Layered Object-Oriented Application Frameworks for Extensible CVS Proxy to Support Configuration Management Process

Kazuhiro Fujieda[*], Ryoh Hayasaka[**], and Koichiro Ochimizu[*]

September 26, 2003

IS-RR-2003-012

[*] School of Information Science

Japan Advanced Institute of Science and Technology (JAIST)

Asahidai 1-1, Tatsunokuchi Ishikawa, 923-1292 Japan

{fujieda,ochimizu}@jaist.ac.jp

[**] SOUMU Corporation

1-29-9 Hatagaya, Shibuya, Tokyo, Japan

ryoh@soumu.co.jp

Layered Object-Oriented Application Frameworks for Extensible CVS Proxy to Support Configuration Management Process

Kazuhiro Fujieda
Japan Advanced Institute of Science and Technology
1-1 Asahidai, Tatsunokuchi, Nomi, Ishikawa, Japan
fujieda@jaist.ac.jp

Ryoh Hayasaka
SOUM Corporation
1-29-9 Hatagaya, Shibuya, Tokyo, Japan
ryoh@soumu.co.jp

Koichiro Ochimizu
Japan Advanced Institute of Science and Technology
ochimizu@jaist.ac.jp

Abstract

CVS (Concurrent Versions System) is widely used in various software development projects. The reason is that it is simple to deploy and use, and little regulates the process of configuration management (CM), unlike rich-featured CM systems. As a consequence developers need to perform the CM process of their project by hand. We propose an extensible proxy server for CVS. It allows developers to add their own extensions supporting their process without any modification to CVS. We use object-oriented application frameworks to realize this extensibility. The architecture of the proxy consists of one library layer and two framework layers: the CVS adapter, the basic system framework, and extension frameworks. The basic system framework provides primitive and wide extensibility, and each extension framework provides domain-specific extensibility. Developers can implement extensions with low programming cost with appropriate use of each layer. We present an extension framework for access control mechanisms, and two access control mechanisms as the examples of extensions.

1. Introduction

Many software development projects have adopted configuration management (CM) systems to facilitate the implementation of CM. Existing CM systems offer different spectrums of functionality at different levels [3]. The process of CM in each project is different from others. The functional requirements to CM systems are also different in each project. It is hard for one single system to provide all the functionality required for all projects. Such systems as offer wide spectrum of the functionality consume too many human and computer resources for small projects to deploy and use them [2].

Open source software (OSS) projects have lightweight CM processes. Each project does not rarely define the workflow of its process, nor even keep the document about it. Whether explicit or not, they have many variations of the process. You can see some variations in [7] [9] [8]. Most OSS projects adopt CM systems supporting only version control. The representative of them is CVS [5]. CVS little regulates CM processes and supports distributed concurrent work via the Internet. The functionality regulating CM processes too much does not help their work. For example, UCM (Unified Change Management), the change control functionality in ClearCase [6], assigns one branch to one developer and isolates it from other developers. It cannot fit the branch usages in most OSS projects and even in other kinds of projects.

These projects don't necessarily need such functionality. Developers and managers in each project implement its process by manual procedures. In large-scale projects, the costs of the implementation become too large to be accepted. Actually, CVS provides a minimum extent of extensibility. It lets users specify certain programs invoked by CVS in specific events. Some projects use this extensibility to implement some facilities reducing the costs. But CVS provides only limited and fixed chances to invoke programs, and passes only limited information to these programs. It can invoke such programs only before the commit command or after successful executions of all command, and pass only the target module name of the command and a tag name, if the command is 'tag', to the programs. They cannot necessarily implement all facilities which they need.

