# Dynamic Web Server Construction on the Intranet Using a Change Management Framework

Soon-Young Huh, Kyoung-Il Bae
Graduate School of Management
Korea Advanced Institute of Science and Technology
P.O.BOX 201 Cheongryang Seoul 130-650, Korea
syhuh@green.kaist.ac.kr, kibae@green.kaist.ac.kr
fax: 82-2-958-3604

## Abstract

As organizations increasingly emphasize effective collaborative working through networked systems, the World Wide Web (WWW) server and associated intranet technologies are gaining wider acceptance as a new enterprise-wide information systems paradigm. In an intranet collaborative working environment, data used in the web server are usually modified more frequently than are those used in a traditional internet web server. Thus, to support synchronous collaboration without causing any inconsistency among multiple concurrent users, the intranet web server must provide users with synchronized and consistent views of shared data. However, current web technologies have limitations in supporting this, largely because the existing Hypertext Transfer Protocol (HTTP) is unidirectional and does not allow web servers to send messages to their web browsers without first receiving requests from them. This paper proposes a web-based change management framework that can overcome such limitations and support synchronous collaboration in an intranet computing environment. The web-based change management framework facilitates management of dependency relationships between shared data in the web server and dependent user views in the client web browser and allows the web server to actively propagate changing details of the shared objects to all users referencing them. On the basis of the change management framework, we propose a dynamic web server that can support synchronous collaboration in various intranet-based collaborative systems including concurrent engineering design systems and electronic approval systems. The prototype system of the dynamic web server is developed on a commercial Object-oriented Database Management System (ODBMS) called OBJECTSTORE using the C++ programming language.

## 1. Introduction

Recently, designers and builders of traditional information systems agree unanimously on the benefits of the intranet founded on the exploding World Wide Web (WWW) and mature internet technologies [3, 14, 15, 29, 37, 39]. Using web technologies, they can enjoy the independence of operating systems as well as openness in network protocols in constructing web-based information systems. Convenient Hypertext Markup Language (HTML) [1] also contributes to fast and easy implementation of web-based applications and to seamless incorporation of the existing information systems with intranet environments. Due to the availability and cost-effectiveness of diverse internet technologies, organizations increasingly plan to convert the traditional collaborative systems such as concurrent engineering design, electronic approval, and electronic conferencing systems to intranet computing environments [6, 35]. Such web-based collaborative systems aims at helping group users to perform tasks individually in an intranet environment while achieving enhanced productivity in group work by facilitating information sharing of whole tasks. Since group users can modify the shared information concurrently in these systems, and the change may immediately affect the tasks that other users are engaged in, support for synchronous collaboration is continuously emphasized as the case in the traditional collaborative systems [15, 20, 25, 27].

In traditional collaborative systems, synchronicity has been implemented through local area network technologies and thus, group users can communicate with one another as if they have the face-to-face meeting, and concurrently modify different parts of the shared objects and be immediately aware of others' modifications through synchronized and consistent views of the shared objects [5, 7, 16, 17, 20]. However, current web technologies have limitations in supporting synchronicity, largely because the existing Hypertext Transfer Protocol (HTTP) is unidirectional and does not allow the web servers to actively send messages to their web browsers without receiving requests in advance from them. Such restrictions in current web technologies make it very difficult to support synchronous collaboration in developing web-based collaborative systems. Though not directly related to synchronous collaboration, two approaches can be referenced for providing bilateral communications in the web technologies. The first approach uses server-based communication program on the basis of widely-used standards such as X window systems or the Common Gateway Interface (CGI) [11, 37] in which a client system visualizes shared objects as directed by the server system [24]. In this approach, since server system controls every operations including concurrent modifications, change notifications, and provision of synchronized views for the client system, the client system does not need any additional program to support synchronicity other than a web browser. Limitations of this approach include restrictive enforcement of an identical communication protocol such as X protocol on both client and server systems and potential overhead burden on the server, which are often unacceptable in a heterogeneous computing environment of the intranet. The second approach adopts additional communication programs in both the client and server systems to facilitate a specialized communication service such as real-time audio/video data transfer in video-on-demand systems or real-time production status supervision in manufacturing systems [9, 37]. In this approach, the server system maintains a server communication program (e.g., real-time audio broadcasting program) that enables the server to actively send messages to the client side without. In this capacity, to create a synchronous task session, the web browser starts the client communication program and a corresponding bi-directional communication channel is assigned between the client and server for subsequent data transfer. Despite the support of bi-directional communication on the web technology, this approach has limitations in supporting synchronous collaboration in the intranet environment due to the lack of the change management functions including dependency management and change notification. In this sense, the two approaches are only partially successful in supporting synchronous communication between web server and client systems, but are not yet satisfactory in presenting a framework how to support the change management functionalities in the intranet-based collaborative systems.

