ELSEVIER

Future Generation Computer Systems 17 (2001) 1039–1049

FGCS
FUTURE GENERATION COMPUTER SYSTEMS

www.elsevier.nl/locate/future

Managing and sharing collaborative files through WWW

Ruey-Kai Sheu [a], Yue-Shan Chang [b,*], Shyan-Ming Yuan [a]

[a] *Department of Computer and Information Science, National Chiao-Tung University, Hsin-Chu, Taiwan, ROC*
[b] *Department of Electronic Engineering, Ming Hsing Institute of Technology, 1 Hsin-Hsing Road, Hsin-Fong, Hsin-Chu 304, Taiwan, ROC*

Abstract

The increasing complexity and geographical separation of design data, tools and teams have demanded for a collaborative and distributed management environment. In this paper, we present a practical system, the CFMS, which is designed to manage collaborative files on WWW. Wherever remote developers are, they can navigate the evolutions and relationships between target files, check in and check out them conveniently on web browsers using CFMS. The capabilities of CFMS include the two-level navigation mechanism for version selection, the relationship model that conceptually manages the relationships between physical files as well as the prefix-based naming scheme that can uniquely identify requested objects.

Keywords: Collaborative file management; Prefix-based naming scheme; Relationship model

1. Introduction

The increasing complexity and geographical separation of design data, tools and teams have demanded for a collaborative and distributed management environment. That is, companies need good management tools to leverage the skills of manipulating the most talented and experienced resources, wherever in the company or the somewhere they are located.

Major challenges for collaborative file management include version control, configuration management, concurrency access control, environment heterogeneity, and the dispersion of developers [1]. Take the electronic design automation (EDA) paradigm, e.g., design teams realize that the number of tools needed to implement complex systems is ever increasing, and such tools are generally provided by many different suppliers [2]. At the same time, developers at different sites would parallelize the developments with many variants of each component, which might contain and share many files and many versions of each file. When controlling versions, developers have to deal with versions and configurations that are organized by files and directories. This is inconvenient and error-prone, since there is a gap between dealing with source codes and managing configurations. It is necessary for engineers to be capable of identifying the items which they are developing or maintaining.

In this paper, we concentrate on issues of developing a web-based and visualized collaborative file management system. The web-based design has advantages of eliminating the need to install and administrate client softwares, resolving the cross-platform problems, providing the accessibility for distributed developers, and having a uniform and compatible interface for different users. Not only does the proposed CFMS have all the advantages of web-based environments, but also captures and manages change requests among team

* Corresponding author. Tel.: +886-3-557-2930; fax: +886-3-559-1402.
E-mail addresses: rksheu@cis.nctu.edu.tw (R.-K. Sheu), ysc@mhit.edu.tw (Y.-S. Chang), smyuan@cis.nctu.edu.tw (S.-M. Yuan).

0167-739X/01/$ – see front matter
PII: S0167-739X(01)00045-0

members transparently. Through the proposed two-level navigation version selection mechanism, CFMS users can define and choose the configuration of the developed component dynamically and visually. Without directly operating the bothersome chaos of physical files, users only need to manipulate the conceptual relationships between requested files on web.

The rest of the paper is organized as follows. In Section 2, we discuss the related works about collaboration management tools. In Section 3, CFMS system models are detailed. Section 4 briefly introduces the CFMS system architecture. Section 5 shows the implementation of CFMS. Finally, the conclusion is given in Section 6.

2. Related works

There are several collaboration management tools in many paradigms. To illustrate, RCS [3], SCCS [4], DSEE [5], and ClearCase [6] concern the activities for cooperative system development. RCS and SCCS are the most widely known version management system. These systems are built on the notion of vault [7] from where users must extract all of the sources, even you do not plan to change them. Besides, parallel development will result in even more copies. These copies are not under the control of configuration systems and may lead to more problems because they do not support concurrency control facilities. There are several later tools, such as CVS [8], built on top of RCS. These tools improve the management of private work areas, but do not really solve the fundamental problem inherent in vaults.

DSEE and ClearCase are tools of the second-generation configuration management environment, which advance upon earlier tools for defining and building configurations. They solve the problems inherent in vaults through the use of a virtual file system. Intercepting native operating system I/O calls, they hook into the regular file system at user-defined points by user-specified rules. Intercepting system calls will tight the private work areas with the central data center together and introduce problems encountered in traditional systems. Additionally, the rule-based configuration mechanism is too complex to be used for web users.

To solve problems of old systems we design the CFMS as a moderate three-tier architecture. A middle-tier is introduced to decouple the front-end users and the back-end file system, and simplify the system complexity. In CFMS, a conceptual relation model is proposed. Instead of the text-based or rule-based methodologies used in traditional tools, front-end users could traverse the relationship graph to define or navigate configurations The web-based CFMS will promote the collaboration management paradigm into the distributed heterogeneous environment.

