Scalability of Variability Management: An Example of Industrial Practice and Some Improvements

Yinxing Xue, Stan Jarzabek
School of Computing, National University of Singapore, Singapore
{yinxing,stan}@comp.nus.edu.sg

Abstract—Having set up reusable core assets for a Software Product Line (SPL), it is a common practice to apply Variation Techniques (VTs) to manage variant features. As each VT can handle only certain types of variability, multiple VTs are often employed, such as conditional compilation, configuration parameters or build tools. Our earlier study of an SPL at Fudan Wingsoft Ltd revealed potential scalability problems of multiple VTs. As a remedy to the above problems, in the follow-up study we replaced multiple VTs originally used in the Fudan Wingsoft product line, with a single, uniform VT of XML-based Variant Configuration Language (XVCL). This paper provides a proof-of-concept that commonly used variation techniques can indeed be superseded by a subset of XVCL, in a simple and natural way. We describe the essence of the XVCL solution, and evaluate the benefits and trade-offs involved in multiple VTs solution and single VT - XVCL solution.

Keywords- Generative technique; Software Product Line; Variability management

I. INTRODUCTION

In previous paper [16], we analyzed a Software Product Line (SPL) called Wingsoft Financial Management Systems (WFMS-PL), developed by Fudan Wingsoft Ltd. WFMSes provide web-based financial services for employees and students at universities in China. Following a common practice, Wingsoft set up product architecture, identified core assets for reuse, and then applied a range of common design-time Variation Techniques (VTs), such as conditional compilation, design pattern or configuration parameters, to manage product-specific features in core assets.

Features vary in the granularity and in the scope of their impact on core assets [5][12]: Fine-grained features affect many core assets of an SPL, at many variation points [8]. Code of such features becomes scattered across core assets. A Coarse-grained feature, on the other hand, can be contained in a component (e.g., a class or package) that is included into a custom product when a given feature is needed. Mixed-grained features involve both fine- and coarse-grained impact.

Coarse-grained features are easier to manage than fine-grained features. Feature granularity depends to some extent on the design of core assets. Good architectural design can change feature granularity in our favor, increasing the number of coarse-grained features, and reducing the number of variation points in core assets for the features that remain fine-grained.

Variation Techniques (VT) must match feature granularity. Therefore, it is common to use multiple VTs, for example, conditional compilation to handle fine-grained features or a build tool such as Ant to handle coarse-grained features. Such VTs are easy to apply, and most of developers are familiar with them. However, as our study revealed [16], applying multiple VTs does not scale well, especially in cases of mixed-grained features. While reuse and modification of mixed-grained features is inherently difficult, applying multiple, often poorly compatible VTs aggravates the problems.

As a remedy to the above problems, in the follow-up study we replaced VTs originally used in the Fudan Wingsoft product line, with a single, uniform VT of XVCL (XML-based Variant Configuration Language) [15]. XVCL applies generative mechanisms to organize software into highly parameterized meta-components. These meta-components form SPL core assets that are adaptively reused in product derivation, automated by the XVCL Processor [9]. In this paper, we propose to use XVCL as a uniform VT to replace the original ones described in [16]. We also present an initial evaluation of benefits and trade-offs involved in adopting a uniform VT.

A practical lesson learned from our study is that in small- to medium-size product lines, applying multiple VTs may be a viable solution, as it requires less training, and variability can still be effectively managed in that way. As the product line grows in size and the impact of features on core assets becomes more complex, a company may experience problems. Then moving towards a uniform variation technique approach may be beneficial. However, this will require a more systematic approach to reuse, and training of SPL personnel.

The paper is organized as follows: Section II summarizes the findings from our earlier study of multiple VTs in WFMS-PL [16]. Section III describes the XVCL solution to WFMS-PL, and explains how it alleviates problems of the multiple VTs. We evaluate the XVCL approach in Section IV. Related work and concluding remarks end the paper.

II. PROBLEM OF MULTIPLE VARIATION TECHNIQUES

WFMS for Fudan University developed in 2003 evolved into a Software Product Line WFMS-PL [16] with more than 100 customers including major universities in China such as Shanghai Jiaotong University (WFMS for this university can be found at http://www.jdcw.sjtu.edu.cn/wingsoft/index.jsp), Zhejiang University, Chongqing University and others.

