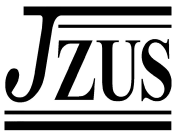
Journal of Zhejiang University SCIENCE
ISSN 1009-3095
http://www.zju.edu.cn/jzus
E-mail: jzus@zju.edu.cn



Component versioning for protocol configuration management*

CHEN Tian-zhou (陈天洲)†, HE Zhen-jie (贺臻杰), HUANG Jiang-wei (黄江伟), DAI Hong-jun (戴鸿君)

(*School of Computer Science, Zhejiang University, Hangzhou 310027, China*)

†E-mail: tzchen@zju.edu.cn

Received Oct. 26, 2004; revision accepted May 20, 2005

Abstract: Classical software configuration management which deals with source code versioning becomes insufficient when most components are distributed in binary form. As an important aspect of software configuration, protocol configuration also encounters those problems. This paper focuses on solving protocol component versioning issues for protocol configuration management on embedded system, incorporating the following versioning issues: version identification, version description and protocol component archiving and retrieving based on the version library.

Key words: Component-Based Software Engineering (CBSE), Protocol component, Protocol Configuration Management (PCM), versioning, meta-model,
doi:10.1631/jzus.2005.AS0075 **Document code:** A **CLC number:** TP393.04

INTRODUCTION

On embedded systems, protocols are frequently reduced or modified to act according to actual circumstances. Nevertheless, these applications must meet special requirement that traditional protocol stack cannot sufficiently support (Chen *et al*., 2004). Consequently, we need to reconstruct protocol stack to solve the problem.

Component-Based Software Engineering (CBSE) (Clements, 1995) has been adopted. Application of CBSE in development of embedded communication protocol yields protocol components that can make up protocol component library according to specific rules.

In the process of protocol component usage, component version issues occur. During the protocol assembly stage we need to track component binaries, and distribution packages or assemblies. During the Protocol Configuration Management (PCM) and dynamic upgrading we have to maintain related deployment descriptors, configuration and persistent state of the components in a particular protocol environment. Versioned entities change their nature during the different stages of a protocol development lifecycle.

To our knowledge, no suitable solution exists for dealing with protocol component in embedded systems. This paper discusses creation of protocol component version library based on protocol component library of embedded system, implementation of protocol component storage, retrieval and update through management of version library, enhancement of protocol component library and optimization of its performance.

RELATED WORK

The most popular tool used for software versioning, Revision Control System (RCS) and its network enabled variant, the Concurrent Version System (CVS), are based on source-code versioning (Alexander, 2005). There are also other file-oriented version models, which strongly reflect the requirements of a development process instead of being

*Project supported by the Hi-Tech Research and Development Program (863) of China (No. 2002AA1Z2306) and HP Embedded Laboratory of Zhejiang University, China

enabling mechanisms for process independent version management (Vijayan and Veda, 2003; Reidar, 1998). We evaluated the following systems with respect to versioning support: MS COM (and related ones like ActiveX and DCOM) and CORBA (Clemens, 2003; Rogerson, 1997).

None of these technologies provide a versioning model sufficient for a modern component based system. Their versioning models are tightly bound to the underlying technology and are not flexible enough for adapting to completely different component architecture.

We also evaluated popular distribution package managers like RedHat Package Manager (RPM), Debian Package Manager and abstract state-machine language (AsmL) versioning support in MS.NET (Clemens, 2003). These technologies feature some interesting concepts not present in many component versioning models. On the other hand, those systems have no notion of software components or data types, so they lack many features required for semantically rich version description.

COMPONENT AND PROTOCOL COMPONENT MANAGEMENT

Software components are the basic building blocks in component based development. A widely accepted definition of the term software component is that software components are defined as a coarse grained black-box software element with contractually specified syntax and semantics on both the provided and required side of the interface. In particular, a software component has to be a unit of deployment. Furthermore, to enable dynamic scenarios, it has to also be a unit of versioning and replacement (Clemens, 2003). We apply 3C model—concept, content and context, for general component modeling.

