



XML-based Variant Configuration Language

<http://xvcl.comp.nus.edu.sg>

A technology for enhanced reusability and maintainability based on Bassett's frames

Technology Summary

XVCL is a public domain technology for enhanced reusability and maintainability developed in the Software Engineering Lab at NUS.

XVCL is a *variation mechanism*. It simplifies managing product variants, so-called software Product Lines (SPL). While product variants may differ in user requirements, design decisions, platforms on which they run, etc., still much similarity exists among them. XVCL helps us reuse what's similar among them, without being overwhelmed by tedious tasks of dealing with differences.

In SPL terminology, XVCL helps design *reusable core assets* for product variants forming an SPL, and automate customizations of assets during reuse-based development of product variants.

Then, XVCL helps in *follow up evolution of product variants*, to meet needs of their respective customers. XVCL's unique capability is that it can selectively propagate changes from core assets to product variants that need these changes, without affecting other product variants that do not need them. This allows us to keep evolving reusable core assets and all product variants in sync one with another.

Product variants consist of code (that may be written in multiple programming languages), documentation, test cases, and many other artifacts. Being language- and application-domain-independent, XVCL can manage variations, propagating changes across all the artifacts comprising your products.

There are plenty of *mundane, repetitive and error-prone changes* we do to software. Especially, when dealing with product variants, the same enhancement in slightly different form must be propagated to product variants. We solve the same problems all over again; replicate code sections, files and designs. Sometimes, even within a single software product there is much repetition, e.g., a similar module may recur in many variant forms – a typical situation in web portals.

XVCL eliminates the need to copy and paste, and repeat similar modifications all over again. XVCL Processor automates reuse: it propagates customizations to all the software product artifacts, configuring product components, modules, architecture and documentation, keeping all the product artifacts in sync one with another. XVCL is a simple yet effective mechanism for change that is missing in contemporary programming languages, platforms and tools.

XVCL is not yet another programming language – you still use conventional programming languages and platforms (e.g., JEE or .NET) to develop program logic, user interfaces, etc. You apply XVCL to tackle problems related to repetitions and frequent changes. XVCL works in synergy with any programming technology, and does not impose any restrictions on how systems are implemented and deployed.

The core concept behind XVCL is to represent each important recurring software structure in generic, adaptable form along with full details of its implementation.

Typical migration path to systematic SPL reuse

This scenario may be familiar to you: You developed a software product and product has been successful. New users appear but each of them has some preferences for changes, and requests for new features. That's how product variants emerge, and at the same time problems in managing them. Initially, you try to speed up development of each new product variant by copying and modifying source files from already implemented products. You store versions of source files pertinent to different products under a Software Configuration Management (SCM) tool such as CVS or SVN. As the number of customers and relevant product variants increases, such ad hoc reuse shows its limits. The product size grows. Also with a growing customer base (good for your business!), increasing product variability becomes a challenge for ad hoc reuse. It becomes difficult to select the right file versions, customize to

meet needs of a new user, and integrate them. Errors show during testing and you repeat the cycle all over again or decide to re-implement certain files from scratch. At the same time, you need maintain all the released product variants, so you have more and more code to maintain.

Usually, you start to feel the above problems as the number of product variants reaches 4-5. Adopting more systematic approach to reuse can help you sustain the business growth. The first thing you could do is to set up and stabilize component architecture (blueprint) to be shared by all product variants. Some variant features of products can be nicely mapped into product components. Handling such features becomes easy with plug-in components. A common architecture is a base for designing core assets (such as product components) to be reused in development of new product variants.

However, in most application domains, plug-in component technique is not enough to sustain reuse. The impact of variant features cannot be contained at the component level, but spreads freely across many components, affecting their code at many variation points. To manage such “troublesome” variability, developers typically adopt variation mechanisms such as preprocessing, manually commenting out variant feature code, parameter configuration files, Ant (or make), annotations (Java/JEE), etc. Such variation mechanisms are simple and available for free. Most developers can understand them without training.

Why developers often use many incompatible variation mechanisms rather than just one? The reason is that some variant features require customizations at the file level, others trigger many customizations in core asset code, yet others may involve a combination of file- and code-level customizations. Each of the common variation mechanisms is meant to handle only one type of variability situation.

The above strategy to managing product variants is simple, cost-effective and works well, as long as the number of variant features differentiating products is below 50, and the product size is below 50 KLOC. Beyond that, it shows drawbacks: Features get complicated; One variant feature may be mapped to many variation points, in many components, and it is difficult to figure out to which ones and how; Features often are inter-dependent, and inclusion of one feature into a custom product must be properly coordinated with modifications of yet other features; Core reusable assets become heavily instrumented with variation points, and using multiple techniques to manage variability makes the core assets even more complex to work with.