To overcome these problems, in this paper, we propose a web-based change management framework for facilitating synchronous collaboration in the intranet environments, and present a dynamic web server employing the proposed framework. In the framework, we specifically address the following three issues: (1) How can intranet-based collaborative systems manage concurrent users accessing shared objects in terms of  dependency management as well as security control ? (2) When a shared object under a collaborative system is changed, how can the system facilitate immediate change  notification to all users viewing it with their web browsers ? (3) How can such a change management framework be implemented in the existing web-based collaborative systems without requiring any modification on the web server and browser programs ? In addressing these questions, our work focuses on the development of a dependency management mechanism and a change notification mechanism in the intranet environments. In developing the framework, we adopt an object-oriented database model [18] and an agent approach [10, 27, 36] to effectively conceptualize the core constucts and mechanisms constituting the framework. On the basis of the change management framework, we present a dynamic web server that can be incorporated into various intranet-based collaborative systems as an underlying platform supporting synchronous collaboration. The dynamic web server is implemented on a commercial Object-oriented Database Management System (ODBMS) called OBJECTSTORE [18] using the C++ programming language [32] and under the Windows NT operating system [13].  To show the wide applicability of the dynamic web server, its use are further discussed in two collaborative systems including concurrent engineering design systems and electronic approval systems.

The paper is organized as follows. In Section 2, core concepts of the change management framework are described and extended to propose a web-based change management framework that can be applied to collaborative systems in the intranet environments. In Section 3, we present the dynamic web server as a generic change management component for web-based colloaborative systems and show the detailed procedures of dependency management and change notification mechanisms on the basis of current web technologies. In Section 4, applications of the dynamic web server to two collaborative systems including concurrent engineering design systems and electronic approval systems are discussed to show wide applicability of the framework. In Section 5, we summarize our work and provide research contributions.

## 2. Web-Based Change Management Framework

In this section, we describe core constructs of the change management framework and introduce new constructs to extend the dependency maintenance and change notification mechanisms in the intranet environments.

### 2.1. Core Constructs of the Change Management Framework

For management of dependency relationship and change notification, several approaches have been proposed in traditional distributed or collaborative systems including Model-View-Controller (MVC) framework [12, 30], Distributed Shared Memory (DSM) system [33, 38], and change management framework [16]. Among them, the change management framework is specifically adopted as a basis of our framework since it is more flexible and has wide applicability to collaborative systems. It can support both transient and persistent shared objects as well as multiple concurrent users in a client-server computing environment, which can be easily extended to an intranet computing environment.

Change management needs in traditional collaborative systems usually arise between shared objects and their dependent user views. To adopt the two in a generalized way, the change management framework uses two primary structural constructs: supporter and dependent. A **supporter** is an object to be referenced and modified as a basic shared source, ranging from textual documents to multimedia data. It is transient in memory or persistent in databases. A **dependent** is an object that provides a visual representation of the supporter or determines the precise fashion in which the supporter is to be manifested. Each dependent in a user’ s computer is created within a process that runs as a single program. As a dependent successfully references a supporter, a **dependency relationship** is established between the two.

Figure 1 exemplifies these constructs in an engineering design environment. On the left-hand side of Figure 1, the tree data structure configuring a bicycle is a supporter as a shared object. On the right-hand side, several representations of the tree structure exemplify different users' views of the same supporter: a product view with a graphic of a bicycle at the top, an indented textual list view in the middle, and a cost management accounting view calculating the overall cost of the product’ s components at the bottom. These three views become the dependents of the tree structure supporter. Considering that the three views can be maintained in multiple processes, the supporter will have three dependency relationships.

