EVALUATING PRODUCT LINE TECHNOLOGIES: A GRAPH PRODUCT LINE CASE STUDY

Liya Chakma [1], Hongyu Zhang [1]

[1] School of Computer Science and Information Technology,
RMIT University

ABSTRACT: In recent years the software product line approach has emerged as a promising way to improve software productivity and quality. Many technologies, such as GenVoca, XVCL and Template Metaprogramming, have been proposed to develop reusable product line assets. An extensive evaluation is required in order to understand the relative strengths and weaknesses among these technologies. Lopez-Herrejon and Batory proposed the Graph Product Line (GPL) as a standard problem for evaluating product line technologies [LHB01]. In this paper, we describe the development of the Graph Product Line using XVCL. We then compare our solution with the GenVoca solution presented by Lopez-Herrejon and Batory. We perform an evaluation between GenVoca and XVCL based on our experiments.

INTRODUCTION

In recent years the software product line approach has emerged as a promising way to improve software productivity and quality. A *Software Product Line* is a family of similar software applications or systems that share a common set of core assets [LHB01]. An asset could be any type of reusable software artifact that are produced during the software development life cycle. For instance, an asset could be a requirement specification, a generic architecture, or programs of a system family. A new family member can be instantiated by reusing the assets of a software product line. The benefits of software product line is its practicality and economy. With product line practice, it is possible to provide higher quality products with lower development and maintenance costs in a shorter period of time.

Many technologies such as GenVoca [BLHM02], Frame Technology [JBZZ03], Object Oriented reuse techniques, Aspect Oriented Programming [KHH+01] and Template Metaprogramming [CE00] have been proposed and used to develop software product lines. However, due to the fact that their relative strengths and weaknesses are not well understood they are often chosen arbitrarily, based on convenience rather than fact. To get the most benefit from the product line approach and to improve the current technologies we need to understand the underlying concepts behind these technologies. In order to accomplish this task we need to experiment and evaluate them in a particular domain on a common set of problems. A standard problem, Graph Product Line (GPL) has been proposed by Lopez-Herrejon and Batory [LHB01]. Graph is a fundamental topic in computer science and choosing it minimises the requirement of becoming a domain-expert which is a prerequisite in the product line development process. Still it is complex enough to expose the underlying concepts of product lines and their implementation. As a reference point, Lopez-Herrejon and Batory presented a solution to this problem using the GenVoca technology.

The aim of this paper is not to rate the technologies but to investigate their differences and similarities (if there are any). We choose GPL as an example of a problem domain. We then implement the GPL using XVCL, a frame based technology that supports product line development. Finally, we compare our solution to the GenVoca solution and evaluate them based on the way they decompose and compose the reusable assets, the way they handle variant requirements, and language and tool support.

The organization of this paper is as follows: in Section 2 we describe the background, in Section 3 we describe the Graph Product Line, in Section 4 we describe a reference implemenation and describe our XVCL-based approach, in Section 5 we evaluate XVCL and GenVoca, in Section 6 we discuss some related work and finally, in Section 7, we conclude the paper.

BACKGROUND

In this section, we briefly introduce the software product line, domain engineering and two technologies that support product line development, namely XVCL and GenVoca.

Software Product Line and Domain Engineering

Clements *et al.* define software product line as *a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way* [CN01]. Features are distinguishable characteristics of a system or a system family that are relevant to some stakeholders. In a product line, the members of the system family share some common features and differ in certain variant features. Unlike a single system development, product line development, therefore, requires a careful process that identifies the common and variant features among family members for future product generation. This process is often termed as Domain Engineering. It involves identifying, describing and storing the reusable assets and providing the means (methods and tools) for reusing these assets when building new systems.

XVCL

XVCL (XML-based variant configuration language) [JBZZ03] is an XML-based frame language that supports product line development. It is based on the concept of *Frame* that was first introduced by Minsky in his paper "A FrameWork for Knowledge Representation" in 1975. The idea of frame relies on human psychology in that we keep our experiences in frames as memory. In the event when we need them we collect them from the memory and use them by adjusting them to a new situation. According to Minsky a frame may be viewed as a static data structure to represent stereotyped situations. A frame has been used widely as a structured knowledge representation scheme.

