

# XVCL: XML-based Variant Configuration Language

Stan Jarzabek<sup>1</sup>, Paul Bassett<sup>2</sup>, Hongyu Zhang<sup>1</sup> and Weishan Zhang<sup>1</sup>

<sup>1</sup>*Department of Computer Science  
School of Computing,  
National University of Singapore  
Lower Kent Ridge Road, Singapore 117543  
{stan, zhanghy, zhangws}@comp.nus.edu.sg*

<sup>2</sup>*Canadian Information Processing Society  
National  
2800 Skymark Ave., Suite 402  
Mississauga, Ontario, Canada L4W 5A6  
pbassett@cips.ca*

## 1. An overview of XVCL

XVCL (XML-based Variant Configuration Language) is a meta-programming technique and tool that provides effective reuse mechanisms [2]. XVCL is an open source software (<http://fxvcl.sourceforge.net>) developed at the National University of Singapore. Being a modern and versatile version of Bassett's frames [1], a technology that has achieved substantial gains in industry, the underlying principles of the XVCL have been thoroughly tested in practice. Unlike original frames, XVCL blends with contemporary programming paradigms and complements other design techniques. XVCL uses "composition with adaptation" rules to generate a specific program from generic, reusable meta-components. Program generation rules are 100% transparent to a programmer, who retains full control over fine-tuning the generated code. Despite its simplicity, XVCL can effectively manage a wide range of program variants from a compact base of meta-components, structured for effective reuse.

Variants arise naturally in software reuse and evolution, in particular, if you deal with software product lines that encompass a family of similar systems. This is exactly the context in which we have developed and applied XVCL. Suppose you have a software product that you want to run on different platforms or hardware devices. Or you deliver versions of the same product to a number of customers and these product versions differ in some functional or non-functional requirements. XVCL allows you to configure product variants from a common core of generic, adaptable and reusable meta-components. And you can apply the same XVCL concepts and mechanisms to manage variants in a range of product line assets such as software architecture, program code, test cases, technical and user-level program documentation or requirement specifications.

XVCL includes a methodology and a tool – the XVCL processor. The XVCL methodology tells you how to discover the structure of the solution for your application domain and for the types of variants you want to address. The XVCL processor automates the routine yet error-prone program construction tasks, allowing you to focus

on what is novel about your problem domains, requiring your creativity.

## 2. How does XVCL work?

XVCL works on the principle of constructing custom systems by composing generic, reusable meta-components, after possible adaptations. Adaptations of a meta-component take place at designated variation points marked by XVCL commands. This "composition with adaptation" process turns meta-components into concrete components of the custom system we wish to build.

To facilitate effective reuse, you split your program into generic, reusable and adaptable meta-components. A meta-component is an XML file with program code (written in any programming language), instrumented with XVCL commands for ease of change and evolution. XVCL commands, designed as XML tags, allow the composition of the meta-components, selection of pre-defined options based on certain conditions, etc. Meta-variables and expressions provide a powerful parameterization mechanism. Values of meta-variables are propagated across meta-components and variable scoping rules enhance genericity and reuse.

You organize meta-components into a layered hierarchy called an x-framework. Meta-components at lower-levels are building blocks for the higher-level meta-components - XVCL processor composes higher level meta-components from lower-level meta-components, after applying possible adaptations to them. An x-framework forms a base of reusable assets, such as a product line architecture, from which you build custom systems.

The XVCL processor traverses an x-framework, executes XVCL commands embedded in visited meta-components and generates custom code.

## 3. Applications and benefits of XVCL

Applications of XVCL include design of reusable program components, implementing product line architectures, design of compact, non-redundant,

therefore, easy to maintain programs, and managing variants in multiple versions of software documents and models. We envision many other applications in software and non-software domains.

XVCL is based on Bassett's frame technology [1]. Frames have been applied in industry to manage variants and evolve multi-million-line, COBOL-based, information systems. While designing a frame architecture is not trivial, subsequent productivity gains are substantial. An independent analysis showed that frames can reduce large software project costs by over 84% and their times-to-market by 70%, when compared to industry norms (refer to QSM report in [1]). By reusing skillfully structured frame architectures, you need to focus on only the 5%-15% of a program solution that is unique; the other 85%-95% is reused. These gains are due to the flexibility of the resulting architectures and their evolvability over time.