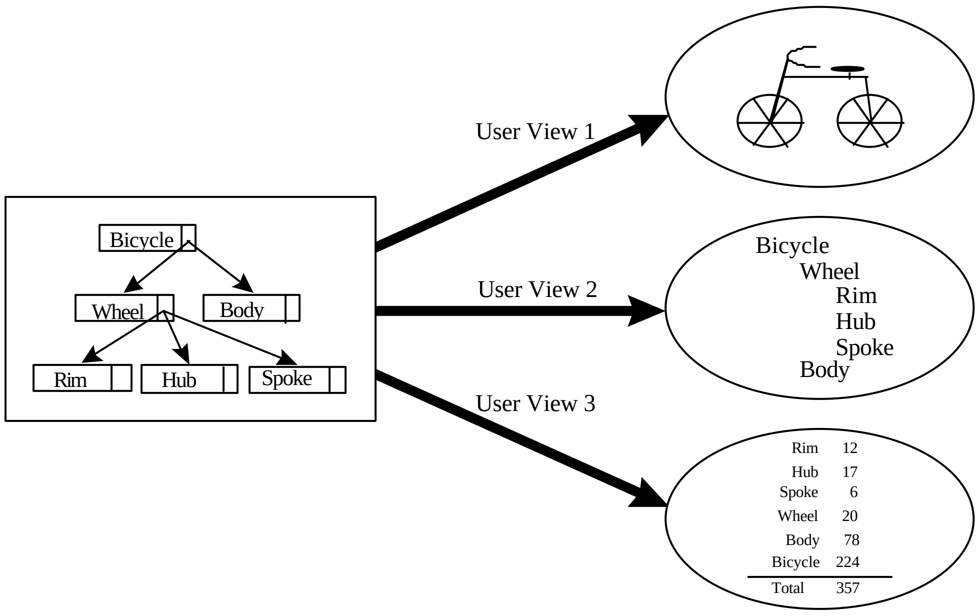


Figure 1. A Supporter and Dependents in an Engineering Design.

Shared objects are apt to be changed dynamically and irregularly during workgroup computing sessions. Therefore, integrity within the dependency relationship is important for securing consistency. This ensures that dependent user views remain synchronized with the current status of the supporter.

In such capacity, the change management framework provides the mechanisms to maintain the dependency relationship as well as synchronicity. Specifically, it provides a **dependency**

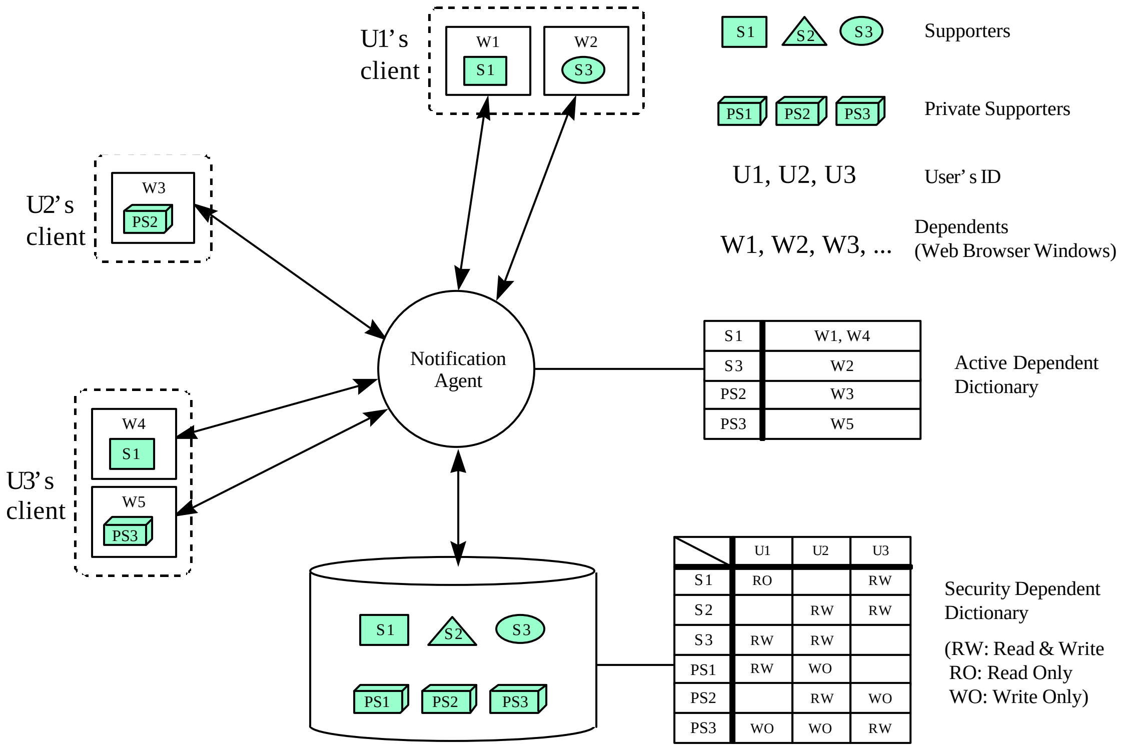
**management mechanism** and a **change notification mechanism**. The former primarily maintains dependency relationships, while the latter preserves integrity within dependency relationships. The dependency management mechanism uses a **dependent dictionary**, which is a data structure representing dependency relationships among multiple dependents and supporters. In addition, the change notification mechanism ensures that the dependents consistently mirror the current valid image of the supporters by continuously reflecting changes made to it. When a supporter is changed, the change notification mechanism is activated by broadcasting its corresponding update requests to the dependent sets of the supporter.