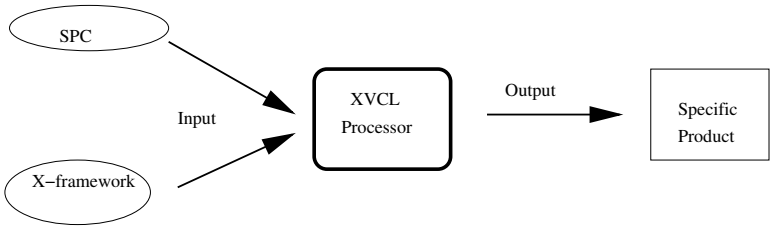

Figure 1: The Process of Product Generation with XVCL

In XVCL, frames are termed x-frames. An x-frame is a reusable code fragment stored as XML document. In Figure 1 we show the process of product generation with the XVCL technology. An SPC is a specification x-frame for a specific product and an x-framework is a collection of inter-related x-frames. The collection preserves the reusable assets. The SPC and x-framework together are processed by the *XVCL Processor* to generate custom products. The XVCL processor plays the role of a composer that performs the composition and customization of the reusable assets. More information on XVCL can be found at the XVCL homepage: http://fxvcl.sourceforge.net.

GenVoca

GenVoca is a software product line technology that is based on step-wise refinement [BSR03]. Step-wise refinement is a powerful paradigm for developing complex systems by adding features incrementally. In Figure 2 we show the process of product generation with the GenVoca technology. It is based on algebraic specifications. In GenVoca, programs are treated as constants and refinements are treated as functions that take constants as inputs. By applying refinements to constants GenVoca can add features incrementally and produce more feature-augmented output. To support GenVoca a model *AHEAD* (Algebraic Hierarchical Equations for Application Design) and a tool suit is developed by Batory et al. [BLHM02]. The main tool of the AHEAD is the *composer*, which takes an equation as a command-line input, recursively expands that equation and forms a composite directory of features. The code implementing features are not in pure programming language rather a superset of that programming

language. Another tool *jak2java* converts the composed files into Java programs. More on AHEAD can be found at the AHEAD homepage: http://www.cs.utexas.edu/users/schwartz/ATS.html.

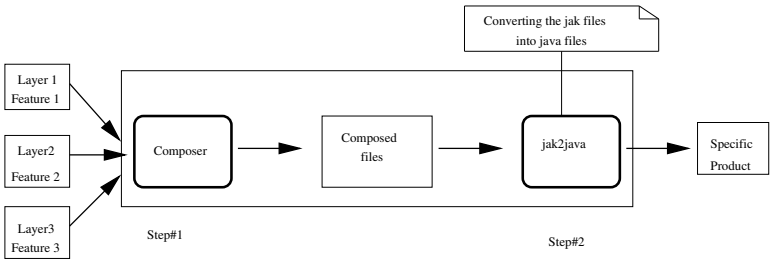

Figure 2: The process of Product Generation with GenVoca

AN OVERVIEW OF THE GRAPH PRODUCT LINE

In order to evaluate the GenVoca and XVCL technology we choose a Graph Product Line as a case study. The *Graph Product Line* (GPL) [LHB01] is a family of classical graph applications. Like any other product line, members of the GPL share some common features and differ in certain variant features. For instance, all the GPL members must have features such as vertex and edge. But some of the members may implement Breadth First Search (BFS) as a searching algorithm, some other members may implement Depth First Search (DFS) and yet, other members may not implement any searching algorithm. Some of the common and variant features that the GPL members share are identified as follows:

- **Common Features:** The common features that the GPL applications share are *Vertex* and *Edge*. Which means a GPL application must have these features.
- **Variant Features:** Members across the GPL share some variant features:
  - *Graph Type:* A graph is either directed or undirected.
  - *Weight:* Edges in a graph can be weighted with non-negative numbers or unweighted.
  - *Search:* A graph application can implement at most one searching algorithm, Breadth First search (BFS), Depth First Search (DFS) or none.
  - *Algorithms:* A graph application implements one or more of the following algorithms: *Number* (Vertex Numbering), *Connected* (Connected Components), *Strongly* (Strongly Connected Components), *Cycle* (Cycle Checking), *Shortest* (Single-Source Shortest Path) and *MST* (Minimum Spanning Tree). Details of these algorithms can be found in any algorithm book such as [LCR90] and more information on GPL can be found at: http://www.cs.utexas.edu/users/dsb/GPL/graph.htm.