To extend this extensibility, we provide an extensible proxy server for the CVS protocol. CVS use a client/server architecture with this protocol over TCP. A server main-

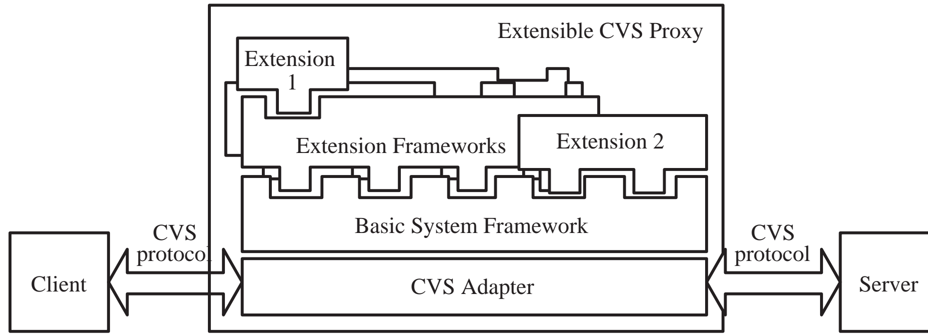

Figure 1. Overview of the architecture

tains a repository and takes almost of all processing in CVS. Clients simply send information necessary for the processing to the server. The proxy server is settled between the server and each client, and modifies requests and responses in the protocol according to the extensions integrated into it by users. These extensions can intercept any request or response so it can provide a wider range of extensibility than the previous way. Users can use their normal CVS client as long as extensions don't modify the protocol itself.

We design the proxy with the composition of one library layer and two layered object-oriented application frameworks [1] to provide the extensibility. The library layer named "CVS adapter" processes the binary representation of the CVS protocol. One framework layer named "basic system framework" provides normal behavior as a CVS proxy and primitive and wide extensibility. Another framework layer named "extension framework" provides extensibility to support specific domains of extensions. Users can implement necessary facilities and integrate them into the proxy at a low programming cost with the proper use of these layers.

This paper is organized as follows. Section 2 illustrates the architecture of the CVS proxy. Section 3 shows the details of the CVS adapter and the basic system framework. Section 4 shows the extension framework to implement various access control mechanisms, and two examples of extensions with this framework. Section 5 discusses related work. Finally, section 6 gives conclusions and future work.

2. Overview of the architecture

Figure 1 illustrates the architecture of the proxy. The following outlines three layers in this figure.

CVS adapter In the CVS protocol, a command executed by the CVS client, such as 'checkin', 'update', 'checkout' or else, consists of many requests and responses. The input and output streams between a server and a client can be compressed. The CVS adapter handles these streams, cut each request or response from them, and then convert it to an event object with its name and all arguments.

Basic System Framework The basic system framework setups the CVS adapter and behaves as a CVS proxy. It defines the flow of control in the proxy, and how to dispatch each event object to the corresponding handler. The handler corresponds the extension 2 and the extension frameworks in Figure 1. These handlers can modify the event object and the flow.

Extension Framework It requires the knowledge of the details about the protocol to define the handlers. A meaningful extension requires not only a handler class but other additional classes. It takes the high programming cost to define these handlers and additional classes for every extension from scratch. The extension frameworks are prepared for reducing the cost to implement the same expected kinds of extensions.

The following sections illustrate the detail designs of these layers.

3. CVS protocol layer and basic system framework

Figure 2 shows the class diagram of the CVS adapter and the basic system framework. The left side of it shows the former and the right side shows the latter. Notice that we omitted the elements to handle errors and construct objects from the figure and will little mention them in the following because of simplification.

3.1. CVS Adapter

CvsIO cuts one line or one transmission of one file in the streams. There are two instances of this class. One han-