3. System model

Three essential system models which compose of the three-tier model, the relationship model and the prefix-based naming scheme for collaborative file management are clearly identified in this section. These three models are orthogonal and are integrated to form the building blocks of the CFMS.

3.1. Three-tier model

In comparison with the vault-based and virtual file design of traditional collaboration management tools, the three-tier model looses the tight-couple relation between the client side representations of requested objects and the back-end file systems. In the front-end, users of different sites could define configurations for their private workspaces and use SCCS-like check in/out operations to access the requested files. In the middle-tier, the relationships between all the managed files are constructed and presented by a bi-direction graph. Front-end users can surf back-end files by traveling the relationship graph. The third-tier is the vast file system, which consists of many types of files generated by different tools. Each physical file could be uniquely identified and conceptually versioned in the second-tier. That is, the second-tier hides the complexity of directly manipulating the physical files and provides higher flexibility and extensibility for CFMS. Fig. 1 shows the three-tier architecture of the proposed web-based collaborative file management system. Front-end browsers show the user-selected configuration graphs, which might share nodes in the central relationship graph. The middle-tier hides the complexity to manage the relations for the third-tier flat files.

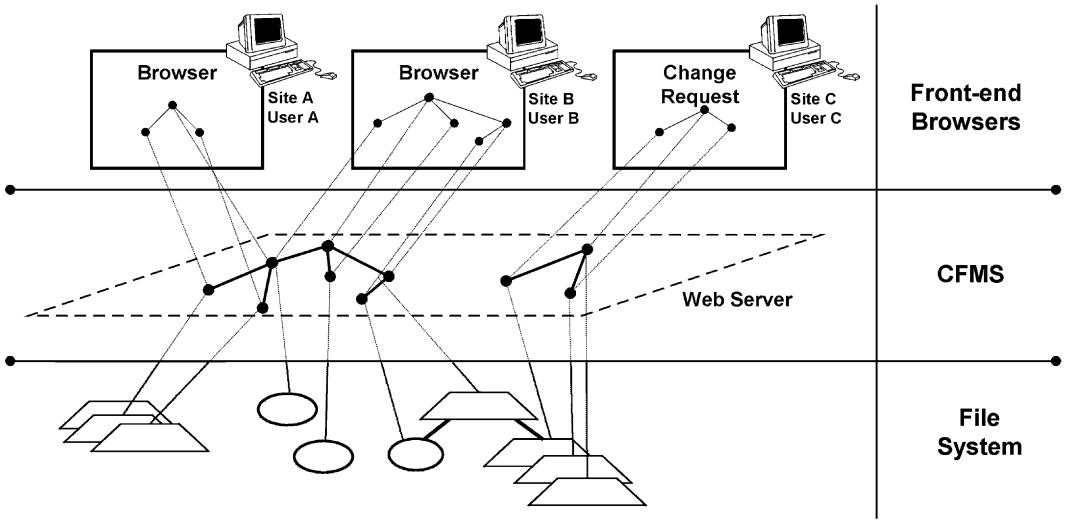

Fig. 1. Three-tier architecture of CFMS.

3.2. Relationship model

The CFMS uses logic structures to manage files, which are organized into a three-level hierarchical architecture and they are projects, libraries and files from the highest level down to the lowest one. A project is the achievement of a specific objective. It involves several libraries as well as stand-alone files. A library is a frequently used group of files. The relationship model identifies the correlations between managed logic structures and classifies them into six categories: version, build, configuration, equivalence, hyper-link and sibling. The main contribution of this paper is to identify and clarify these concepts into orthogonal, rather than sequential relationships.

Version. Version relationships are composed of branches and derived items, and can be represented by two-dimensional graphs. In Fig. 2, the generic version graph is described. Object A.1 is the root of the version history for the logic structure A. Object A.1.2 is a new version of A.1 after the creation time of A.1.1. The relationship between A.1 and A.1.2 is the branch relation. Other edges in the version history are derived relations. The logic structure A could be a file, a library or a project.

Build. Build relation stands for the aggregation of a specific logic structure. Fig. 3 shows the version graph as well as the build relation of the logic structure A. The logic structure A is composed of two logic structures X and Y. Both X and Y could be any logic structures that are lower or equal to the level of A in the hierarchy of logic structures.

Equivalence. Equivalence relationships are different representations of a logic structure in different development stages or platforms. For example, a C++ source program could be compiled as a

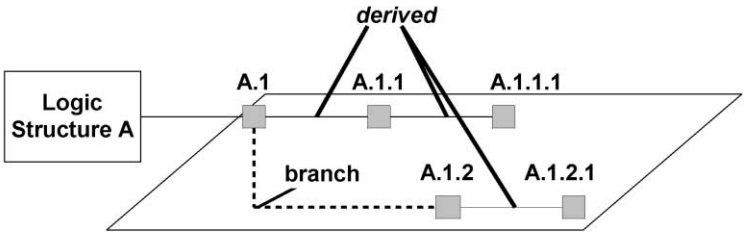

Fig. 2. The generic version graph of a logic structure.