Main functionalities of WFMS include the Financial Management Subsystem (FMS) that manages all the university income and expenses, the Salary Management Subsystem (SMS) that manages salary of employees, the Reward Management Subsystem (RMS) that manages rewards for...
employees and students, and the Tuition Management Subsystem (TMS) that manages student tuition fees. The TMS is a web-based portal for students to pay their tuition fee online, with functions such as login, service customization, on-line payment and history query. In addition, the TMS also provides accounting services (e.g., report generation and bill settlements) that interface universities with banking systems.

First five WFMS product variants were developed by ad-hoc copy-and-paste reuse, and each WFMS was maintained as a separate product. As the number of customers was growing, ad-hoc reuse and maintenance was becoming more and more taxing on company resources. To address this problem, Wingsoft set up SPL core assets as follows: First, Wingsoft designed architecture to be shared by future products and adopted commonly available VTs, such as conditional compilation and configuration parameter files. Wingsoft did not use any advanced VTs due to the practical need for an easy-to-implement, cost-effective SPL. Such an SPL was built and reused with minimum training of the staff in SPL techniques.

We described implications of using multiple variation techniques in WFMS-PL in [16], so here we only summarize the main results. We analyzed the Tuition Management Subsystem (TMS), as it involved types of variability and variation techniques that are representative for the whole WFMS. The code of TMS is 25% of the whole WFMS system, comprising 58 Java source files, 99 JSP web pages, and several configuration files. In TMS feature model, there are 32 variant features and 9 mandatory features.

Wingsoft adopted simple, freely available VTs, selecting the right VT for features at hand [16]. Fine-grained features were managed by conditional compilation and commenting code in core assets. Coarse-grained features were managed by build tool Ant. Mixed-grained features were managed by configuration parameters. For the 32 variant features in TMS, conditional compilation or commenting technique was involved in 31 variant features. Ant was used by 19 variant features, overloading fields - by 13 features, configuration file - by 12 features, and design patterns - by 3 features.

Our study [16] revealed potential problems of multiple VTs in an SPL: One feature was often handled by several VTs which had to be properly synchronized. 26 among 32 variant features were managed by more than one VT, 13 features - by 3 VTs, and 3 features - by 4 VTs. The difficulty roots in the management of variant features involved in several VTs at the same time. Examples below illustrate what was involved in managing such features, and hint at complications that are bound to arise as the size of the system and the number of inter-dependent variant features grow.

Feature PayByItem (Fig. 1, Fig. 2 and Fig. 3) is a mixed-grained feature managed by configuration parameters that parameterize reflection, strategy design patterns [7], and also the build tool Ant. In Fig. 2, method user.getPayMode() at line 6 returns the value of the parameter paymode at line 2 in the configuration file (Fig. 1). According to that value, strategy pattern generates the specific code skeleton for feature PayByItem.

---

1 Details on TMS’s feature particulars and feature dependency are available at this link: [www.comp.nus.edu.sg/~yinxing/TMS-information.html](http://www.comp.nus.edu.sg/~yinxing/TMS-information.html)

---

```xml
<project name="webFee" basedir="." default="main">
  <target name="copy-src" depends="create-folders">
    <copy todir="${web-root.dir}">
      <!-- Copy webpages of Feature PayByItem -->
      <fileset dir="${core-src.dir}/${PayByItem}"/>
    </copy>
  </target>
  <target name="copy-webpage" depends="create-folders">
    <!-- Copy webpages of Feature PayByItem -->
    <copy todir="${web-root.dir}" filesystem="html" excludes="docs/*">
      <fileset dir="${web-page.dir}"/>
    </copy>
  </target>
  <target name="copy-jsp" depends="create-folders">
    <!-- Copy JSP files of Feature PayByItem -->
    <copy todir="${web-root.dir}">
      <!-- Copy JSP file selFeeItem.jsp, which was included by the command <fileset dir> at line 12. -->
      <fileset dir="${web-page.dir}"/>
    </copy>
  </target>
</project>
```

---

**Fig. 1.** Managing PayByItem with configurations parameters

**Fig. 2.** Managing PayByItem with reflection and strategy patterns

**Fig. 3.** Managing PayByItem with Ant

Coarse-grained impact of feature PayByItem is managed by build tool Ant (Fig. 3). Using parameters, Ant could include/exclude source files that were relevant/irrelevant to a given feature. An Ant script sets the target path for the source code to be included in the final build. Feature PayByItem has a related JSP file selFeeItem.jsp, which was included by the command `<fileset dir>` at line 12.