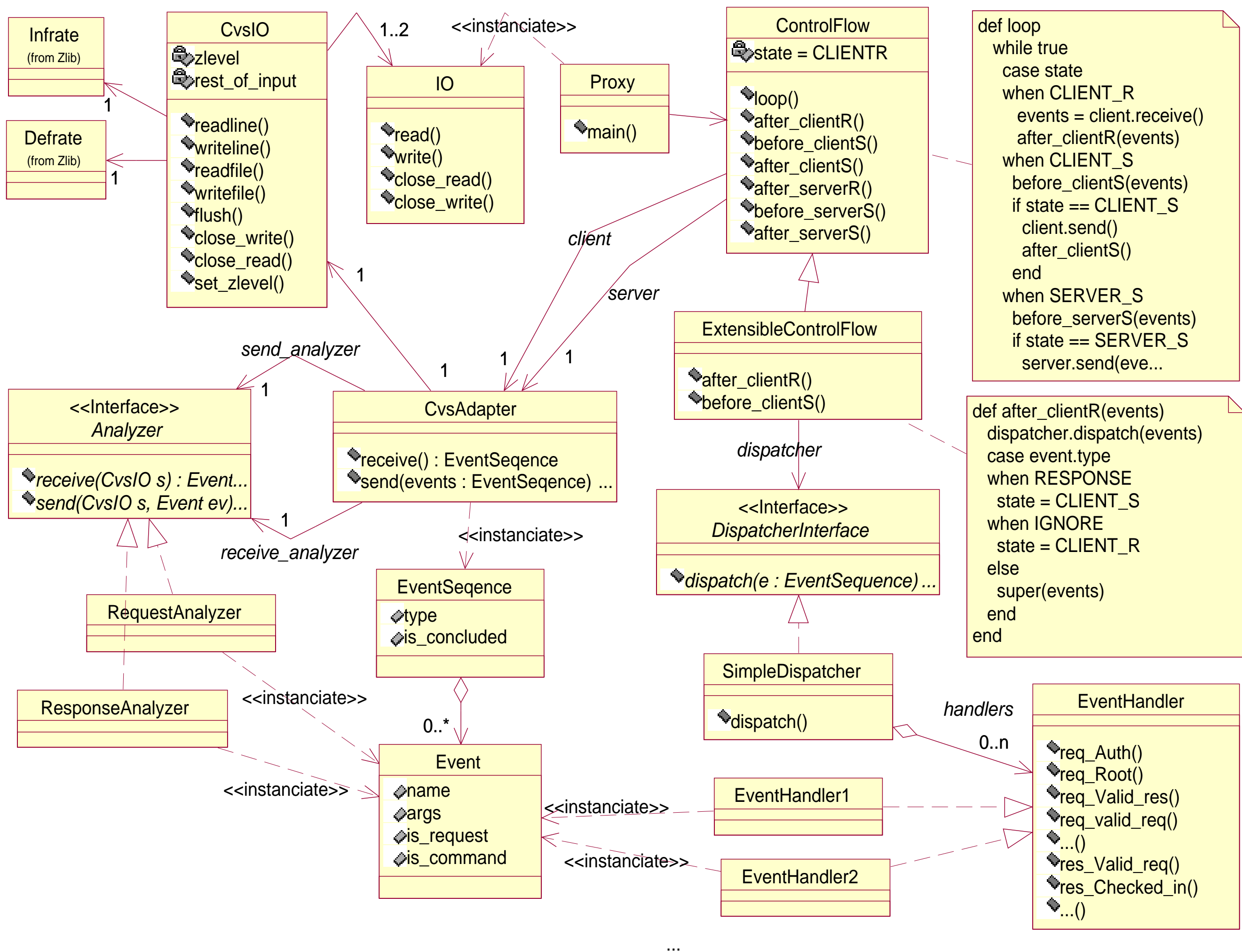

Figure 2. Class diagram of CVS adapter and basic system framework

dles the stream for a client, and another handles the stream for a server. In the CVS protocol, each request or response consists of one line or a few lines. Some of them also follows one file transmission. The RequestAnalyzer knows the structure of each request, and the ResponseAnalyzer knows one of each response. These analyzers return an instance of Event representing each request or response. They also flush the stream or set the compression level according to the protocol.

Two instances of CvsAdapter are constructed by the basic system framework so each instance has proper analyzers and one CvsIO object. Each instance corresponds to either a client or a server. CvsAdapter does not know anything about the protocol, but handles sequences of Event objects. It does not have to handle such sequences if the basic system framework behaves as a simple proxy. But an extensible proxy may pass multiple requests to a server against one request from a client. The operations in this class handle this case.

3.2. Basic system framework

The basic system framework defines the control flow of the proxy, how each event is dispatched to the corresponding handlers, how to define the handler.