DEVELOPMENT OF THE GRAPH PRODUCT LINE

Reference Implementation with GenVoca

Lopez-Herrejon and Batory [LHB01] provided a reference implementation. They converted the conceptual objects that the GPL has such as Graph, Vertices, Edges and Neighbors into physical objects by defining distinct classes. The *Graph* class contains a list of Vertex objects and a list of Edge objects. The *Vertex* class contains a list of Neighbor objects. The *Neighbor* class contains a reference to a neighbor Vertex object and a reference to the corresponding Edge object. The *Edge* class extends the Neighbor class and contains the start Vertex of an Edge. Their solution helps the user by adding features incrementally to the base feature. As mentioned earlier in the Background section, GenVoca takes the input as equations and generates applications as follows:

```
GPLapplication1 = Number(Connected(Weighted(DFS(Directed))))
GPLapplication2 = MST(Connected(Weighted(BFS(Undirected))))
GPLapplication3 = Shortest(Weighted(directed))
```

```
//First part: feature Directed is implemented as a
//layer Directed with three inner classes
public class Directed {
   class Graph     {....}
   class Vertex    {....}
   class Neighbor  {....}
   class Edge      {....}
}

//Second part: refinement of the feature Directed by adding the DFS
//feature. The superclass here is passed as a parameter.

public class DFS$$<Directed> extends Directed {
   class Graph$$DFS     extends Graph$$Directed {....}
   class Vertex$$DFS    extends Graph$$Directed {....}
   class Neighbor$$DFS  extends Graph$$Directed {....}
   class Edge$$DFS      extends Graph$$Directed {....}
}
```

Figure 3: An example of the generated code in the reference implementation

To implement refinements Batory and Lopez-Herrejon used mixin layers [SB98] where features are implemented in layers and where layers are collections of classes. A refinement, the addition of a feature is implemented by inheriting the classes from the base layer. We illustrate their implementation using the first application, GPLapplication1. The base feature here is Directed and it is implemented as a layer which has four inner classes, Graph, Vertex, Neighbor and Edge as shown in the first part of Figure 3. A refinement on this feature is an addition of the DFS feature which is implemented with another layer with four inner classes. The mixin-layer allows the current layer to extend the previous layer and the name of the superclass is passed as a parameter as shown in the second part of Figure 3. The rest of the feature set of the GPLaplication1 is implemented in the same way by extending the previous layer.

An XVCL-based Approach

We design an x-framework for GPL according to the principles of separation of concern. We try to design each x-frame so that it is concerned only with a feature or a set of features. For instance, 'Search' x-frame is concerned only with the code related to the *Search* feature which can be of either BFS or DFS. Our second concern is the representation of the graph. We try to minimise the number of classes so that the inter-x-frame dependencies are minimised. Finally, we come up with a solution