Mixed-grained impact of features is the main source of problems for scalability of the multiple VTs to manage SPL variability. Scattered impact of mixed-grained features brings forth the difficulties to keep multiple VTs in synchronization one with another. Inter-related configuration parameters control both Ant and Java conditional compilation. There are many examples of such interactions between multiple VTs in the original TMS core assets. Its maintenance entails the accurate understanding of multiple VTs, and familiarity with variant features and core assets. As the size of the system grows and the feature dependencies increase, the above inconveniences aggravate. These observations encouraged us to experiment
with a single VT to manage the common variability situations found in TMS core assets in a uniform and traceable way.

### III. SINGLE VARIATION TECHNIQUE APPROACH TO TMS CORE ASSETS

**XVCL** [9][15], based on Frame Technology [2], is a generative language-independent variation technique for SPLs.

XVCL encapsulates core assets in so-called *x-frames*. Coarse-grained features are contained in dedicated x-frames. Each variation point in core assets is marked with a suitable XVCL *command*, such as `<adapt>`, `<insert-before>` `<insert>`, `<insert-after>` and `<break>`, to enable customizations. SPL variant features are formally mapped into all the relevant variation points in core assets by means of XVCL parameters and commands. The SPeCification x-frame, called *SPC*, sets values of XVCL parameters according to feature selection. XVCL Processor interprets x-frames starting from the *SPC* (Fig. 4), traverses x-frames, propagates customization information (parameters) to them, adapting visited x-frames accordingly, and emitting code for a custom product. XVCL mechanisms allow us to manage features with fine-, coarse- and mixed-grained impact on core assets. Due to its language-independence, any type of SPL artifacts including Java code, JSP files, DB scripts, WORD files, test cases or even UML models in XMI can be consistently customized for any legal selection of features required in a custom product.

#### A. TMS core assets instrumented with XVCL

Fig. 4 provides a snapshot of the WFMS core assets in XVCL representation, and Fig. 5 expands some x-frames to highlight the working mechanism of XVCL. The *SPC* specifies which features we need in a custom WFMS product by setting values for XVCL parameters that correspond to selected features. Values of those parameters propagate to x-frames below, navigating configuration and detailed customizations of core assets and features accordingly. *Level 2* x-frames define architecture-level customizations, in terms of configuration of core assets for a custom WFMS product. Some of the coarse-grained feature impacts are also addressed at *Level 2*. *Level 3* x-frames contain the actual code of core assets and instrumented with XVCL commands to enable customization of fine-grained features.

Features we want to select for a custom product are assigned non-empty string values, while features to be deselected are assigned empty string values. Thus, *SPC* shown in Fig. 5 selects features *IDCard* and *SSO* (related to *Login*), and feature *PayByItem* (related to *Paymode*) for a custom product. It deselects feature *Direct*, *PayByYear* and *PayByYearOrder*.

*<select> commands mark variation points in x-frames below *SPC*. The value of an XVCL parameter that controls *<select>* identifies an *<option>* to be processed. *<select>* *PayByItem* in x-frame *OnlinePayment* at Level 2 illustrates a simple variation point affected by one feature only, namely *PayByItem*. If feature *PayByItem* is selected, then the Processor emits feature code to the custom product; otherwise, *<select>* has no effect.

---

2 Details on TMS’s XVCL solution are available at this link: [www.comp.nus.edu.sg/~yinxing/TMS-XVCL-solution.html](http://www.comp.nus.edu.sg/~yinxing/TMS-XVCL-solution.html)
IV. EVALUATION

What are the implications of replacing multiple variation techniques with a single one on SPL productivity? To answer this question, we conducted lab studies and collected inputs from Fudan Wingsoft Ltd. regarding the original WFMS core assets developed by Wingsoft using multiple variation techniques, and core assets in XVCL. Below, we comment on productivity during domain engineering (i.e., building and evolving core assets), and product derivation.