3.2.1 Control flow of the proxy

In the CVS protocol, a client sends many requests not requiring any response, concludes with one request requiring responses called ‘command’, and then wait them. Almost all the command requests correspond the actual CVS commands. A server sends many responses against the request, and concludes with the either ‘error’ or ‘ok’ response, and then waits requests. This flow of control is shown in the

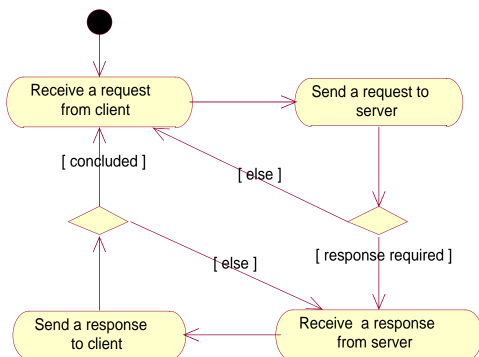

Figure 3. Default flow of control

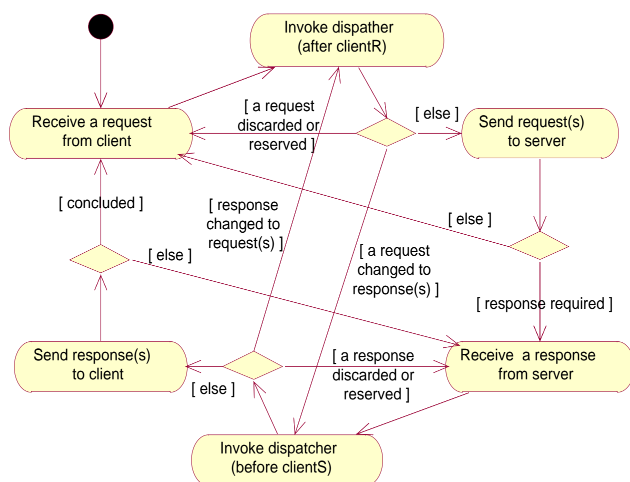

Figure 4. Extensible flow of control

Figure 3 ControlFlow defines it and allow its subclass to intercept and change the flow at some timings.

ExtensibleControlFlow defines another flow of control and provides chances to change the flow to the extensions. This flow of control is shown in Figure 4. It puts the following assumptions on the behavior of the extensions. Each extension may change one request from the client into another request or a sequence of requests to the server. It may also discard or keep the request so it isn't sent to the server. Moreover, it may change it into responses to the client. It may change one response from the server likewise.

3.2.2 How to dispatch events to handlers

ExtensibleControlFlow uses one instance of a concrete class realizing DispatherInterface to dispatch each event to the corresponding handlers. The basic system framework provides SimpleDispather as such class. You may, however, define another class and let the ExtensibleControlFlow use its instance.

A SimpleDispather object (a dispatcher) has some EventHandler objects (handlers). The dispatcher invokes an operation of each of the handlers on an event in the order where they were registered. If a handler changes the event into another, it invokes the rest of the handlers on another. If a handler consumes the event, it does not invoke the rests. If a handler changes the event into a sequence of events, it invokes the rests just like these events are fed separately.

Rationale In the CVS protocol, An actual command forms a sequence of many requests containing its arguments and a concluding command request. Handlers cannot know how the arguments are used until it receives the command request. If a handler wants to change the arguments according to the command, it must consume all request events until receiving the command, and then return the resulting sequence of events. Each request sent to the proxy as an argument may include a whole contents of a file, so these handlers also consumes the resources of the proxy server. If a handler does not want to change anything but cause side effects or only change the result of the command, it can pass the requests to the server. The dispatcher is so designed as to manage such two types of handlers.

3.2.3 How to define handlers

Each extension to the proxy is implemented as a subclass of EventHandler. EventHandler defines operations for all request and response events. These operations do not anything by default. It is necessary to define all operations in one class because some extensions may have to track the protocol and record its state. If these operations were separated into several classes, the instances of those subclasses might have to share an instance recording the state.

4. Extension framework

Although CvsAdapter manages the byte representation of the CVS protocol, it requires a deep knowledge of the protocol to implement an extension with the basic system framework. It is necessary to prepare some frameworks for making it easier to implement the same kinds of extensions. We call them 'extension frameworks'. They are implemented as extensions to the basic system framework.