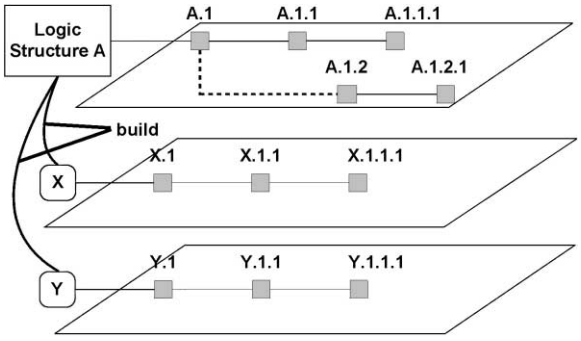

Fig. 3. The build relation of the logic structure A.

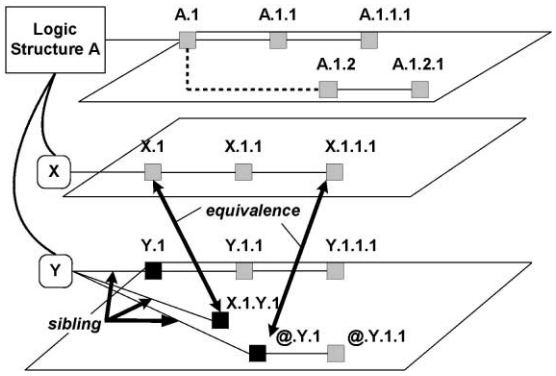

Fig. 4. The equivalence and sibling relation of the logic structure A.

PC-based or workstation-based object files. These files are of the same meaning logically, but of different representations in the real world. In Fig. 4, we assume that X.1.Y.1 is created from X.1 and @.Y.1 is created from X.1.1.1 by tools which take X-versioned objects as inputs and generate outputs objects of the same type as Y. Here, edge (X.1, X.1.Y.1) and edge (X.1.1.1, @.Y.1) are two equivalence relations. Where the '@' notation is used to represent the prefix discussed in next section.

Sibling. Siblings are relationships between parallel-developed versions of a logic structure. In other words, siblings are parallel version history paths of a logic structure. Object Y.1, X.1.Y.1 and @.Y.1 are siblings of Y in Fig. 4. When someone creates an new item of Y type from X.1.1.1, @.Y.1 (@ = X.1.1.1) is the most adequate identity of version name than others. Neither Y.1.1.1 nor X.1.Y.1 is the parent of @.Y.1 for the derived relation. If we connect the edge (X.1.Y.1, @.Y.1) or (Y.1.1.1, @.Y.1) with relations rather than sibling, users will be confused with them.

Configuration. Configuration relationships consist of snapshots of higher-level logic structures. Fig. 5 illustrates the snapshots of the logic structure A. Object A.1 consists of X.1 and Y.1. Object A.1.1 is composed of X.1.1 and Y.1. Object A.1.2.1 involves X.1.1.1 and X.1.Y.1.

Hyper-link. Hyper-link relation tights the manipulated files as well as the documentations that describe the related information of them. In Fig. 5, we shows the examples of hyper-link relations that record the

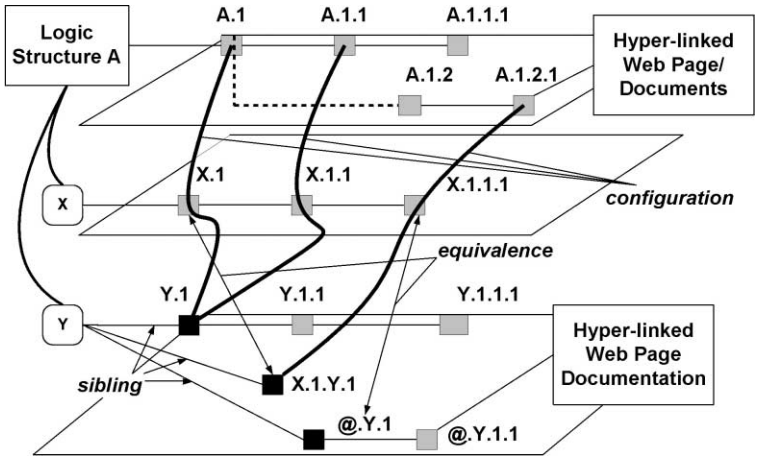

Fig. 5. The configuration and hyper-link relation of the logic structure A.

hyperlinks to web pages describing related information, including why they are improved, when the object is updated, who did the changes, where did the author locate, how it is operated, etc.

3.3. Prefix-based naming scheme

Engineers must be able to identify the items which they are developing or maintaining. In particular, each item must have a unique name and the name must be meaningful. In CFMS, the proposed prefix-based name has the meaning of the evolution of the item, the logic structure type and name, and the corresponding development stages. The prefix-based name is primarily used to help user to traverse and navigate the multi-dimensional relation graph. It can also be used for text-based query for version selection based on logic structure names.

