically to build additional components on top of it, while being able to handle variability throughout. As a consequence, we see the need to support flexible composition of variability-rich ecosystem components, where the individual components may be both numerous and of arbitrary size as an important challenge to software ecosystem research.

A separate problem is the specialization problem. Specializing a product line in the sense we use the term means to partially instantiate it. Thus, some (but not all) of the variability is bound, giving rise to a (partially) instantiated variability model, along with associated (correspondingly instantiated) artifacts. The way we use the term specialization here is similar to the way it is used in [9], the major difference is that we generalize it from a single product line to a whole ecosystem. In particular, this specialization can work on arbitrarily composed and potentially already (partially) specialized parts of the ecosystem, which makes this operation very powerful. It
also requires that homogeneous implementation mechanisms must be appropriately managed. Again this is not absolutely new as product line engineering research already addressed the need to handle compositions that contain multiple instantiation mechanisms, but here it must be managed throughout the whole ecosystem. This we illustrate below.

If refinement and composition are available, we can create rather complex software ecosystems as shown in Figure 1. The capability to model and implement such complex combinations of product lines is a cornerstone to support variability-rich ecosystems. In this example ”Med Prod A” may contain open variability from all predecessors, but now it needs to be resolved. We configure the appropriate values, but as each of those also contains different artifacts and may use different variability mechanisms, the application of the diverse variability mechanisms must be adequately orchestrated.

There are, however, further capabilities that are relevant for software ecosystems. For example, the support of evolution is an important problem. While evolution is already difficult in classical product line engineering, it is even more difficult in ecosystems as individual product lines will evolve autonomously and the composition should not introduce problems as a result. One major design choice that is relevant to the distributed variability-evolution in variability-rich ecosystems is to separate the variability model into a number of different partial models that can be integrated on the fly to yield the finally relevant variability model. This is also done some successful ecosystems as we discussed in [8]. We also introduce explicit variability interfaces to minimize the impact of evolution and introduce explicit mechanisms to handle versions, e.g., it is possible to determine that only versions that fulfill certain conditions can be used.

Further, often it is necessary to ensure that even a partial combination yields an executable result. This means, that we need to provide default values that can be used in the absence of more detailed variability information to create a runnable implementation. From an evolution perspective this entails that the management of defaults and their overriding needs to be adequately supported as part of the variability modeling approach [12]. In [13] we provided a calculus to support the handling of defaults in variability modeling which also includes default constraints. We could show that this provides the most adequate results for configuration for a wide range of ecosystem situations and identifies problems otherwise.

V. A Solution Approach

In the preceding section, we discussed the main operations that we must support to address ecosystems: composition and refinement. Together they allow to create all combinations in complex variability-rich ecosystems as discussed above. From the perspective of evolution the support for defaults seems very important as we discussed earlier in [13]. This is supported by the analysis of Savolainen et al. [12]. In this section, we will now discuss the current state of the EASY-Producer tool, which aims at providing these capabilities and some consequences the corresponding design decisions had. An earlier version of the toolset was already described in [14].

The EASY-Producer toolset is build on the notion of decision modeling, but this does not introduce any fundamental difference as we showed the equivalence of decision modeling and feature modeling in [15]. In its current instantiation the toolset aims at simplifying the above operations as much as possible without introducing any, perhaps problematic, assumptions. Composition is realized in a rather simple way: at any point the user may create a new project which can include (import) already existing projects. This operation leads to making all variability of the different projects available in the new (composed) project. In order to ensure that no name-clashes exist, the imported projects receive their own namespace. Regarding the associated assets, it is possible to explicitly decide whether associated projects may be copied or only referenced. There are reasons for both. A copy-operation is necessary, if variation is effectively resolved in the instantiation, meaning a refined artifact is created. However, in some cases such a refinement does not actually happen on a code-level. In this case, only a reference needs to be made, in particular, if more than one reference exists, only a single copy is required. The tools allow to explicitly make this choice. By now, we found practical situations, where actually either one can be necessary.

However, the tools at this point do not try to automatically link the different product lines. Heuristics like ”similar names probably refer to the same variability” are not used. Rather, the developer must link the variabilities explicitly in terms of constraints. In this way arbitrary relations among the variabilities in the different product lines can be expressed. We see such heuristics, however, as a way of supporting the domain engineer in constructing the composition, thus we might integrate them in the future into the user interface.

Specialization is supported as well by the tools. It is realized by a straight-forward partial instantiation of variability in the sense that some variability is selected and the resulting instantiation of the implementation artifacts are performed. This operation takes a product line infrastructure and generates a new (instantiated) one. The tool supports arbitrary many instantiation methods, which are typically artifact-type-specific. The only restriction that is made for any variability instantiation method is that it needs to be possible to also create partial instantiations, i.e., even if the variability is not fully specified an instantiation is possible, defining all
implementation as far as values are given, while leaving everything else untouched. This enables to create instantiation over several levels (cf. Graphics in Figure 1, which stretches over three levels). Allowing multi-level, successive refinement automatically leads to a different issue: product and reusable infrastructure can no longer be clearly differentiated. Look for example at medical imaging in the picture: is it a product (relative to 3D-Streaming yes, relative to Med Prod A no)? Is it a reuse infrastructure (yes relative to Med Prod A, no relative to 3D-Streaming)? Thus, we decided to completely abandon the differentiation between product line (infrastructure) and product: they are the same, only with varying degree of variability. A product then simply happens to be a project without (development time) variability. In order to make this variability. A product then simply happens to be a project

product: they are the same, only with varying degree of variability. A product then simply happens to be a project without (development time) variability. In order to make this variability. A product then simply happens to be a project

Finally, from an evolution perspective the toolset also supports localized variability models (each project has its own variability model, which describes variability relevant to any assets). Also defaults are supported as first-class citizens in the variability management approach, in order to simplify the evolution as discussed in [13].

VI. CONCLUSIONS

In the research, described above, we focused on supporting variability-rich software ecosystems. To the best of our knowledge, we are the first to introduce this concept explicitly into software ecosystem research, although it might have been implicitly present, earlier. We presented some key challenges that are important in this context, and briefly discussed main characteristics of a toolset that addresses these problems by providing easy to use support. The toolset is in an advanced state as it is currently used in several research collaborations and also used for industrial transfer projects. While, based on existing, ongoing evaluations the approach seems rather satisfactory for the situations for which it was initially intended, there are also some shortcomings. However, these are due to the fact that the research described here, excluded these contexts right from the beginning.

We see the following shortcomings: the tool only provides support for cases where the underlying implementation and variability can be directly accessed. This seems to a widespread situation also in ecosystems discussed today (cf. [3]). However, we can envision cases, where it is not possible to make the assets and/or the variability model available. In such a case we would need a form of distributed configuration as described in [16], however, this has not yet been implemented.

Another shortcoming at this point is the strong focus on development time. A consequence of this is that we typically aim at instantiating the assets in the development environment. However, within the INDENICA-project we also performed of applying the approach and tools during runtime. We assume that we will eventually fully extend the approach towards dynamic software product lines (DSPL) [17].

ACKNOWLEDGEMENTS

We would like to thank the numerous people who were involved in this research so far, in particular Hendrik Brummermann, Holger Eichelberger, Sascha El-Sharkawy, Markus Keunecke, Christian Kröher, and Phani Saripelli (alphabetic order). This work was partially supported by the INDENICA-project, funded by the European Commission grant 257483, area Internet of Services, Software & Virtualisation (ICT-2009.1.2) in the 7th framework programme.

REFERENCES


\footnote{As this is an Eclipse-based tool, it is also mapped to Eclipse-projects.}