CVS only supports the access control to specify who can write or not the whole repository. More sophisticated control depends on the underlying operating system. It needs access control at least better than CVS to perform change control properly, so we choose the framework to implement various type of access control as an example of extension frameworks. We will show the following two extensions implemented with the framework: ACL (Access Control

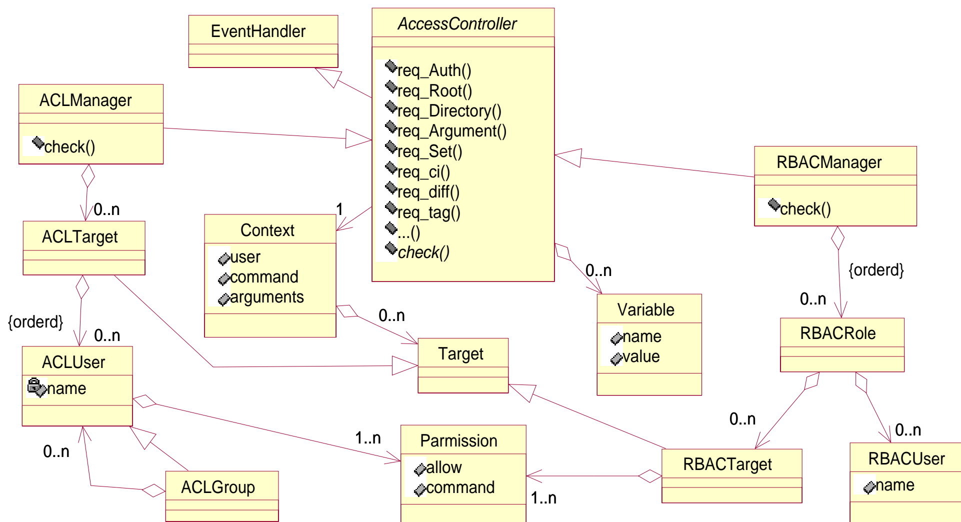

Figure 5. Class diagram of an extension framework and two extensions

List) independent of underlying OS and RBAC (Role Based Access Control). The class diagram of the framework and these extensions is shown in Figure 5

4.1. Extension framework for access control

Access control generally determines which subject can or cannot have which type of access to which object. The models of access control are roughly divided into either Access Control List (ACL) or capability list. The former defines the policies on each object and the latter define them on each subject. In either model, the policies are described about subjects, objects, and types of access. AccessController in Figure 5 overrides several methods of EventHandler to extract this triple from a sequence of requests flowing in the basic system framework and store it into a Context object. When a user connects the proxy with rsh or ssh without the password authentication in the CVS protocol, AccessController gets the user name with the process ID of the proxy because it cannot know the user name via the protocol.

AccessController passes the requests to the server until receiving a command request. It invokes the *check* operation for each command request to let its subclass determine whether the command is acceptable or not. If it is acceptable, AccessController return it to the basic system framework. Otherwise AccessController changes it to the 'error' response. When the client accepts this response, it closes the connection to the proxy. The proxy also closes the connection to the server in its error handling, so the requests sent previously are discarded by the server.

The Variable class holds user variables. Users can set a user variable by specifying the option '-s' to the cvs command with an argument 'VARIABLE=VALUE'. This argument is sent to the server with the 'Set' request. This option is originally prepared to pass some variables to the programs invoked by CVS. It is also useful for passing some variables to the extensions. AccessController intercepts the 'Set' requests and records these arguments in Variable objects.

4.2. ACL (Access Control List)

ACL defines policies as a list of pairs of a subject and an access type for each object. ACLUser and ACLGroup, ACLTarget, and Permission in Figure 5 are to represent subjects, objects, and access types respectively. ACLManager constructs these instances to represent the policies according to some configuration files when the proxy start up or these files are modified. ACLManager override the check operation to determine whether a command is acceptable or not according to the policies and the Context object. This operation iterates checking policies for each target, When both of the user and the command in the Context object matches a policy, the operation returns the value of the 'allow' attribute in the Permission object corresponding to the policy.