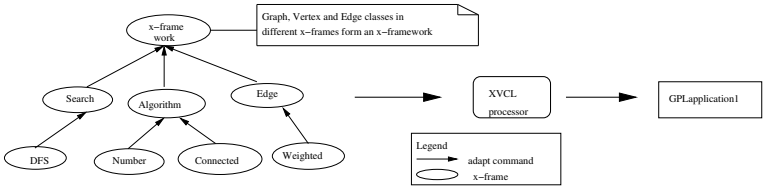

Figure 4: The generation of GPLapplication1 in XVCL

where the graph is represented as three classes, Graph, Vertex and Edge. Features are implemented as x-frames. Every x-frame that implements a feature contains the code fragment that implements the feature. We use XVCL command [ZJ04, JZ01]<adapt> and <select> to achieve this. In Figure 4, we use a specific example, GPLapplication1 as mentioned in the previous section, to show how this adaptation works. The top x-frame represents an x-framework that has three classes in three diferent x-frames. These x-frames adapt the common feature *Search* which is implemented as the Search x-frame. The Search x-frame adapts the DFS x-frame to implement the *DFS* feature. In a similar way,

```
<?xml version="1.0"?>
<!DOCTYPE x-frame SYSTEM "file:///xvcl.dtd">
<x-frame name="Graph.xvcl" language="java" outfile="Graph.java">
<set var="Search" value="VERTEX_DFS"/>
..... <!-- Some more code fragments here -->
<select option="Search">
   <option value="VERTEX_BFS">
      <set var="BFS_Code_Fragment" value="VERTEX_BFTNODE_SEARCH"/>
      <adapt x-frame="BFS.xvcl"/>
   </option>
   <option value="VERTEX_DFS">
      <adapt x-frame="DFS.xvcl"/>
   </option>
</select>
..... <!-- Some more code fragments here -->
</x-frame>
```

Figure 5: An example of the selection of features in XVCL

x-frames for *Connected* and *Number* features are adapted by the Algorithm x-frame, and *Directed* and *Weighted* x-frames are adapted by the Edge x-frame. In this way, we generate a product with Directed, DFS, Weighted, Number and Connected features. By using the same adaptation and selection process we can generate applications that are composed of any combinations of features.

Figure 5 shows a typical x-frame that uses <select>, <option> and <adapt> command to select a feature and then adapt the corresponding x-frame. We use a variable to hold the value of a feature. Here, the variable 'Search' is used in order to identify which search feature the application will implement. The <option> command allows us to incorporate the related features, here BFS and DFS. The example application we use here therefore, will adapt the DFS x-frame.

EVALUATION

In this section, we evaluate GenVoca and XVCL using the GPL application discussed earlier in Section 4. This application (GPLapplication1) implements vertex numbering, connected components algorithms on weighted, directed graph.

Decomposition

Decomposition of software assets is an important issue in evaluating technologies that support development of software product lines. It reduces the complexity of the software assets and encourages reuse. In our project our first concern is the features. We want to develop the GPL that separates GPL assets into different components that are concerned with a particular feature. Our experience reveals that the decompositions of GPL assets are straightforward in both XVCL and GenVoca. In GenVoca feature components are layers. Twelve layers were used in the reference implementation. Every layer represents a particular feature that is comprised of different classes (or class like artifacts such as .jak files). In XVCL GPL assets are decomposed into x-frames. Each x-frame is concerned with a particular feature and thereby supports separation of concerns.

Composition

GenVoca composes the features dynamically and it refines the previous state of the component that was modified by the previously selected features. GenVoca achieves this by step-wise refinements. In Figure 6 we show how GenVoca composes and refines units of layers (here classes) The bottom most class in the inheritance chain is the class we use to generate the final code. For a domain like GPL GenVoca technology is probably acceptable. But the weakness of this technology is that the inheritance hierarchy could be very long in a large domain that has many features.

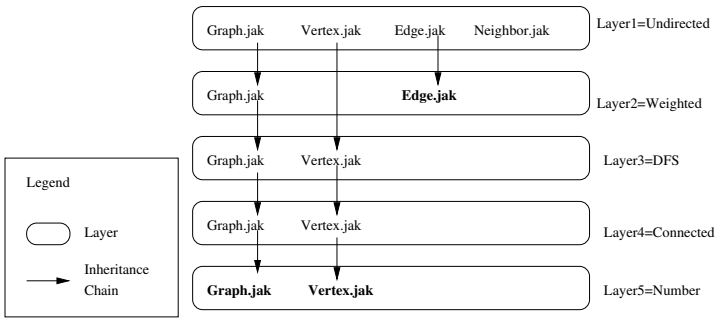

Figure 6: The composition of the GPLapplication1 in GenVoca

The composition of the reusable GPL assets generates specific products. In XVCL, we use the <adapt> command to compose assets from feature specific x-frames. Figure 7 shows how XVCL composes the x-frames and generates reusable code. Again, we illustrate composition with XVCL by using the same example. The composition in XVCL is done at construction time and the <adapt> command is used to accomodate one specific feature. We design the Graph x-frame to adapt the Search x-frame for the DFS feature. The Search x-frame then adapts DFS x-frame. The same goes for the other x-frames. Each x-frame represents a particular feature in GPL domain. However, features are not completely independent. There are cross-cutting features (adding/changing one feature affects other features). Some features, for example involve more than one classes and yet another feature can be dependent on one of those classes. Here, the Algorithms x-frame was adapted twice; first, directly for the class Graph and second, indirectly through Vertex x-frame for the class Vertex. This is an impact of feature interactions and it is not always possible to design x-frames that are completely independent of other features.