The concept of a component is a description of what the component does, providing abstract information about a component's functionality and associated semantics by using a faceted schema, which forms a facet descriptor using a set of predetermined facets and their values to specify component interface. The content describes how the concept is realized, forming a chunk of executable code providing a specific service for system implementation, or a diagram specialized structure, a function, or a state transition for system analysis and design. The context defines the contextual dependencies among components and specifies the "domain of applicability". We transfer the 3C model into a concrete scheme using faceted classification based on library and information science methods.

Protocol Configuration Management (PCM) defined as the discipline of controlling the evolution of complex protocols. PCM deals with development, assembly, configuration, updating and maintenance of protocols. PCM covers all the stages of protocol evolution. We focus on describing version management and related content in this paper.

The protocol component selection and substitution into the application architecture we call a protocol assembly process. In all the above cases, there is a need to provide efficient version management tools to help the application developer to manage this complex task.

We designed our versioning model as a flexible meta-model allowing us to instantiate a concrete versioning model for particular component model or software architecture. It can also serve as an abstraction layer for a conceptual integration of different component technologies. On the other hand we limit ourselves to the protocol component version archives and retrieval. We do not cover other aspects of an end-to-end versioning system like development process support (locking, triggers, scripting, etc.).

PROTOCOL COMPONENT VERSIONING FOR PCM

Versioning

Establishment of a new version model for PCM should meet some preconditions. The features that the version model must provide are: (1) The model allows defining protocol components, interfaces and high-level concepts as first-class entities; (2) The model provides unique identification of versioned entities in time and space; (3) The model supports revisions as well as variants for every versioned entity; (4) The model supports encapsulation in order to scale to large software systems; (5) The model provides powerful retrieval capabilities usable during PCM.

We present a versioning mechanism for PCM which can reasonably solve the version issue.

Versioning mechanism for PCM

A protocol component is assigned unique version identification. According to the terms defined above we can view the identification of a particular entity version as a two-step process: (1) We have to identify the individual protocol component; (2) Identify a particular version in a given version group. We are using the term Globally Unique Identification (GUID) for entity globally unique version identification.

Version attributes are properties describing a particular entity version. It may seem that the version attributes apply are applied to the version models using grid version space only.

We consider the grid-space concept as a generalization of graph-space (naming-convention) based version models and adopt facet classification defining each protocol component attribute as following format: Protocol component (guide, platform, format, provider, release, name).

Version relations are used to describe connections between versioned entities. A relation can be defined between entities (version groups), for example: ftpd requires libc versus ftpd2.5 requires libc5. The dependency relation (requires) is one of the most common relations the developers are dealing with.

(1) Relations versus attributes. Attributes can be considered as a special case of relations. E.g. an attribute platform="Linux" can be easily translated into requires relation.

(2) Attribute value taxonomies. Our versioning mechanism distinguishes two types of attributes: descriptive and taxonomic. The descriptive attributes correspond to generic entity attributes as used by other versioning models. The taxonomic attributes provide more information about their value domains: enumeration of legal values and taxonomy of the values.

Version library

Our solution is based on the version library concept. A version library contains version-related information for all versioned protocol components maintained in a protocol component library. The version library serves as a search engine which allows the user to query the library and obtain a set of GUIDs of a particular protocol component as a result. The GUIDs then can be used to archive protocol component into the protocol component library or retrieve them from it.

We use the concept of object oriented database to model our version library (Fig.1).

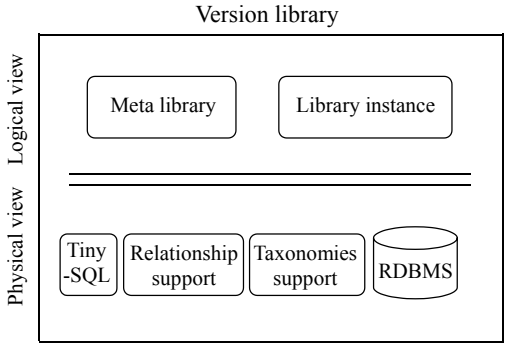


Fig.1 Version library

Tiny-SQL is a simulate language which we used as a symbol in the prototype version library implementation. We made this prototype to be our testing environment which enables us to test the version model described in the previous sections. Tiny-SQL is implemented on top of a relational database—it uses a relational database as the underlying data store. Tiny-SQL is more compact and points to PCM. We adopt Tiny-SQL to implement version archive and version retrieval. Tiny-SQL query language can be used to formulate advanced queries to the version library instance.