XVCL: one-stop solution to managing variability

Variation mechanisms cater for product-specific features, help customize components for reuse. XVCL plays the same role as the above mentioned variation mechanisms. But instead of using many incompatible variation mechanisms, we use one. Unlike other variation mechanisms, XVCL was designed with managing variability in mind. It provides a uniform way of solving the variability problems, replacing the need for multiple, ad hoc variation mechanisms, and problems they trigger.

XVCL exercises the total control over product variability, from architecture, to component configuration, to any detail of code (e.g., variations at the source statement, expression or keyword level). XVCL Processor streamlines and automates customizations involved in implementation of selected variant features into custom products, from component re-configuration, to detailed customizations of component code.

The key concept of XVCL is to represent component configuration and customization/reuse knowledge in meta-data that is both human-readable and machine-executable. In particular, component configurations that recur across product variants in similar form are represented in XVCL as generic, highly parameterized, adaptable meta-structures. These meta-structures form core assets that are adaptively reused (instantiated) during product development.

The principle of XVCL Processor operation is analogical to code expansion done by preprocessors. Unlike other variation mechanisms, XVCL is a one-stop solution to variability management at all levels of abstract, from architecture to code, and in all types of software assets, both code and documentation. XVCL complements conventional architecture-centric, component based design for reuse, and works with any conventional programming language and/or platform such as JEE, .NET, Ruby on Rails or PHP.

XVCL technology includes a *language* that adds variant configuration capability to software, *methods* guiding project application of XVCL, and *tools*. XVCL Processor is an interpreter of the XVCL notation. The Processor automates derivation of custom, executable programs from their generic meta-level XVCL

representation. XVCL Workbench is an eclipse-based plug-in with additional tools such as a static/dynamic analyzer, debugger and meta-level visualizer.

In the nutshell, XVCL offers these capabilities and benefits:

- (1) Organizing all the product assets (code, components, design, documents) for ease of adaptive reuse in development of product variants. In the processes of doing so, conventional programming languages (or any other notations) are used to write product code, and XVCL is used to instrument code components for change, and then to define product configurations at the component, architecture and system level.
- (2) Representing a complete knowledge of product configuration in human-readable and machine executable (by XVCL Processor) form. XVCL software representation shows a clear picture of similarities and differences among product variants at all abstraction and granularity levels.
- (3) Creating formal links among all the variation points in core assets that must be customized together, in consistent way during reuse.
- (4) Representing program code along with design information crucial for program understanding, maintenance and reuse, in fully integrated form.
- (5) Synergy with any other programming language, design technique or component platform.
- (6) Synergy with both formal and agile development processes.
- (7) Managing variability in all product assets including code components, test cases, end-user documentation, UML models [8], and formal specifications [17].
- (8) Incremental, lightweight adoption: low-cost start, quick results, refinement of a solution for increased productivity. Tools for similarity analysis in existing software products that help re-engineer legacy code into XVCL representation [3][4][6].

An Example

In the ASP Web Portal (WP) Product Line project [9] ST Electronics (Info-Software Systems) Pte Ltd applied state-of-the-art design methods to maximize reusability of a Team Collaboration Portal (TCP) in other contexts. Still, a number of problem areas were observed that could be improved by applying XVCL. The benefits of a generic ASP/XVCL solution were the following:

- Short time (less than 2 weeks) and small effort (2 persons) to transform the TCP into the first version of a “Generic TCP” in ASP/XVCL.
- High productivity in building new portals from the Generic TCP. Based on the Generic TCP, ST Electronics could build new portal modules by writing as little as 10% of unique custom code, while the rest of the code could be reused. This code reduction translated into an estimated eight-fold reduction of effort required to build new portals.
- Significant reduction of maintenance effort when enhancing individual portals. The overall managed code lines in Generic TCP supporting nine portals were 22% less than the original single portal.
- Wide range of portals differing in a large number of inter-dependent features supported by the Generic TCP.

Current status of XVCL

The XVCL Web site xvcl.comp.nus.edu.sg contains XVCL specifications, the processor, case studies and other learning materials. We have implemented an XVCL Workbench with the following tools:

Basic Workbench – supports developers in organizing XVCL project information, creating, maintaining and reusing XVCL representation. Basic Workbench includes Smart Editor that hides the XML syntax of XVCL commands, and allows developers to work in terms of XVCL structures rather than the text.

Debugger – interprets XVCL representation in the interactive mode, allowing a developer to trace the processing sequences and states. The processing can be suspended at designated break points and a developer can examine/change the state of processing.

Static/dynamic analyzer of XVCL representation – answers developer’s queries about the properties of XVCL representation and processing.

Visualizer – shows graphical views of XVCL representation.

X-Profiler – computes XVCL metrics such as meta-component reuse statistics, the number of times variation points have been modified during customization, and other statistics that help in reusing and evolving XVCL representations.

X-Profiler – computes XVCL metrics.

Backward Propagation Tool (BPT) – allows a developer to propagate code fixes from the generated source code back to meta-structures, in a semi-automatic way.