4.3. RBAC (Role Based Access Control)

RBAC is a kind of capability list and defines the policies for each role assigned to subjects [4]. The relationships of elements in Core RBAC are shown in Figure 6. Users are assigned to roles, permissions are assigned to roles, and

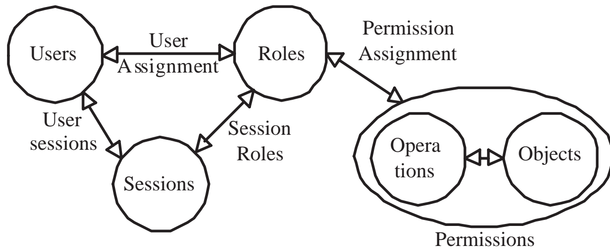

Figure 6. Relationships between elements in RBAC

users acquire permissions by being members of roles. Core RBAC includes the concept of user sessions, which allows selective activation of roles. Hierarchical RBAC also supports role hierarchies, which define an inheritance relation among roles. The concept of role hierarchies is an essence of RBAC, but we do not support it to avoid its complexity. The concept of session roles is attractive to support the change control in some OSS projects.

These OSS projects often give the write permission to their repository to contributors rather easily. These contributors are called 'committers' named after the commit command of CVS. They can write the repository although they have to satisfy some requirements. The Committer's Guide of the FreeBSD project [8] requires committers to discuss any significant change with other committers before committing. It also requires that changes against an area owned by a 'maintainer' are reviewed by him/her before committing. But these are merely verbal rules. From the point of view of CVS, all of committers and maintainers have the write permission to the whole repository. Although the committers respect these rules, CVS can't prevent their careless mistakes. We believe the concept of session roles can prevent them. Each committer can usually specify his/her session role as an ordinary contributor, and can specify the session role as a committer when he/she really commits his/her changes.

RBACUser, RBACRole, Permission, and RBACTarget in Figure 5 correspond Users, Roles, Operations, and Objects in Figure 6 respectively. Roles are not hierarchical but ordered to make the first role assigned to the user of the default session role. The notion of the default session role is defined in RBAC. We, however, introduce it for user's convenience. Users can specify their session roles via the '-s' option mentioned above. The RBAC extension uses the 'ROLE' variable to specify the session roles. Users can invoke the command like '`cvs -s ROLE=committer commit`'. When users want to specify multiple roles, they can list the roles separated with a semicolon.

5. Related work

5.1. NUCM

NUCM [10] provides a generic repository model and a programmatic interface of it to implement new CM systems on the generic model. This approach tackles the following problem similar to one in our approach.

> ... the basic functionality provided by a given CM system is fixed; if specialized functionality is needed in a particular situation ...

A particular situation is specific to a CM system in this approach, while the situation is specific to a project in our approach. NUCM provides only low-level generic model not specifying even any version model and concrete CM procedures to let us implement new models and procedures. When a project implements a CM facility specific to it with NUCM, it has to implements a CM system itself at the first place. The programming costs to do it do not pay. Our approach provides extensibility to an existing CM system, so it does not take lower programming costs to implement a project specific facility than NUCM.

5.2. Visual SourceSafe

Visual SourceSafe (VSS) allow users to hook into and control various events within it [11]. This is accomplished through the creation of a VSS add-in and the registration of it as a add-in. Most actions accessing a repository, which called a database in CVS, trigger corresponding events. The add-in can trap these events and prevent them from occurring before they actually occur, and trap them after they occurred. VSS provides well designed interfaces to access the database. The add-in can use them to read and write it. Our approach does not allow extensions to access the repository because it can break the consistency of the behavior as a CVS proxy. But we consider most extensions get necessary information through the CVS protocol.

VSS clients share the database through the file sharing in Windows and carry out all functions of VSS. The extensibility with the add-ins mentioned above is provided by the clients. When a project implements the facilities to support its CM process with this extensibility, the members of the project must share the same add-ins. This architecture helps the add-ins to interact with both of users and the database. It, however, is not feasible for distributed projects via the Internet, because it is difficult for all members of each project to share the exactly same add-in. Our approach focuses it and can enforce the facilities on all members accessing the repository via the proxy.