A. Domain engineering effort

The original WFMS core assets were built by gradual re-engineering of existing WFMSes. Core components and their interfaces were stabilized first, and then variation techniques were used to prepare them for ease of customization, as described in Section II. While it is difficult to precisely determine the effort to build core assets, we obtained some relevant information from Wingsoft engineers who were involved in re-engineering. Selecting suitable variation techniques for various features was not difficult for experienced engineers. Also, each step of applying variation techniques was quite simple. New staff joining the Wingsoft team had little difficulty to understand the variation techniques used in WFMS core assets and their role. However, some problems could be observed during evolution of the WFMS core assets. When multiple variation techniques were used together to implement a variant feature, it might not be clear how to find all the relevant variation points, and understand the exact interplay between variation techniques. Still, given the size of WFMS core assets and relatively small number of features, the solutions adopted by Wingsoft team were considered to be adequate for the purpose.

To get insights into the effort of unifying multiple variation techniques with XVCL, one PhD student and one developer re-engineered the original WFMS core assets into XVCL representation. PhD student was an XVCL expert, and the
developer was a WFMS expert, also participating in maintenance of the original WFMS core assets. It took two weeks for them to replace multiple variation techniques with XVCL in core assets for TMS subsystem. Applying XVCL was greatly simplified, as core assets were already in place, and they preserved most of the variation points. The main task was to work out overall XVCL controls and then to replace multiple variation techniques with XVCL commands at respective variation points.

Evolution of core assets involves adding new features and modifying features. The effort to evolve core assets depends on the number of variation points involved in change, and the complexity of finding, analyzing, changing variation points and tracing the impact of change. While the number of variation points in both solutions is almost the same, we assume that evolution of XVCL solution is easier than evolution of the original solution. This is due to uniform treatment of features, formal links between all the variation points relevant to a given feature, and feature query system [10].

B. Product derivation and maintenance effort

Deriving new products includes reuse of existing features, modifying features, and implementing extra features required by product customers. Similarly, the effort of each such task depends on the number of variation points involved in product customization, and the complexity of finding, analyzing, customizing variation points and tracing the impact of change.

Table I summarizes statistics relevant to product derivation effort. “Managed variation points” means variation points that have to be revised manually when reusing or modifying features. “Managed variation points” is a subset of all the variation points at which one feature affects core assets. For example, among core assets affected by feature InitPayMode are Java files and DB schema files. To reuse this feature in the original WFMS-PL, all affected files need to be manually changed. In the XVCL solution, once we <set> value of XVCL parameter InitPayMode in SPC (Fig. 5), all the customizations for feature InitPayMode spark from there, can be found by feature queries, and automatically performed by the XVCL Processor. Therefore, feature InitPayMode requires only one managed variation point in XVCL solution.

<table>
<thead>
<tr>
<th>TABLE I. MANAGED VARIATION POINTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>#variation points</td>
</tr>
<tr>
<td>Original WFMS core assets</td>
</tr>
<tr>
<td>XVCL WFMS core assets</td>
</tr>
</tbody>
</table>

As another example, core assets affected by feature Settlement include seven Java files, four JSP scripts, one configuration file, and one file containing DB schema, totally 13 variant points. To reuse feature Settlement in the original WFMS-PL, we must customize code at 8 managed variation points handled by conditional compilation, comments and Ant. The location of managed variation points as well as relationship among them is not formally captured, therefore must be communicated via external documentation or re-discovered when needed. In XVCL solution, for the same feature there are also 13 variation points, but only 3 managed variation points (XVCL parameters for Settlement and for two dependent features). All the variation points are inter-linked via relevant XVCL parameters <set> in SPC, and reuse of the feature is automated by the XVCL Processor.

C. Other inputs from Wingsoft

Besides the above comparison study, we also collected some feedback and comments on the XVCL solution from interview with several Wingsoft engineers.

Comments on code readability. Both XVCL representations and the original final-boolean conditional compilation and commenting out applied variation techniques to embed fine-grained feature code in the code of core assets at relevant variation points. Wingsoft engineers reported about 30% of code in class FeeOrder, 20% of code in FeeInfo and 35% of code in FeeUser was managed by final-boolean conditional compilation and commenting out (the respectively similar percentages managed in XVCL representations).

Comments on copyright protection. In the original WFMS core assets, run-time binding variation techniques such as design pattern and configuration parameters are widely used. Therefore, Wingsoft engineers often included unnecessary feature code into a custom product release because of the characteristics of runtime variation binding and also time involved in feature removal. When extra functionality is contained in files that are released in readable form (e.g., JSP or XML configuration files), this practice can sometimes create copyright problems, as other customers may use extra functionality that was not meant for them or not included in their licenses. In XVCL, unwanted features are never included into a custom product, as the job of feature manipulation is consistently and automatically done by the XVCL Processor. Other than protecting copyrights, such precise and flexible control over feature inclusion/exclusion to/from custom products also matters in situations when we need to build highly optimized products, for example embedded software.