VERSION MANAGEMENT

Protocol component archive

In traditional ways, a new protocol component is entirely delivered to PCM when it is finished. In our mechanism, version is delivered in advance. Depending on the version library, search engine traverses the version library to find the version identification match with new arrival. It is allowed to archive into the library only if no match arises, consequently protocol component goes to protocol component and its version archives into version library respectively.

We use Tiny-SQL to check and archive a new protocol component.

Protocol component retrieval

The success of component retrieval depends on

how a version library and component library are managed. Fig.2 shows an outline of the protocol component retrieval process.

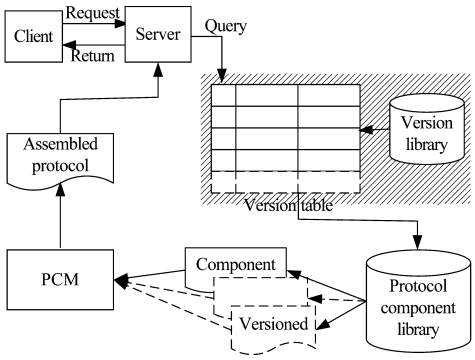


Fig.2 Protocol component retrieval process

Tiny-SQL allows the retrieval of a set of entities of a particular entity type defined in the version library, and can constrain a result set using selection criteria applied to attributes of a given object type.

EXPERIMENT

We update protocol component based on protocol component versioning mechanism for PCM, with component verification and transmission to test its performance with traditional updating method.

If the version already exists, then there is no need to archive it again, we call it a miss situation. We suppose the total number of protocol components is n (a fixed value), component coming in sequence, the i component size is s_i and its missing rate is m_i, let f_1, f_2 be the flow quantity of case 1 and case 2 respectively. Let s_v be pre-send version package size which is 1 kB in our version model. Now, we can get the flow quantity for each case, as Eq.(1) presentation.

$$\begin{cases} f_1 = \sum_{i=1}^{n} s_i \\ f_2 = ns_v + \sum_{i=1}^{n} m_i s_i \end{cases} \tag{1}$$

$$\begin{cases} f_1 / n = \sum_{i=1}^{n} s_i \Big/ n \\ f_2 / n = s_v + \left(\sum_{i=1}^{n} m_i s_i \right) \Big/ n \end{cases} \tag{2}$$

As n is fixed, we can change Eq.(1) to Eq.(2), because $0<m_i<1$, $\left(\sum_{i=1}^{n} m_i s_i \right) \Big/ n$ is always less than $\sum_{i=1}^{n} s_i \Big/ n$, the smaller the average value of $m_i s_i$ is, the larger is the difference between $\left(\sum_{i=1}^{n} m_i s_i \right) \Big/ n$ and $\sum_{i=1}^{n} s_i \Big/ n$. Value s_v can be ignored when n and s_i are big enough.

CONCLUSION

Our version model supports evolution of versioned entities (revisions) employing the concept of the user-defined attribute taxonomies and entity relations. The proposed version model also directly supports variants (parallel versions). We present a version library which facility version management and PCM. This feature reduces the overhead of versioning for the developer and allows the version model to scale to the enterprise level. For the prototype implementation we have developed a special query language (Tiny-SQL) which seamlessly incorporates the version model and allows for flexible and powerful version archive and retrieval.

References

Alexander, S., 2005. Component evolution and versioning state of the Art. *ACM SIGSOFT Software Engineering Notes*, **30**(1):7.

Chen, T.Z., Hu, W., Wu, Z.H., 2004. Parameterized Assembling Model of Communication Service. IEEE Conference on Computer, Communication, Control and Power Engineering Proceedings, Tencon.

Clements, P.C., 1995. From subroutines to subsystems: component based software development. *American Programmer*, **8**(11):3-6.

Clemens, S., 2003. Component Technology—What, Where, and How? Portland, Oregon, p.684.

Reidar, C., 1998. Version models for software configuration management. *ACM Computing Surveys*, **30**(2):232-282.

Rogerson, D., 1997. Inside COM. Microsoft Press.

Vijayan, S., Veda, C.S., 2003. A semantic-based approach to component retrieval. *The DATA BASE for Advances in Information Systems*, **34**:8-24.