Variability Management and Compositional SPL Development

Jilles van Gurp

Nokia Research Center, Helsinki, Finland
jilles.vangurp AT nokia.com

This position paper reflects on the implications for variability management related practices in SPL development when adopting a compositional style of development. We observe that large scale software development is increasingly conducted in a decentralized fashion and on a global scale with little or no central coordination. However, much of the current SPL and variability practices seem to have strong focus on centrally maintained artifacts such as feature and architecture models. We conclude that in principle it should be possible to decentralize these practices and identify a number of related research challenges that we intend to follow up on in future research.

Introduction

Over the past ten years software product line (SPL) conferences and related workshops have established SPL research as a new discipline in the broader field of software engineering. We, and others, have been contributors to this field with publications on software variability management [1, 2] and involvement in earlier workshops [3]. In more recent work, we have published about compositional development of software products [4]. Compositional development decentralizes much of the traditionally centralized activities in an integration oriented SPL, including requirements management, architecture and design. These activities are pushed down to the component level.

So far, Moore's law has accurately predicted the exponential growth of transistor density on chips and software developers seem to have matched this growth with a similar growth in software size (generally measured in lines of code). Decentralization of development activities allows development to scale better to such levels. This scalability is required to create more products that integrate a wider diversity of components and functionality. Making software products that support an ever wide range of functionality is necessary in order to differentiate in the market.

Unfortunately, decentralization has far reaching consequences for SPL methodology and tooling. A common characteristic of many of the currently popular SPL methodologies is the use of centrally maintained feature models that describe the variability in the SPL. For example, much of the SPL specific tooling depends on such models. This includes build configuration tools; requirements management tooling and product derivation support tools. Additionally, processes and organizations are generally organized around these tools and models.

In a compositional approach where development is decentralized and different components are developed by different people, business units and organizations using different methodologies and tools, these approaches break down. Products are not created by deriving from a central architecture and feature model but by combining different components and writing glue code.

The intention of this position article is to reflect on this topic and identify new research issues. It seems that much of the current research is conducted under a closed world assumption where one central organizational entity is in charge of overall design, management and governance of the SPL software. We believe this assumption to be flawed. The reality on the ground is quite different. Increasingly software companies are collaborating either directly or through open source projects on software assets that they have a shared interest in. It seems that it is almost impossible to develop software these days without at least some dependencies on external software. Additionally, in large companies software development is distributed throughout the organization. Consuming software from a business unit on a different continent poses very similar challenges to partnering with a different company.

In the remainder of this article we first briefly introduce the topic of compositional development before reflecting on what that means for variability management and finally reflecting on some future research topics related to that.

Compositional development

Compositional development might be interpreted as a move back to the COTS approaches popular in the past decade. In those days, it was suggested that companies would either buy components developed by other components or use in house developed components from a reusable component base. These approaches fragmented in roughly four directions over the past decade:

- **Integrated Platforms**. In this approach, one vendor offers a fully integrated software platform complete with tools, documentation, vast amounts of reusable software and consulting. Examples of companies that provide such vertical stacks of software are IBM, Oracle and Microsoft. While successful, this approach is mostly limited to the domain of enterprise systems. Characteristic of this domain is that most systems built are one of a kind.
- **SPLs**. For other domains than enterprise systems (e.g. embedded software), SPLs have emerged as a successful way to develop a platform in house and use that to build software products. Unlike enterprise software, most embedded software products are not one of a kind. Product lines for embedded software also tend to be highly specialized for the domain (e.g. mobile phones; audio visual equipment; medical equipment). Within those domains, the product line aims to support a wide range of products.
- **'True' COTS**. The vision in the nineties was not SPLs or huge vertical stacks of technology but a market of component vendors whose products

could be combined by product developers. Except for a few limited domains (e.g. GUI components), this market never emerged [5]. However, this market is comparatively small. Problems with respect to ownership of source code; interoperability; documentation and support are usually cited as the causes for this.