References

- [1] XVCL is fully documented in Jarzabek, S. *Effective Software Maintenance and Evolution: Reused-based Approach*, Auerbach, CRC Press Taylor and Francis, May 2007
- [2] Basit, H.A., Rajapakse, D.C., and Jarzabek, S. “[Beyond Templates: a Study of Clones in the STL and Some General Implications](#),” *Int. Conf. Software Engineering, ICSE’05*, St. Louis, USA, May 2005, pp. 451-459
- [3] Basit, A.H. and Jarzabek, S. “[Detecting Higher-level Similarity Patterns in Programs](#),” *ESEC-FSE’05, European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering*, ACM Press, September 2005, Lisbon, pp. 156-165
- [4] Basit, H. A., Jarzabek, S. “[Data Mining Approach for Detecting Higher-level Clones in Software](#),” *IEEE Trans. on Soft. Eng.*, July/August 2009 (vol. 35 no. 4) pp. 497-514; Published online January 2009
- [5] Bassett, P. *Framing software reuse - lessons from real world*, Yourdon Press, Prentice Hall, 1997
- [6] Clone Miner and Clone Analyzer: Technology Summary
- [7] Jarzabek, S. and Li, S. “[Eliminating Redundancies with a “Composition with Adaptation” Meta-programming Technique](#),” *Proc. ESEC-FSE’03, European Software Engineering Conf. and ACM SIGSOFT Symp. on the Foundations of Software Engineering*, ACM Press, September 2003, Helsinki, pp. 237-246; paper received ACM Distinguished Paper award; extended version: Jarzabek, S. and Li, S. ”Unifying clones with a generative programming technique: a case study,” *Journal of Software Maintenance and Evolution: Research and Practice* John Wiley & Sons, Volume 18, Issue 4, July/August 2006, pp. 267-292
- [8] Jarzabek, S. and Zhang, H. “[XML-based Method and Tool for Handling Variant Requirements in Domain Models](#)”, *Proc. 5th International Symposium on Requirements Engineering, RE’01*, August 2001, Toronto, Canada, pp. 166-173
- [9] Jarzabek, S. “Genericity - a “Missing in Action” Key to Software Simplification and Reuse,” *13th Asia-Pacific Soft. Eng. Conference, APSEC’06*, 6-8 December 2006, Bangalore, India, pp. 293-300
- [10] Pettersson, U., and Jarzabek, S. “[Industrial Experience with Building a Web Portal Product Line using a Lightweight, Reactive Approach](#),” *ESEC-FSE’05, European Software Engineering Conference and ACM SIGSOFT Symp. on the Foundations of Software Engineering*, ACM Press, Sept. 2005, Lisbon, pp. 326-335
- [11] Rajapakse, D. and Jarzabek, S. “Towards generic representation of web applications: solutions and trade-offs,” to appear in *Software, Practice & Experience*
- [12] Zhang, H. and Jarzabek, S., “[An XVCL-based Approach to Software Product Line Development](#)”, *Proc. 15th International Conference on Software Engineering and Knowledge Engineering (SEKE’03)*, San Francisco, USA, 1 - 3 July, 2003.
- [13] Zhang, W. and Jarzabek, S. “[Reuse without Compromising Performance: Experience from RPG Software Product Line for Mobile Devices](#),” *9th Int. Software Product Line Conference, SPLC’05*, September 2005, Rennes, France, pp. 57-69
- [14] Zhang, H., Jarzabek, S. and Myat Swe, S. “XVCL Approach to Separating Concerns in Product Line Assets,” *Proc. 3rd International Conference on Generative and Component-Based Software Engineering*, LNCS, Springer Verlag, September 2001, Erfurt, Germany, pp. 36-47
- [15] Zhang, H. and Jarzabek, S. [A Mechanism for Handling Variants in Software Product Lines](#),” special issue on Software Variability Management of Elsevier’s journal *Science of Computer Programming*, Volume 53, Issue 3, Dec. 2004, pp. 255-436
- [16] Yang, J. and Jarzabek, S. “[Applying a Generative Technique for Enhanced Reuse on J2EE Platform](#),” *4th Int. Conf. on Generative Programming and Component Engineering, GPCE’05*, Sep 29 - Oct 1, 2005, pp. 237-255
- [17] Yuan, L., Dong, J.S. and Basit, H. :Generic Fault Tolerant Software Architecture Reasoning and Customization,” *IEEE Trans. on Reliability*, vol. 55(3), 2006, pp. 421-435

Contact: Stan Jarzabek

Department of Computer Science, School of Computing, National University of Singapore
Computing 1, #03-68 Law Link, Singapore 117590

e-mail: stan@comp.nus.edu.sg; <http://www.comp.nus.edu.sg/~stan>

fax: 65-6779-4580; tel: 65-6874-2863 (office) 65-96255863 (mobile)