D. Evaluation summary and comments on relation work

Overall, it was felt that for small-to-medium systems such as WFMS (around 50KLOC), adopting multiple variation techniques is still practical. Variation techniques used in the original WFMS are simple and known to most of engineers. They came into engineers’ mind naturally, could be applied on the fly during core asset design, with minimum disruption of conventional programming. Multiple traditional VTs provide an elementary infrastructure for SPL support. Handled by the experienced engineers, the original WFMS core assets serve well for the derivation of almost 100 product variants.

As the size of core assets and the number of variant features grows, and feature interactions get more complicated, problems may show up. Feature reuse and maintenance may become more complex because of the many variation points at which feature code needs be understood. Manual customizations become time-consuming and error-prone, even for skilled domain engineers. Then, it may be worth to consider migrating to a uniform variation technique such as XVCL.

In XVCL, for a feature reused as-is we need small number of managed variation points, at which we <set> XVCL
parameters for that feature and its dependent features (in SPC). All the variation points for a given feature are formally linked to XVCL parameter representing that feature. The ability to locate and analyze traces of customizations for each feature helps developers reuse and modify features with less errors and unwanted side-effects as compared to working with the original WFMS core assets. Reuse is automated by the XVCL Processor.

However, the adoption of XVCL is not without pitfalls, some of which XVCL shares with other variation techniques. Much of the code of features still remains tangled with core assets, affecting readability. This is a big problem, but so far alternative approaches based on specification-based variation points such as AOP [13] or FOP [3] failed to provide an effective solution to fine-grained feature management in SPLs [11][12].

Industrial tools such as GEARS [4] and pure::variants [14] could certainly manage the WFMS SPL. However, we do not have hands-on experience with those tools or specific studies to provide detailed comparison. GEARS can handle configurable software artifacts — such as source code, test cases and requirement documents. Its capability is similar to the XVCL. Pure::variants captures the problems (family model) and the solutions (variant model), which records the customized feature models for product variants) separately and independently, to reuse the solutions and the feature models in new projects.

The new emerging academic tool FEATUREHOUSE [1], in virtue of the FST, the direct annotation in artifacts is avoided and the readability is not undermined. The trade-off is that it has to integrate the various adapters and computation rules for the different languages. Since no annotation inside the artifacts for feature code at arbitrary granularity, we find that FEATUREHOUSE has to adopt hook a method to deal with the fine-grained impact of features. Compared with XVCL, FSTCOMPOSER in FEATUREHOUSE is flexible in supporting additional features. It cannot really change an existing fragment. But XVCL is more flexible in what can be variable.

V. CONCLUSION

This study was conducted jointly by Fudan Wingsoft Ltd., and researchers at Fudan University and National University of Singapore (NUS). Our earlier study of a Wingsoft Software Product Line (SPL) revealed that applying multiple variation techniques poorly scale to larger SPLs. In particular, it may become increasingly difficult to find and understand feature code scattered through core assets (so-called fine-grained features), and to coordinate changes at multiple inter-related variation points. Multiple variation techniques also hinder readability, reuse and evolution of core assets heavily affected by fine-grained features.

In this paper, as a remedy to the above problems, we presented an approach based on a single, uniform variation technique of XVCL, capable of managing both fine-grained features, as well as features whose impact requires customizations at the product architecture/component level. We evaluated the XVCL-based product line representation in lab experiments, and in Fudan Wingsoft Ltd, a company that initially used multiple VTs and then applied XVCL.

An SPL in our study was small, with only 39 features. Feature dependencies were few and feature impact on core assets was not very complicated. Still, we could sense some of the above problems. However, as the impact of features on core assets accumulates and gets more complex, understanding and synchronizing multiple, poorly compatible variation techniques may become difficult. Other SPLs may contain much larger code base in core assets, with hundreds or thousands independent features [6]. We believe that in case of such SPLs, the problems reported in our study are bound to become more severe, and a single variation technique approach even more attractive.

ACKNOWLEDGMENT

This work was supported by Fudan University grant (the NSF of China under Grant No. 60703092), and National University of Singapore grant R-252-000-336-112.

REFERENCES