A Review on Variability Mechanisms for Product Lines

Jihyun Lee and Sunmyung Hwang*
Daejeon University, 62, Daehak-Ro, Dong-Gu, Daejeon, Korea
{jihyun30, sunhwang}@dju.kr

Abstract. Software product line (SPL) is an approach that develops the family of similar software by maximizing the reusability of development artifacts. Together with commonality, variability that differentiates a product from others plays a pivotal role in SPL. During decades lots of variability mechanisms have been developed for software development, and they have been used in SPL as they are. And new mechanisms have been proposed in SPL. This paper gathers a set of data from the existing literatures that describe variability mechanisms, and thereafter try to find out the state of the practices and possible weaknesses in the practices

Keywords: Software Product Line, Variability Mechanism, Variability Realization, Platform, Variability.

1 Introduction

Mass customization means to provide personalized products or services to customers. Variability differentiates a product from other products and introduces products’ flexibility, which is the basis for mass customization. Variability plays a pivotal role for realizing mass customization. Software product line (SPL), which is an approach that develops a family of similar software through systematic variability management, is an approach that realizes mass customization in software development.

Variability mechanism means a method for implementing the variability of software product line, and it incorporates variability into the product line. During decades variability mechanisms have been used in the product line engineering arena. Variability mechanisms have been used in single software development and most mechanisms that have used well in product line are those defined in single system development approaches. This paper reviews variability mechanisms defined in the existing literatures and finds out the state of the practices and possible weaknesses in the variability mechanisms. The remainder of this paper organized as follows: Section 2 describes the selection of literatures and Section 3 provides the review results. And Section 4 provides conclusion remarks.

* Corresponding author.

© SERSC 2013
2 Literature Selection

We screened published books and papers through IEEE, ACM, and other conferences. Among the screened literatures following literatures describe variability mechanisms exactly:

• Anastasopoulos et al. [1]: Implementing Product Line Variabilities.
• Jacobson et al. [8]: Software Reuse: Architecture, Process, and Organization for Business Success.
• H. Gomaa [7]: Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architecture.
• Muthig et al. [10]: Generic Implementation of Product Line Components.
• Atkinson et al. [2]: Component-based Product Line Engineering with UML.
• Batory et al. [4]: The GenVoca Model of Software-System Generators.
• Kiczales et al. [9]: Aspect-Oriented Programming.
• OMG [11, 13]: Common variability language (CVL).

3 Variability Mechanisms in Practices

Jacobson et al. [8], Anastasopoulos et al. [1], Pohl et al. [12], H. Gomaa [7], and Muthig et al. [10] suggest mechanisms for realizing variability with the phase used which means the time of variability specialized. Jacobson et al. [8] discuss the mechanisms for supporting variability in component based development while Anastasopoulos et al. [1] expound on variability mechanisms in object-oriented technique including brief discussions on variability mechanisms from frame technology, aspect-oriented programming, and design patterns viewpoints. Table 1 depicts the summarized results of variability mechanisms described by those literatures together with phase used and their binding times.

Table 1. Variability mechanisms in widely used approaches

<table>
<thead>
<tr>
<th>Variability mechanism</th>
<th>Phase used</th>
<th>Binding time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extension</td>
<td>Requirements</td>
<td>Requirements time</td>
</tr>
<tr>
<td>Uses</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Plug-in</td>
<td>Architecture design</td>
<td>Architecture design time</td>
</tr>
<tr>
<td>Inheritance/ Overloading</td>
<td>Realization(detailed design and coding)</td>
<td>Realization time</td>
</tr>
<tr>
<td>Parameters</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Template instantiation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pre-compiler macro</td>
<td>Realization</td>
<td>Pre-compilation time</td>
</tr>
<tr>
<td>Conditional compilation</td>
<td></td>
<td>Compile time</td>
</tr>
<tr>
<td>Makefile parameter</td>
<td>After realization</td>
<td>Compile or link time</td>
</tr>
<tr>
<td>Static library</td>
<td>After realization</td>
<td></td>
</tr>
<tr>
<td>Configuration</td>
<td>After realization</td>
<td>Previous to runtime</td>
</tr>
<tr>
<td>Generation</td>
<td></td>
<td>Before or during runtime</td>
</tr>
<tr>
<td>Dynamic link libraries</td>
<td>After realization</td>
<td>Runtime</td>
</tr>
</tbody>
</table>
There are approaches for implementing variability such as GenVoca [4], Generative programming [6], Model driven architecture, KobrA [2], and Aspect Oriented Programming (AOP) [9]. The way these approaches implement variability can be summarized as following:

- **GenVoca**: Each abstract data type or feature is represented as a separate layer and concrete components (or systems) are defined by type expressions describing layer compositions.
- **Generative programming**: Parameterization represents families of components. Parameters have dependencies referred to as horizontal configuration knowledge.
- **Model-driven architecture**: PIM describes important characteristics of the domain in terms of classes and their attributes and then PSM chooses which technologies will be used.
- **KobrA**: The representation of the generic component model is implementation independent and the most appropriate kind of implementation method can be used.
- **AOP**: Different aspects may be weaved together into a single piece of code (in accordance with chosen aspects weaved codes differ).
- **Generic programming**: This allows components customizable while statically configured codes are retailed through parameterization.

Common Variability Language (CVL) which is a domain specific language and in progress of standardization in OMG defines variability mechanisms [11]. CVL provides mechanisms such as Substitution, Existent, Value assignment, Opaque, Choice, Parametric, and Repeatable. AUTOSAR [3] describes that variability can exist within the internal behavior of software components. AUTOSAR provides Virtual Functional Bus (VFB) that describes communication mechanisms and VFB defines the three elements of variability; software component variability, port variability, and connector variability. However, AUTOSAR does not mention about their implementation.

### 4 Conclusions

This paper reviewed variability mechanisms, the ways for implementing variability, supported by the widely used approaches such as object oriented approach and component based development including related programming languages and by the innovative approaches such as aspect oriented programming, Genvoca, generic programming, generative programming, KobrA, and model driven approach. Especially, we reviewed CVL, which is developed for variability modeling and ongoing standard by OMG. The review results concludes that there were several kinds of variability mechanisms introduced by approaches for the single system development while there were few variability mechanisms introduced by approaches for the product line development approaches [16, 17].
References

3. AUTOSAR: Virtual Functional Bus V2.2.0 R4.0 Rev3 (2011)