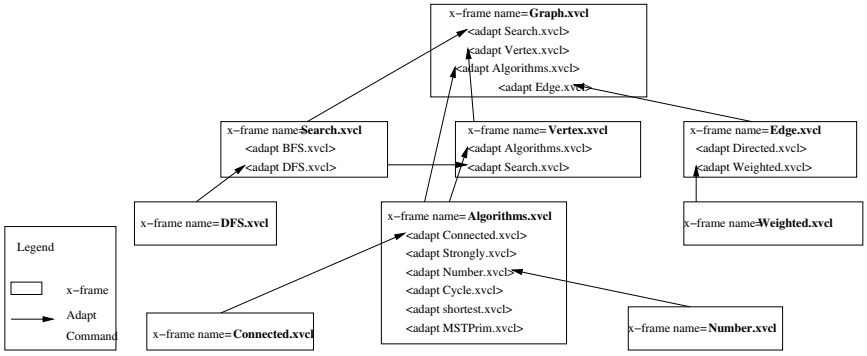

Figure 7: The composition of the GPLapplication1 in XVCL

Variability Mechanism

Variability [JB02] is the ability of a system to change or evolve. As we mentioned earlier in the Section 3, different GPL applications have different combinations of features. Therefore, every GPL application is unique and requires the system to change for every application. Since GenVoca uses step-wise refinements it does not require any extra facility to handle the variants among the GPL members. From the input equation it adds the features incrementally. XVCL, on the other hand, uses specific commands to select a specific feature. In Figure 5 we show an example on how the <select> and <option> commands select variant code fragments for BFS and DFS.

Generated Code Size

Figure 8 indicates that the generated code size in GenVoca is larger than the generated code size in XVCL, for three classes. The differences grow as the number of feature grows. As expected, this is due to the generation of intermediate abstract classes in GenVoca (for different layers), which makes the inheritance chain longer. Therefore, the difference is proportional to the inheritance chain.

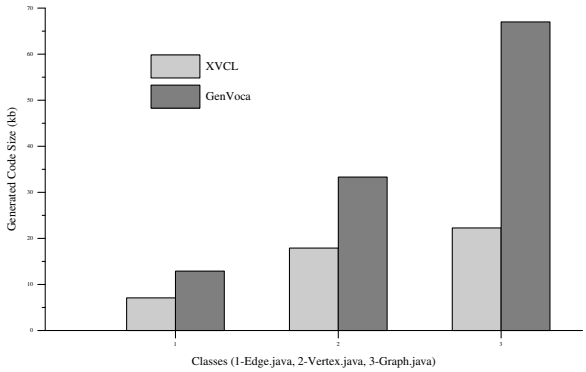

Figure 8: The size of the generated classes in XVCL and GenVoca

Language and Tool Support

The wide acceptance of a technology depends on how flexible it is in supporting a programming language. GenVoca supports programming languages such as C++, Java and Ada. Since XVCL is an XML based language it has the advantage of supporting any programming language. GenVoca provides the AHEAD tool suit and XVCL provides the XVCL processor to compose software assets during product line development.

RELATED WORK

Ferré and Vegas [FV03] performed an evaluation of the Domain Analysis methods to systematically reuse artifacts. Their work is based on the literature survey and no implementation is provided, we present an evaluation based on our experiences during the development process.

Matinlassi [Mat02] presented an evaluation of architecture design methods that support product line development. She performed a survey and defined a framework to evaluate them. The framework she defined is concerned with issues such as the process and use of that method. It addressed the questions such as ”what skills does the user need to accomplish the tasks required by the method” or ”what are the domains the method is validated in”. We also evaluate different technologies that support product line development. Our project, however, is concerned with technical issues such as how these technologies decompose or compose the reusable assets.