- **Open source**. From the mid nineties a vast amount of software has been released in the form of open source. Currently there are tens of thousands of active projects releasing high quality software. Most commercial software development, including embedded software development, now depends on substantial amounts of this software. Very few commercial software companies are 100% open source though. It seems many companies have a small layer of differentiating software & services that are not open source.

Of course, these approaches overlap. For example, several SPL case studies have been published for enterprise systems. Additionally using COTS in combination with either SPLs or enterprise platforms is quite common. In fact, most of the COTS companies seem to make components that are specialized for a particular platform. Finally, open source is important for COTS, integrated platforms and SPLs. Compositional development of products involves combining elements from all of these approaches and is certainly not about just COTS.

In compositional development, development teams of components or subsystems operate with a higher degree of autonomy then they would in a SPL organization. Identification of key requirements and design solutions is largely the responsibility of these teams. They interact with developers of other components they depend on and with developers of components (or products) that depend on them. However, the central coordination of this communication is absent.

The rationale here is to bring the decision processes as close to where it has an impact; and also to where the domain and technical experts operate. This is a quite different working model from the traditional one where a group of seniors decides together with the major stakeholders on design, requirements and other issues.

The problem with integrated SPL model is that it does not scale to the current industrial practice where software systems spanning multiple millions of lines of code are now the rule, not the exception. Managing the design and architecture of such software centrally is extremely difficult. The amount of people with a detailed enough knowledge of the software is very low in such companies. Additionally these people tend to be very busy and are generally very hard to replace.

In practice, this means that as software size grows, decision makers at the top are increasingly detached from the design details of the software. In other words, it disqualifies these individuals for making the technical decisions they should be making. The logical, and in our view inevitable, approach is to stop trying to take most of these decisions centrally.

A useful analogy here might be that of the communist era planned economies vs. the capitalist free market system. Making government level decisions about when, where, and how to move a few tons of tomatoes is obviously nonsensical to proponents of the latter. Yet this is exactly what happened in the strictly hierarchical organized planned economies leading to obvious issues such as one half of the country having a shortage of tomatoes and another half having tons of tomatoes rot in some central deposit. Similarly, detailed decisions about design and features are best left to the experts working on the actual software and any depending stakeholders.

Variability Management

Software variability is the ability of software assets to be extended, customized, configured or otherwise adapted. In SPLs, the intention is to have a set of reusable assets and architecture as well as a means to create software products from those. In other words, the reusable assets and architecture feature a degree of variability that is put to use during product creation.

In line with research from others at the time, our earlier work on variability management identified feature models as a means to identify so-called variant features in the requirements; and also as a means to plan the use of variability realization techniques to translate the variant features into variation points (i.e. concrete points in a software system where there is an opportunity to bind variants to the variation points during product creation).

More recent research, has focused on (partially) automating and supporting this process; formalizing the underlying models (e.g. using the UML meta model); tool support; etc. Some of these approaches are now used successfully in industry. A small commercial tool and support community is emerging. E.g., Big Lever (www.biglever.com) can support companies with support and tooling when adopting a SPL approach. MetaCase (www.metacase.com) provides similar services. Additionally, various research tools integrate feature modeling support into popular IDEs. Clearly, these tools are useful and various case studies seem to confirm that.

However, all of them more or less depend on the presence of a centrally governed architecture and feature model. Introducing compositional development implies less central control on these two assets. Consequently, requirements analysis and architecture design activities are also affected.

Assuming that software development is fully decentralized, this means the following:

- New features or variant features are identified, prioritized and implemented locally rather than at a central level.
- Important architecture decisions with respect to component variability and flexibility are mostly taken without consulting a central board of architects.
- New variant features are not represented in centrally maintained feature models unless the updating of such models is either automated or enforced with (central) processes and bureaucracy. This may be hard or even impossible given differences between organizations & processes of the various software development teams involved.
- For the same reason, any tool mapping of such new variant features to software variation points is not updated. Such mappings are critical in e.g. build configuration tools.

