Bamboo: An Architecture Modeling and Code Generation Framework for Configuration Management Systems

Guozheng Ge, E. James Whitehead, Jr.
Dept. of Computer Science, University of California, Santa Cruz
{guozheng, ejw}@cs.ucsc.edu

ABSTRACT
We describe an architecture modeling and code generation framework called Bamboo. Using Bamboo, engineers design SCM repository and feature models, and then generate a running SCM system from the models.

Categories and Subject Descriptors
D.2.13 [**Reusable Software**]: Domain engineering, Reuse models; I.2.2 [**Automatic Programming**]: Program synthesis.

General Terms
Design, Languages.

Keywords
Configuration management, software modeling and generation.

1. INTRODUCTION
Software configuration Management (SCM) systems, such as CVS, ClearCase, and Subversion, manage the evolution of complex software systems [1]. With existing SCM systems, users have little control over feature customization and architecture evolution [2]. These systems have many features customers "possibly" need, not "exactly" need. Some SCM systems have open API libraries for tool integration, but using the API requires intricate interaction with the existing system architecture and the customization cost is high. According to IDC, there are more than 300 SCM systems, but we still see people building their own SCM tools from scratch to satisfy their particular requirements. We believe taking a model-driven design approach will help this situation. A uniform SCM architecture modeling and code generation framework can provide rapid feature customization at a lower cost. Our modeling framework, called Bamboo, opens up the possibility for architecture extensibility and evolution. Users can start from a small set of features and gradually add new features to avoid migration cost from one system to another.

The Bamboo framework allows SCM users to perform feature customization to include those features exactly required by users. An engineer using the framework first designs a repository model that includes major abstractions used in the system. Then, he follows a top-down approach to design the

SCM feature model and refine those features in terms of smaller functional units. Feature patterns that capture recurring SCM structures and behaviors are used in feature definition. Finally, the code generator consumes the model specifications and produces a running SCM system that reflects the feature design.

2. METHODOLOGY AND DESIGN
The SCM architecture modeling framework consists of two parts: Containment Modeling Framework (CMF) [3] and Semantic Modeling Framework (SMF). CMF includes a modeling language to specify data models and a code generator to automatically generate the repository. The containment modeling language is a variant of E-R modeling: *entities* represent data abstractions such as version, version history, branch; *relationships* describe connections and constraints between entities. The modeling language has two equivalent forms: the graphical form [3] is for human designers to draw the model in the GUI design tool; the equivalent XML form [4] is for the generator to produce the repository for the target system.

Semantic Modeling Framework (SMF) sits on top of the repository to design SCM features, logics and operations. SMF consists of three layers: feature structures, SCM utilities, and repository API. The feature structure layer defines the features an SCM system will exhibit. Each feature describes one aspect of an SCM system, such as version control, access control, concurrency control, and a feature can be further decomposed into smaller units called functions. For example, the version control feature can be decomposed into functions such as version history, compression, delta, version identification and composition, etc. Functions are designed using repository entities and services provided by the SCM utilities layer, including CRUD operations, network communication, auditing and logging, testing, etc. The repository API is exposed by the CMF to provide access to the SCM repository. The SCM semantic model specification consists of three parts: 1) the structure for each feature describes how the feature is built and with what functions/sub-functions; 2) feature interaction defines how features relate to each other through data, event exchange and constraints; and 3) semantic role mapping assigns SCM roles to the containment data model.

In SMF, we identify implementation patterns for each function so that users can simply make selections to compose a semantic model specification. Figure 1 shows an example of five different implementation patterns for the version history function. In a similar way, we have identified implementation patterns for other functions like access control, concurrency control, etc. Users can also define or edit function implementation patterns

using the containment modeling language provided by SMF if existing patterns don't match their requirements.