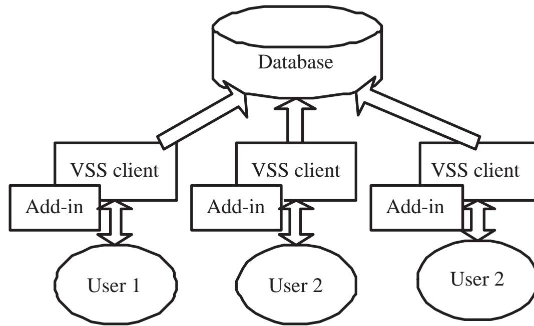

Figure 7. Architecture of Visual SourceSafe

6. Conclusions

In this paper, we have discussed the architecture of the extensible CVS proxy. This proxy allows a project to integrate facilities necessary for its CM process into itself. It has layered object-oriented frameworks of the basic system framework and the extension frameworks. The former defines the default flow control of the proxy and provides its extensibility. It defines a customized control flow to dispatch the events of requests and responses in the CVS protocol to corresponding handlers. It also defines how events are dispatched to the handlers and how the handlers are defined. We presented an example of the latter to implement access control mechanisms, and it can help to implement two extensions of ACL and RBAC. We showed Core RBAC including session roles could be implemented as an extension to the proxy.

It is rather easy to design the extension framework for access control mechanisms, because they has been researched and consolidated since long ago. We suppose we can provide extension frameworks for only limited kinds of extensions, that is, the same kinds of simple extensions, or ones of which models are fully consolidated. For the former kinds of extensions, we suppose the extensions of prescribing naming conventions of files or tags, or writing styles of logs on committed changes, or coding styles. As for other kinds of complex extensions, It may be necessary to implement them from scratch or on the basis of other similar extensions or extension frameworks.

Object-oriented frameworks are generally sophisticated through iterative reuse in various applications. The frameworks shown in this paper should be redesigned through implementing various extensions and extension frameworks. We will implement some extension frameworks for the simple extensions mentioned above, and also implement extensions to support explicitly change control specific to each project as mentioned in the Committer's Guide.

References

[1] D. Bäumer, G. Gryczan, R. Knoll, C. Linienthal, D. Riehle, and H. Züllighoven. Framework development for large systems. *Communications of the ACM*, 40(10):53–59, Oct. 1997.

[2] I. Crnkovic. Why do some mature organizations not use mature CM tools? In *Proceedings of 9th International Symposium on System Configuration Management (SCM-9)*, pages 50–65, 1999.

[3] S. Dart. Concepts in configuration magement systems. In *Proceedings of Third International Workshop on Software Configuration Management (SCM-3)*, pages 1–18, 1991.

[4] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramoli. Proposed NIST standard for role-based access control. *ACM Transactions on Information and System Security*, 4(3):224–274, 2001.

[5] K. Fogel. *Open Source Development with CVS*. CoriolisOpen Press, 1999.

[6] Rational Software Corporation. *Unified Change Management from Rational Software: An Activity-Based Process for Managing Change*. Rational Software Whitepaper TP710A, 02/02.

[7] The Apache Software Foundation. Apache HTTP server developer information. `http://httpd.apache.org/dev/`, 2002.

[8] The FreeBSD Documentation Project. Committer guide. `http://www.freebsd.org/doc/en_US.ISO8859-1/articles/committers-guide/`, Feb 2003.

[9] The Mozilla Organization. Mozilla hacking in a nutshell. `http://www.mozilla.org/hacking/`, Feb 2003.

[10] A. van der Hoek, A. Carzaniga, D. Heimbgner, and A. Wolf. A testbed for configuration management policy programming. *IEEE Transactions on Software Engineering*, 28(1):79–99, Jan 2002.

[11] T. Winter. Visual sourcesafe 6.0 automation. `http://msdn.microsoft.com/ssafe/technical/articles.asp`, Sep 1998.