Our XVCL is extensible and, of course, free of COBOL heritage. We applied XVCL to design product line architectures for component-based systems written in Java, and using RMI, J2EE or CORBA for component communication. We designed Facility Reservation System (FRS) and Computer Aided Dispatch system (CAD) product line architectures [3,4]. An FRS helps in the reservation of facilities (such as tutorial rooms, hotel rooms, lecture theaters) and specific equipment. Different organizations such as universities, hotels, hospitals and companies, have different physical facilities and arrangements for their reservation. The similarities and differences among FRSes yield a product line. We started by selecting common and variant functional requirements to be implemented in the FRS product line architecture. Our FRSes were built as EJB™ components, organized into a 3-tier architecture. We used XVCL to create generic, reusable meta-components from which we could generate custom runtime EJB components that met functional variants for a specific FRS we wish to build.

Computer Aided Dispatch systems (CAD for short) are used by police, fire and rescue and other similar organizations to dispatch resources (e.g., police cars) to handle incidents<sup>1</sup>. CAD systems are distributed over the Internet, with components dedicated to different roles (indicated in Figure 1) running on different computers. At the basic operational level, all CAD systems are similar - they support the dispatch of units to handle incidents. However, the specific context of the operation (such as police or fire & rescue) results in many variations on the basic operational scheme. We addressed CAD variants related to functional requirements, component distribution, platforms and reliability. For example, in some CAD systems Call Taker and Dispatcher roles may be played by two different people and in others - by the

same person. This variant has impact on system functions, user interface and component distribution.

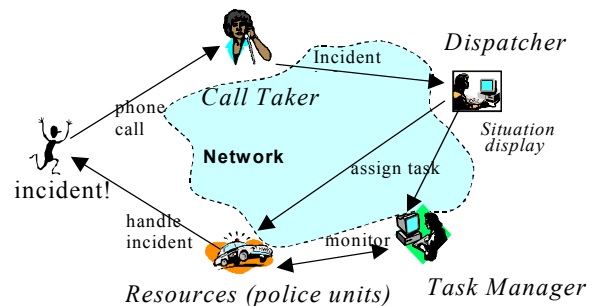


Figure 1. Highlight of a CAD system for police

We applied XVCL to manage variants in UML software models documenting FRS and CAD product lines [5]. In yet another project, we managed evolution of the City Guide System product line with XVCL.

XVCL is also effective in improving program maintenance. It is a well-known fact that redundant code obstructs program understanding and maintenance. Yet, programs are often polluted by redundant code. We analyzed a recently released Java Buffer library, JDK 1.4 Merlin and found many redundancies – the same or very similar code fragments recurring in many places. With XVCL, we generated the same Buffer library classes from compact, non-redundant meta-components whose size, in LOC, is just 40% of the original library. We posted the detailed results of this analysis at: <http://www.comp.nus.edu.sg/labs/software/xvcl/buffer.html>.

We envision many other applications in software and non-software domains, as we can apply XVCL to any domain that can be represented as a collection of textual documents.

## References

- [1] Bassett, P. [Framing software reuse - lessons from real world](#), Yourdon Press, Prentice Hall, 1997
- [2] Wong, T.W., Jarzabek, S., Myat Swe, S., Shen, R. and Zhang, H.Y. "XML Implementation of Frame Processor," *Proc. ACM Symposium on Software Reusability, SSR'01*, Toronto, Canada, May 2001, pp. 164-172
- [3] Jarzabek, S. and Seviara, R. "Engineering components for ease of customization and evolution," *IEE Proceedings - Software*, Vol. 147, No. 6, December 2000, pp. 237-248, a special issue on Component-based Software Engineering
- [4] Cheong, Y.C. and Jarzabek, S. "Frame-based Method for Customizing Generic Software Architectures," *Proc. Symp. on Software Reusability, SSR'99*, Los Angeles, May 1999, pp. 103-112
- [5] Jarzabek, S. and Zhang, H. "XML-based Method and Tool for Handling Variant Requirements in Domain Models", *Proc. of 5th IEEE International Symposium on Requirements Engineering, RE'01*, IEEE Press, August 2001, Toronto, Canada, pp. 166-173

<sup>1</sup> Project funded by Singapore National Science and Technology Board and Canadian Ministry of Energy, Science and Technology, involving National University of Singapore, Singapore Engineering Software Pte Ltd, University of Waterloo and Netron, Inc. (Toronto).