Provided vs. required variability

Feature models may be regarded as descriptions of either required or provided features in a software system. Feature models of required features are the output of the software analysis process. They may be interpreted as specifications of the software or be used to guide the design process. On the other hand, feature models of provided features describe implemented software systems in terms of the features actually implemented in the software. Models of provided features may be of use for e.g. configuring software products derived from the platform. In theory, these models should be the same but in practice requirements constantly change and few software products actually conform to initial requirements specifications. In fact, most development on large software systems is software maintenance and concerns changes to both the software and its provided feature specifications.

A similar distinction can be made for architecture documents. While the words 'architecture document' suggest that software is developed according to the blueprints outlined in this document, a more popular use of architecture documentation tools seems to be to document the design of already implemented software. This type of documentation is generally used to, for example, communicate the design to various stakeholders. Additionally, models described in an architecture description language may be used to do automated architecture validations; simulations; or system configuration.

When using a build configuration tool based on feature models, developers select existing implementations of features or variant features. In other words, they make use of a model of the provided variability. The tool in turn needs to map the feature configuration to variation points in the implementation artifacts. In other words, it has an internal model of the provided variation points in the software architecture.

This distinction of provided vs. required variability is highly relevant because we observe that much of the SPL tooling is more related to provided rather than required variability. De-centrally developed components may not conform to a centrally pre-defined model of required features but they certainly do provide features that may be described. Similarly, these components do not realize a pre-defined central architecture but may still provide explicit variation points. There is nothing inherently central about either feature models or architecture.

Research Issues & Concluding remarks

This article observes that there is a trend to decentralize software development in the current software industry and that this raises issues with respect to SPL development, particularly where it concerns the use of centrally defined feature models, architecture models, and related tooling. Fundamentally, this centralized/top down style of software development is not compatible with the bottom up style development seen across the industry and we foresee that this centralized approach will not continue to scale to the required levels. Already, the incorporation of de-centralized elements is evident in the increasing popularity & use of open source components, and also in publications such as Van Ommering.

From this observation, we explored a bit what it means to do decentralized compositional development and what that means for the centralized use of feature models and architecture models in current SPL development. An important conclusion we make is that most of the current tooling is focused around using feature models of provided features in a software system to configure provided variability points in a software architecture. We do not see any fundamental objections to continue doing that in a decentralized development model. Feature models of individual components may be provided and similarly the variability provided in these components may be described.

The above suggest that much of the tooling that currently exists for variability management may be adapted for use in a de-central fashion. Some potential research topics related to this that we intend to explore further in future work are:

- How to synthesize aggregated feature models and architecture models from the individual component level models given a component configuration.
- How to validate component configurations given incomplete feature & architecture information from components.
- How to deal with integrating components without formally documented features and variation (e.g. most open source software comes without such documentation).
- How to deal with crosscutting variant features that affect multiple, independently developed components. E.g., security related features generally have such crosscutting properties.

Some preliminary work related to this has already been done by amongst other Van Ommering [6] who wrote articles on KOALA and development issues related to compositional development. The trend, judging from KOALA, similar approaches and, also from the increased popularity of component frameworks such as standardized in OSGI, seems to be to address these issues with microkernel like architectures that explicitly requires components to state dependencies and interfaces.

References

[1] J. van Gurp, J. Bosch, M. Svahnberg, On the Notion of Variability in Software Product Lines, Proceeedings of WICSA 2001, 2001.

[2] P. Clements, L. Northrop, “Software Product Lines - Practices and Patterns”, Addison-Wesley, 2002.

[3] J. van Gurp, J. Bosch, Proceedings of First workshop on software variability management (SVM 2003), Groningen 2003.

[4] J. van Gurp, C. Prehofer, J. Bosch, Scaling Product Lines to new Levels: the
Open Compositional approach, submitted December 2006.

[5] B. Lang, Overcoming the Challenges of COTS, news @ SEI, 4(2) http://www.sei.cmu.edu/news-at-sei/features/2001/2q01/feature-5-2q01.htm, 2001.

[6] R. van Ommering, Building product populations with software components, proceedings of Proceedings of the 24rd International Conference on Software Engineering (ICSE 2002), pp. 255-265, 2002.