Literature has addressed how to capture variant and common features in domain models and architecture level, however, little attention has been paid to how to actually implement this generosity at the code level. Anastaspoulos and Gacek [AG01] outlined a number of qualities that can be associated with handling variants in implementing Product Line assets. They define a framework for evaluating different approaches for implementing product line, depending on the requirements that support implementation.

CONCLUSION AND FUTURE WORK

In this paper, we identify some of the common and variant features of the GPL. We describe a reference implementation of the GPL in GenVoca provided by Lopez-Herrejon *et al.* and the GPL implementation in XVCL. We then compare our XVCL-based solution to the GenVoca solution. We perform an evaluation based on the experience from the development process. In particular, we show how they decompose and compose the GPL assets and support the generation of specific GPL applications. Our findings suggest that, in GenVoca variant features are resolved at the time of composition, while in XVCL they are resolved before the composition by the means of some of the selection commands. In GenVoca, as the number of feature grows the inheritance chain also grows proportionately. In future, we would like to evaluate some other technologies that support product line development such as Aspect Oriented Programming and Template Metaprogramming.

ACKNOWLEDGEMENT

We would like to thank Roberto E. Lopez-Herrejon and Don Batory for providing us the code of their implemention of GPL with GenVoca technology.

REFERENCES

[AG01] M. Anastasopoulos and C. Gacek. Implementing product line variabilities. In *Proceedings Conference on Symposium on Software Reusability (SSR '01)*, Toronto, Canada, 2001.

[BLHM02] D. Batory, R. E. Lopez-Herrejon, and J. Martin. Generating product-lines of product-families. In *Proceedings International Conference on Automated Software Engineering (ASE '02)*, Edinburgh, Scotland, 2002.

[BSR03] D. Batory, J. N. Sarvela, and A. Rauschmayer. Scaling step-wise refinement. In *Proceedings International Conference on Software Engineering (ICSE '03)*, Portland, Oregon, 2003.

[CE00] K. Czarnecki and U. Eisenecker. *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley, MA, 2000.

[CN01] P. Clements and L. Northrop. *Software Product Lines: Practices and Patterns*. Addison-Wesley, 2001.

[FV03] X. Ferré and S. Vegas. An evaluation of domain analysis methods, 2003. citeseer.ist.psu.edu/360224.html, Date: 15 May, 2004.

[JB02] M. Jaring and J. Bosch. Representing variability in software product lines: A case study. In *Proceedings the second International Conference on Software Product Line (SPLC '02)*, San Diego, California, 2002.

[JBZZ03] S. Jarzabek, P. Basset, H. Zhang, and W. Zhang. XVCL:XML-based variant configuration language. In *Proceedings International Conference on Software Engineering (ICSE '03)*, Portland, Oregon, 2003.

[JZ01] S. Jarzabek and H. Zhang. XML-based method and tool for handling variant requirements in domain models. In *Proceedings IEEE International Symposium on Requirements Engineering (RE '01)*, Toronto, Canada, 2001.

[KHH+01] G. Kiczales, E. Hilesdale, J. Hugunin, M. Kirsten, J. Palm, and W. G. Griswold. An overview of aspectj. In *Proceedings European Conference on Object-Oriented programming (ECOOP '01)*, Budapest, Hungary, 2001.

[LCR90] C. E. Leiserson, T. H. Cormen, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, 1990.

[LHB01] R. E. Lopez-Herrejon and D. Batory. A standard problem for evaluating product-line methodologies. In *Proceedings International Conference Generative and Component-based Software Engineering (GCSE '01)*, Erfurt, Germany, 2001.

[Mat02] M. Matinlassi. Evaluation of product line architecture design methods. In *Young Researchers Workshop, International Conference on Software Reuse (ICSR'02)*, Austin, Texas, 2002.

[SB98] Y. Smaragdakis and D. Batory. Implementing layered designs with mixin layers. In *Proceedings European Conference on Object Oriented Programming (ECOOP '98)*, Brussels, Belgium, 1998.

[ZJ04] H. Zhang and S. Jarzabek. XVCL: A mechanism for handling variants in software product lines. In *Science of Computer Programming*, volume 53(3), pages 381–407, Elsevier Science, 2004.