The followings are the productions for version name resolution, where ⟨⟩ stands for non-terminals and others are terminals. Version names can be abstracted as a prefix concatenating with the relationship name. To be specific, the version name of a new item for the derived relation will be the prefix, the original old version name, concatenating with a derived name. The prefix-based naming scheme gives each relationship a unique name, respectively. Because the prefix-based naming scheme gives a unique identity for each orthogonal relation, the version name for each object will guarantee uniqueness even versioned objects have the same prefix.

The BNF of the prefix-based naming scheme:

⟨**version name**⟩	→	⟨prefix⟩.⟨relation⟩
⟨relation⟩	→	⟨equivalence⟩|⟨derived⟩|⟨branch⟩
⟨branch⟩	→	.(number of branches + 1)
⟨derived⟩	→	.1|⟨derived⟩
⟨equivalence⟩	→	⟨prefix⟩.⟨base⟩.(number of siblings + 1)
⟨prefix⟩	→	⟨base⟩.⟨derived⟩|⟨base⟩.⟨branch⟩|⟨version name⟩
⟨base⟩	→	⟨logic name⟩|⟨logic name⟩.1
⟨logic name⟩	→	P_⟨name⟩|L_⟨name⟩|F_⟨name⟩|F_⟨name⟩_⟨stage⟩
⟨stage⟩	→	(number of development steps+ number of platforms)
⟨name⟩	→	string of character set excluding the dot

4. System architecture

The CFMS is a three-tier architecture, as shown in Fig. 6. The front-end user interfaces are WWW browsers. Through HTTP, users download the Java Applet into the client sites. The Java Applet is responsible for drawing the partial relationship graph based on configuration for users. The reason why Java Applet is used here is that current SGML-based standard markup languages are not suitable to draw the complex multi-dimensional relationship graph. In the middle-tier, the CFMS is composed of relationship manager, query manager, concurrency control manager and the resource manager. The relationship manager administrates the relations between collaborative files. The query manager provides facilities for text-based query. Users can submit search terms to select the logic structures that satisfy the search criteria. Concurrency control manager is responsible for the management of parallel developments on target files. CFMS supplies the SCCS-like check in/out operations to access files. It allows multiple read requests for a specific file at a given time. Only the user who gets the write lock can update that file. Resource manager controls all the physical files through the physical file I/O system calls.

5. System implementation

In this section, we discuss the most significant portions, which represent the major prominent discrepancies between CFMS and other collaboration management tools.

5.1. Logic structure

Logic structure is the main idea to manage the logic relationships of files. In CFMS, a database is used to keep the mapping between relation graph and physical files, which could be distributed in different locations. That is, isolating the logic relations from the vault also provides the potential for our further development of CFMS to a distributed management fashion instead of a centralized one.

In CFMS, the build and configuration relations are kept in container objects. A container is composed of tree objects. Each tree comprises many node objects.

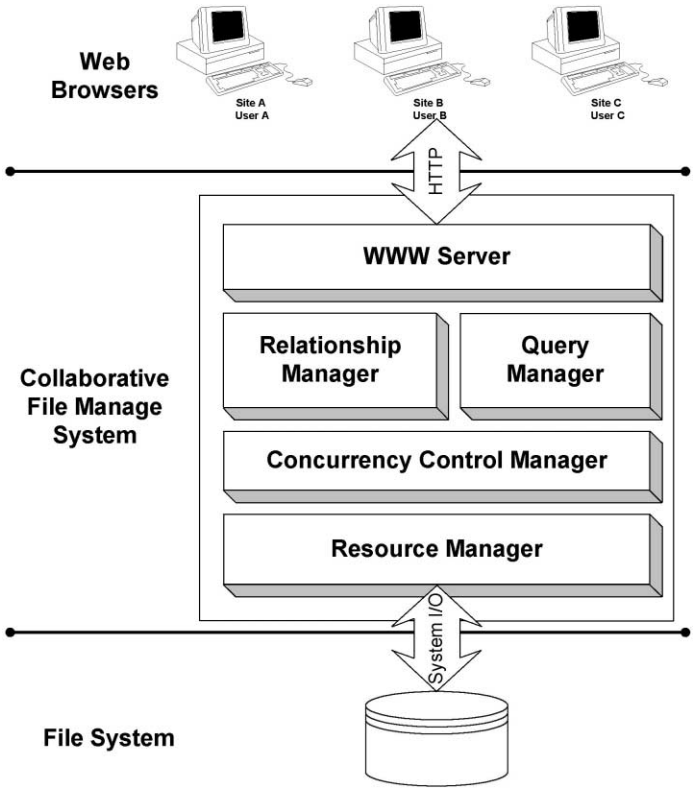

Fig. 6. CFMS system architecture.