In the following section, we extend the traditional constructs of the change management framework into the intranet computing environment.

## 2.2. New Constructs of the Web-Based Change Management Framework

Figure 2 shows an example of a web-based collaborative system that consists of a server and three client systems that are distributed in different systems over the computer network. We assume that each client is used by one user (U1, U2, or U3) and displays one or more web browser windows (W1, W2, to W5). In Figure 2, S1, S2, and S3 are the supporters residing at the transient or persistent storage of the server, while the client web browser windows are their dependents that display HTML documents containing the supporters.

In web-based collaborative systems, supporters can be categorized into two classes, depending on whether they can be accessed in public or not. Public supporters are owned and accessed as collaborative tasks by the workgroup in the intranet. Since members of the workgroup want to interact closely with one another to achieve a common goal in a given project time frame, they usually access or modify the same public supporter in a synchronous manner. Thus, public supporters are highly shared by members but are protected from non-members. The engineering design supporter in Figure 1 is a typical example of a public supporter. In contrast, workflow applications such as electronic approval systems or order processing systems allow individual members to have their own local information, such as personal information folders or private to-do lists. **Private supporters** are individually owned by each member instead of the workgroup, and accommodate such private local folders as a part of communication channels that are loosely coupled among the members. In these loosely coupled applications, only the owner member can read arriving electronic messages or new tasks in the private supporter, while both owner and non-owner members can change the contents of the private supporter by sending electronic messages or assigning tasks. In Figure 2, such private supporters are denoted as PS1, PS2, and PS3. We term public supporter as supporter as a short name, as opposed to private supporters.



| | |
|---|---|
| S1 | W1, W4 |
| S3 | W2 |
| PS2 | W3 |
| PS3 | W5 |

| | U1 | U2 | U3 |
|---|---|---|---|
| S1 | RO | | RW |
| S2 | | RW | RW |
| S3 | RW | RW | |
| PS1 | RW | WO | |
| PS2 | | RW | WO |
| PS3 | WO | WO | RW |

Figure 2. Dependency Relationship of the Web-Based Change Management Framework.

Since supporters can be accessed and updated by various types of users who have different access authorities, a user-level security policy needs to be additionally considered to control access to the supporters. Thus, as a construct to ensure user-level security, we define a security relationship and a security dependent dictionary that manages the security relationship.

- A **security relationship** specifies the access right relationship between a supporter and a user to allow the authorized user to access and make changes to the supporter. It provides a simplistic uni-level security schema where individual workgroup users are assigned with a limited set of rights. For each supporter, users can have three types of access rights: Read Write (RW), Read Only (RO), and Write Only (WO). For instance, the Read Write right allows the user to both read and write the supporter while the Read Only right only allows to read the supporter. Thus, when a new supporter is to be added, the security relationship needs to be established in advance to prescribe the access relationship among the supporter and the authorized users. Conversely, when the supporter is no longer used, all corresponding security relationships need to be removed. Depending on the security requirements of the collaborative system, more sophisticated security system can be implemented [22, 31, 34] in the security relationship.