To define semantics and operations in function implementation patterns, we use "SCM roles", a predefined SCM entity type library. A role can have different attributes and operations depending on the selected function implementation pattern. For example, in the "Ordered Revision" version history pattern in Figure 1, *s_revision* role has the attributes such as *s_rev_identifier*, *s_rev_author*, *s_rev_comment*, etc., which are SCM roles themselves. It also has operations, such as retrieving attribute values, querying predecessor and successor revisions, delta computation, etc. But for the "Floating Object" version history pattern, *s_revision* has a unique *s_obj_identifier* for global query processing and it has additional query matching operations. By using SCM roles and function implementation patterns, we can precisely encode the knowledge captured during the SCM domain analysis process into the code generator.
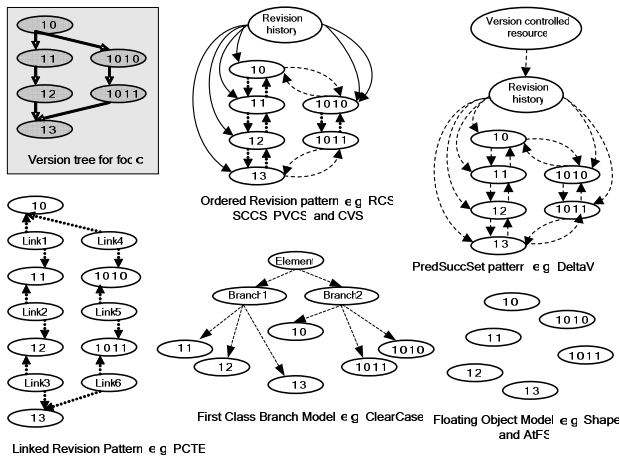

Figure 1. Version History Implementation Patterns.

SCM roles also bridge the gap between the repository layer and the semantic layer—we map entities in the containment model to roles in the semantic layer. Table 1 shows a simple example of mapping version control roles to entities in the containment model for the well-known RCS system (a partial list is presented because of the page limit).

Figure 2 shows the architecture for our code generator. Users design the containment and semantic models for a particular SCM system with the design tools. The design tool collects the XML representation of the model specifications and triggers the code generator. So far, we use template-based code generation: the generator replaces special markup tags in the template files with model-specific information. The output of the generator includes the source code for the repository, SCM features and operations, a command line client (for testing and demo purpose), and an Ant build file. Users build a running system from the generated source code and start using the generated system. To integrate the generated system with other tools, the command line client can be studied as a sample.

Table 1. A Sample Role Mapping between CMF and SMF.

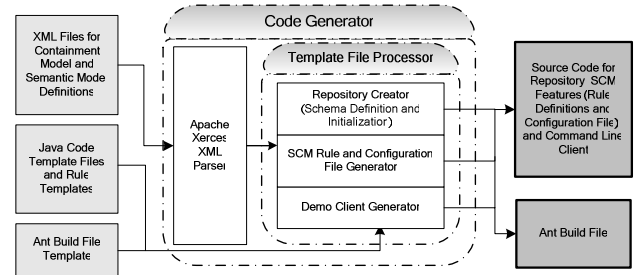| SCM Roles (SMF)	Meaning and Usage	RCS Entities (CMF)
s_revision	A container for versioning content and metadata items	revision
s_rev_identifier	Identifier for each version	version identifier
s_rev_author	Author for a revision	author
s_rev_comment	User comments submitted with each revision	comment

Figure 2. SCM System Generator Architecture.

3. RELATED WORK

The most relevant work to our SCM modeling framework is MCCM [5], which designs a pluggable infrastructure for composing SCM systems from fine-grained SCM policies/features. However, the policy composition and process still requires much manual work and the interactions between policies are not fully explored in MCCM. Clemm [6] proposes the WVCM API as the effort to unify workspace version control libraries. However, it has little industry support except from the Jakarta Slide project.

4. REFERENCES

[1] S. Dart, "Concepts in Configuration Management Systems." *SCM-3*, Trondheim, Norway, 1991.

[2] J. Estublier, D. Leblang, G. Clemm, R. Conradi, A. v. d. Hoek, W. Tichy, and D. Wilborg-Weber, "Impact of the Research Community on the Field of Software Configuration Management." *ACM SIGSOFT Software Engineering Notes*, vol. 27, no. 5, pp. 31-39, September 2002.

[3] E. J. Whitehead, Jr. and D. Gordon, "Uniform Comparison of Configuration Management Data Models." *SCM-11*, Portland, Oregon, 2003.

[4] G. Ge, "Automatic Generation of Rule-based Software Configuration Management Systems (a PhD proposal)." University of California, Santa Cruz November 2004.

[5] R. v. d. Lingen and A. v. d. Hoek, "An Experimental, Pluggable Infrastructure for Modular Configuration Management Policy Composition." *ICSE'04*, Edinburgh, United Kingdom, May 2004.

[6] G. Clemm, "JSR 147: Workspace Versioning and Configuration Management." 2003.