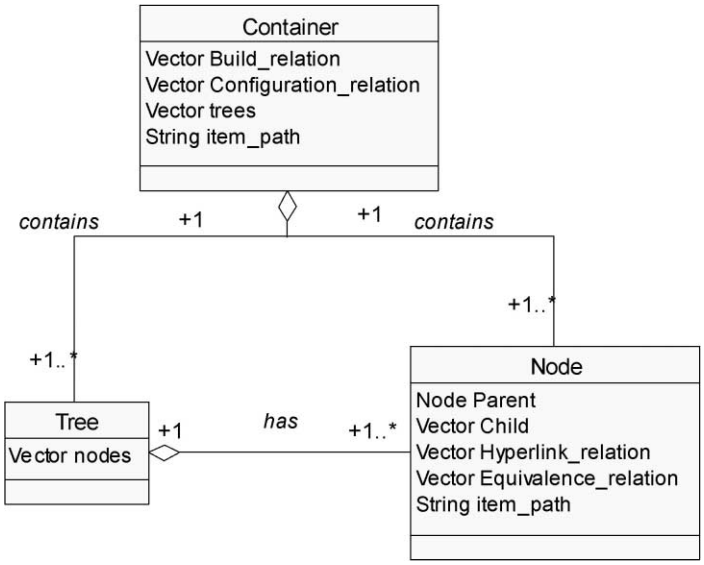

Fig. 7. Class diagram of logic structures.

Every edge in a tree represents the version and sibling relationship. Each node plays the role of a versioned item. In addition, each node keeps the equivalence relationships between itself and other versioned items. Each node also keeps hyperlinks referring to auditing documents. The path of configuration relationships will be retrieved and shown recursively from node to node in a container. Besides the version and build relationships, other relationships will be shown on the web browser on demand.

The class diagram of Fig. 7 shows the relationships between container, tree and node classes. In the real cases, there must be some structures used to keep the information about their inner ingredients, such as the physical path in the storage. Generally, those structures are projects or libraries, which could be represented as a container node in the relation graph. A container can involve other container nodes. That is, in some project, a versioned library could be a root node of a tree and it could be an internal node of a tree in another project. The container and tree classes are also used to keep the information of version, build, configuration relationships. On the other hand, all the physical stand-alone files are treated as a node structure in the relation graph. The mapping between physical files and the logic structures in CFMS can be various and it is based on users' criterion.

We have implemented a prototype of the CFMS. The following figures show the front-end GUI while using CFMS to navigate or define the configuration for

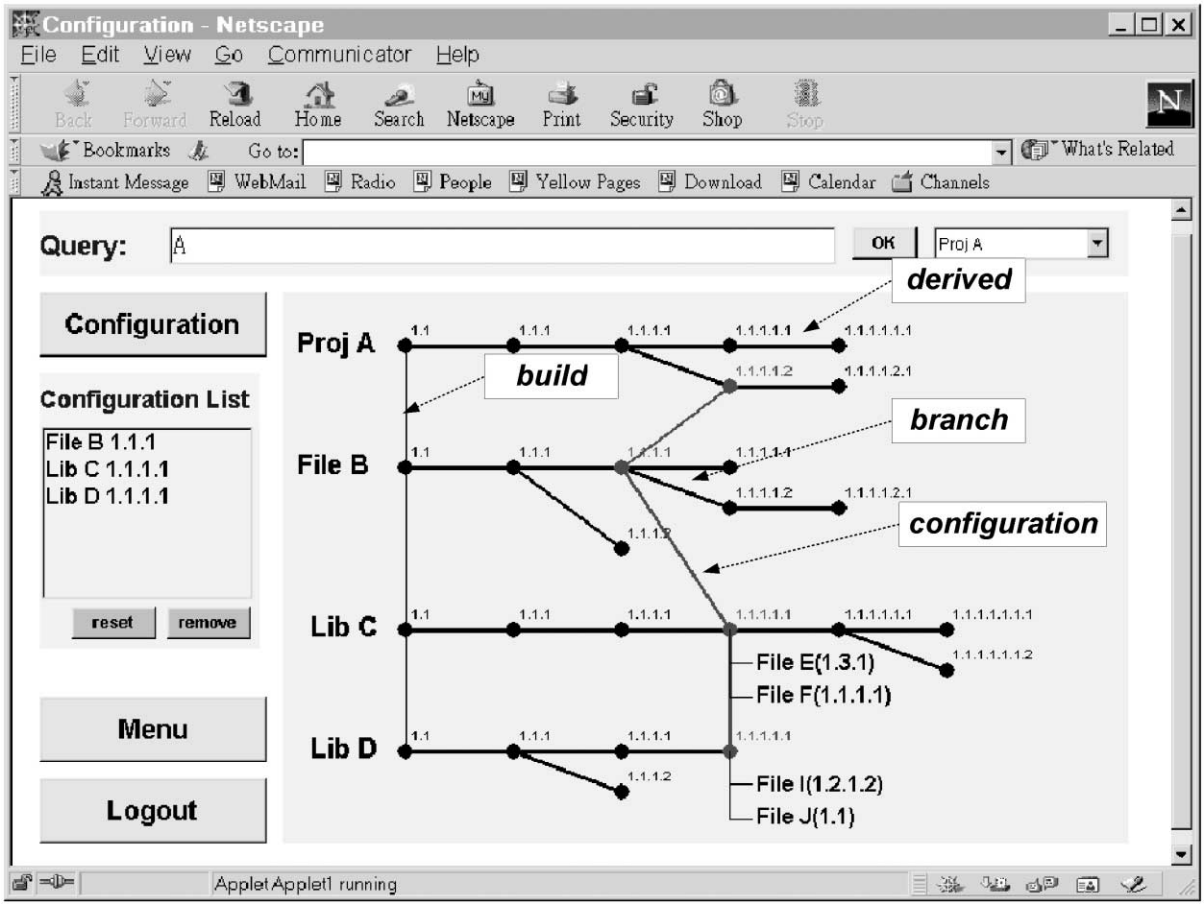

Fig. 8. CFMS clients traverse the relationship graph for configuration.

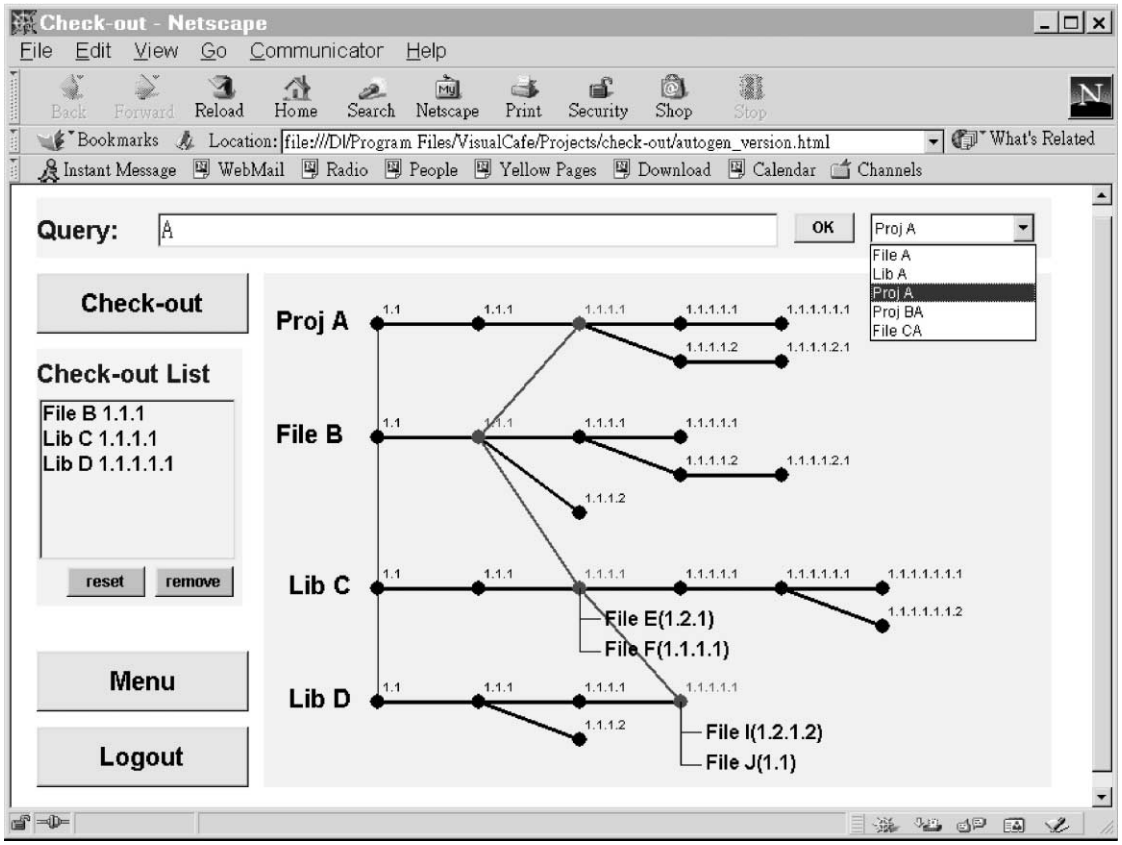

Fig. 9. CFMS clients traverse the relationship graph to check out files.

a collaborative purpose. In the prototype, a two-level navigation mechanism is designed for version selection. The first level stands for the initialization for version selection. Users can input a search term to get all the logic structures that contain the term in their version names. All logic structure will be returned by default. The second level is the visualized navigation by traversing the relation graph returned from the first level search. Because the relation graph is bi-directional, users can traverse the version graph in any valid direction.

Fig. 8 describes that a project manager is defining the configuration of version 1.1.1.1.2 for Project A. The manager first submits a query term "A" to search Project A. Then, he traverses the version graph of Project A. The selected items for version 1.1.1.1.2 are listed in the configuration list in the left side. Fig. 9 is an example to check out files. A user first submits a query term "A" to search any logic structure that contains "A" in its logic name. Then, the user selects Project A from the list box in the corner of the right side. The check out list shows items which he wants to check out. Fig. 10 describes the case to check in files. While users get into the check in menu, all the checked out items are shown on the browser by default. Users can choose items to check into the server-side file system.

5.2. A sample scenario

In this section, a scenario for using the CFMS is suggested. All users in CFMS system are cataloged into three roles. As shown in Fig. 11, they are publishers, subscribers and committees, respectively. A publisher is a user capable of checking in some documents. Generally speaking, after completing of a job,

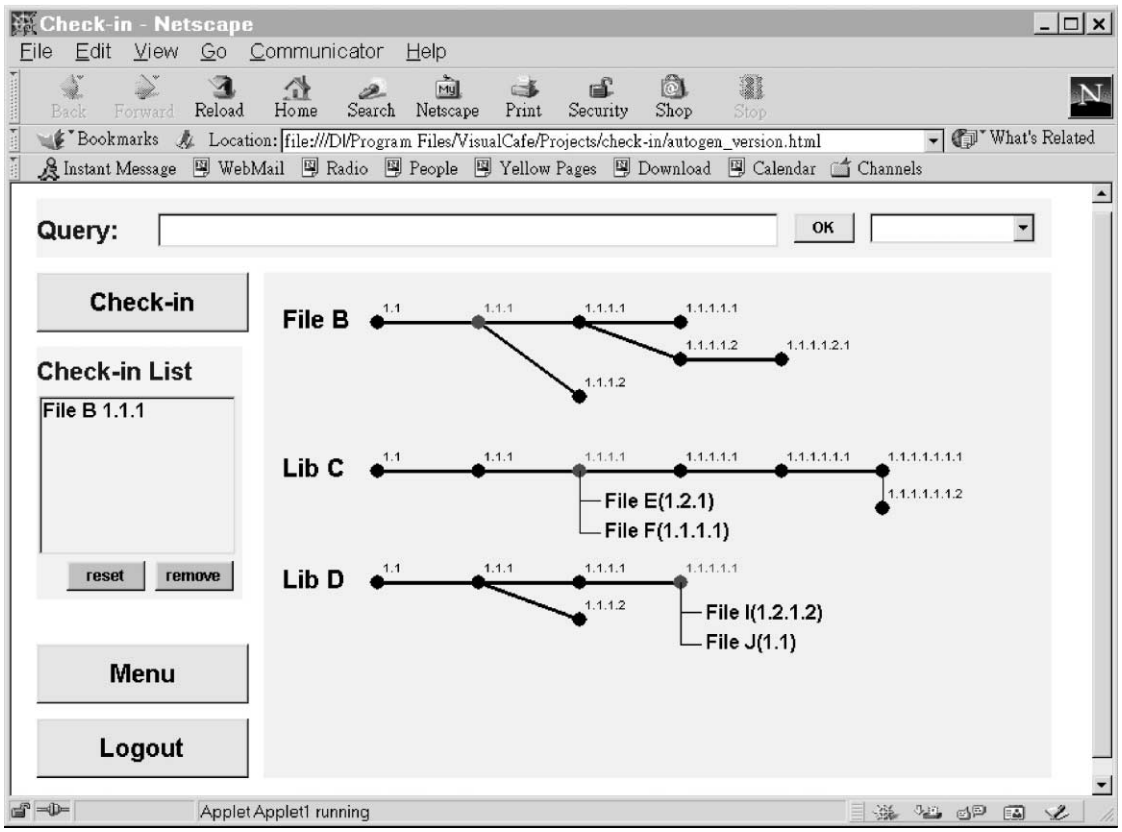

Fig. 10. CFMS clients traverse the relationship graph to check in files.

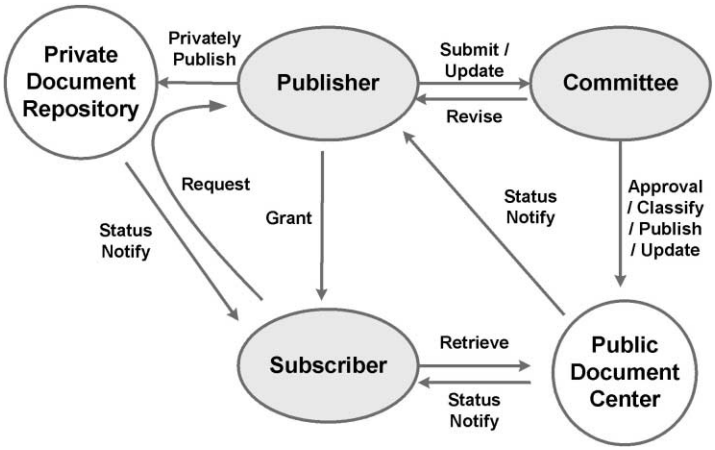

Fig. 11. A CFMS scenario.

a publisher will check documents which are related to that job into the public document center. The public document center is conceptually a unique view for all users and could be separated physically at several different disks. A subscriber is a user who checks out documents from public document center or is interested in documents. A committee is a user who has the right to decide whether a document or file is suitable to be checked into the public document center. In addition, the committee must define the access levels based on the user roles as well as documents.

Assume that in an enterprise, there are two types of view for navigating all documents or files. That is, users can search the interested files from the document relationship graph from two different viewpoints. One is presented by organizational tree view and the other is shown by catalogs of domain knowledge or projects. To complete a mission, users usually need use documents to help them analysis its complexity, and further find solutions for it. Consequently, they need to get some documents from the public document center or other employees' private working spaces. In the beginning, a user is playing the role of a subscriber and he would try to get useful documents through the conceptual view of document group using the two-level navigation mechanism. Then he checks out the interested documents from the public document center or others' private working spaces based on the access levels defined by committees. If a document is stored physically at users' private disks, the subscriber must get the grant from the publisher or document owner. Besides, if a newest version of a document is published or checked out again, the system will notify subscribers the update information automatically. After getting the documents, the user will extend, revise or renew them in his private working space. After the modification of creation of documents, the user can change his role from a subscriber to a publisher and publish the document to the public document center or just at his private working space. Committees will review the published documents and approve the qualified ones.

In this scenario, all the documents or files are conceptually organized as a unique view and can be physically stored in a public document center or dispersedly in several private user working spaces. After finding the interested documents or files, users check out documents from physical storages. Then, users can do some changes to those files downloaded previously and check them into the CFMS storages. If a new version of document is checked into the CFMS, a correct position for it will be generated to the unique conceptual view simultaneously. If a new file is checked into CFMS, committees have the responsibility to organize the document to a pertinent position in proposition to the unique document relationship graph.

6. Conclusion

In this paper, the CFMS, a web-based collaborative file management, is proposed and implemented to demonstrate its feasibility. The relationship model clarifies the relations between collaborative logic structures, which represent physical files. Integrating with the prefix-based naming scheme, the two-level version navigation provides a visualized and dynamic mechanism for version selection on WWW.

Acknowledgements

We are grateful for the many excellent comments and suggestions made by the anonymous referees. We also thank the Ministry of Education of the Republic of China for their financial support through the project titled as "MOE Program of Excellence Research" and numbered as 89-E-FA04-1-4.

References

[1] U. Asklund, Distributed development and configuration management, Licentiate Thesis, 1999. ISSN 1404-1219. http://www.cs.lth.se/~ulf/lic.html.
[2] L. Benini, et al., Distributed EDA tool integration: the PPP paradigm, in: Proceedings of the International Conference on Computer Design, 1996, pp. 448–453.
[3] F.T. Walter, RCS — a system for version control, Software Pract. Experience 15 (7) (1985) 637–654.
[4] M.J. Rochkind, The source code control system, in: IEEE Transactions on Software Engineering, SE-1, No. 12, 1975.
[5] W. David, Methods and tools for software configuration management, Wiley Series in Software Engineering Practice, Wiley, New York, 1991, pp. 89–108.
[6] ClearCase Concepts, Atria Software, Inc., Natick, MA, 1993.
[7] F.T. Walter, Configuration Management, Wiley, New York, 1994.

[8] CVS — Concurrent Versions Systems, 2000. http://www.gnu.org/software/cvs/cvs.html.

Ruey-Kai Sheu was born on August 29, 1974 in MiauLi, Taiwan, Republic of China. He received his BS and MS degrees in computer science from National Chiao-Tung University in 1996 and 1998. He is now a PhD student advised by S.M. Yuan. His research interests include distributed system design, database, object-oriented technology, and WWW technologies.

Yue-Shan Chang was born on August 4, 1965 in Tainan, Taiwan, Republic of China. He received his BS degree in electronic technology from National Taiwan Institute of Technology in 1990 and the MS degree in electrical engineering from the National Cheng Kung University in 1992. Currently, he is a candidate of PhD in computer and information science at National Chiao-Tung University. His research interests are in distributed systems, object-oriented programming, fault tolerant, and Internet technologies.

Shyan-Ming Yuan was born on July 11, 1959 in MiauLi, Taiwan, Republic of China. He received his BSEE degree from National Taiwan University in 1981, MS degree in computer science from University of Maryland, Baltimore County in 1985, and PhD degree in computer science from University of Maryland, College Park in 1989. Dr Yuan joined the Electronics Research and Service Organization, Industrial Technology Research Institute as a Research Member in October 1989. Since September 1990, he had been an Associate Professor at the Department of Computer and Information Science, National Chiao-Tung University, Hsinchu, Taiwan. He became a Professor in June 1995. His current research interests include distributed objects, Internet technologies, and software system integration. Dr Yuan is a member of ACM and IEEE.