- A **security dependent dictionary** is a data dictionary to manage security relationships. When a security relationship is established, it is inserted into the security dependent dictionary, which specifies the composition of a supporter, an authorized user, and the user' s access right. As shown in Figure 2, the content of the security dependent dictionary can be represented by a matrix with columns and rows representing the users and supporters respectively, and each matrix entry denoting access rights. Every empty matrix entry implies that the user in the corresponding column has no access right on the supporter in the corresponding row. When collaborative computing environments are changed with respect to users' access rights, task roles, or working sequences on the supporters, security relationships in the security dependent dictionary should be revised accordingly.

Next, to support concurrency control with shared objects or multiple users, we define an active relationship between a supporter and a dependent, which is dynamically added and removed depending on the user' s creation of dependents with respect to the supporter.

- An **active relationship** is a dependency relationship between a supporter and a dependent currently referencing the supporter. When a dependent starts to reference a supporter, a corresponding active relationship is established. When the dependent switches to another supporter or is removed, the active relationship needs to be removed immediately. Due to the user-level security policy, a dependent can reference only these accessible supporters that are allowed in the security relationships. In Figure 2, the active relationship between supporter S1 and dependent W1 shows that user U1 using dependent W1 already has the access right (either RO or RW) in the security relationship with supporter S1.

To manage the active relationships systematically, we also introduce an active dependent dictionary.

- An **active dependent dictionary** is a data dictionary to manage active relationships. When an active relationship is established, it is inserted into the active dependent dictionary to register the relationship between a supporter and a dependent. When registered, the network channel associated with the dependent is additionally inserted in the dictionary to help locate the dependent in the network. When the dependent is removed or accesses another supporter, the active relationship is immediately removed from the active dependent dictionary. In Figure 2, the content of the active dependent dictionary is represented by a table. In the table, the first column represents active supporters being used by the dependents while the second column of the table displays the current dependents which reference the individual supporters. Over the collaborative computing sessions, the contents of the active dependent dictionary tend to be changed much more frequently than those of the security dependent dictionary. To ensure robustness and efficiency of these dictionaries, the ODBMS-based dictionary tool is adopted [16, 18].

Using the security and active relationships implemented on the dictionaries, we can provide the dependency maintenance and the change notification mechanisms in the intranet environment. To facilitate such mechanisms, a **notification agent** is introduced as an intermediary between a server and its client systems. The notification agent has two main responsibilities: managing the active dependent dictionary and broadcasting the change notification messages generated by the server to relevant dependents. First, regarding the management of the active dependent dictionary, when a dependent accesses a supporter, the notification agent adds the active relationship with the associated network channel between the supporter and the dependent into the active dependent dictionary. The notification agent also removes the active relationship if the dependent is removed and the corresponding network channel is disconnected.

Second, in terms of message broadcasting, when a workgroup member modifies a supporter, the server sends the change notification message associated with the supporter to the notification agent. The notification agent then looks up the active dependent dictionary and identifies the current dependents of the supporter and broadcasts the change notification message to the dependents. This is a contrasting advantage over the traditional web server, since the traditional one does not have any means to keep track of the dependent web browsers that are currently referencing the server. Besides, location transparency is achieved in broadcasting change messages to relevant client systems. When a client makes a change to a supporter, the server storing the supporter is not required to know which client systems it has to notify of the change. All it has to do is to send the notification agent a change notification message with the identity of a modified supporter. Once the message is delivered, the notification agent takes all responsibility of broadcasting the change notification message to adequate dependents.

## 3. The Structure of the Dynamic Web Server

In this section, we present a dynamic web server that incorporates the web-based change management framework and serves as an underlying platform in making web-based collaborative systems. We first describe system components of the dynamic web server, and second detail the dependency management procedure and the change notification procedure in the dynamic web server.

## 3.1. System Components of the Dynamic Web Server

The dynamic web server is composed of three units: a web service unit, a data management unit, and a notification broadcasting unit as shown in Figure 3. These units may either reside in a machine or be distributed in different machines. A **web service unit** consists of a web server process and CGI applications. The web server process interacts with the individual user' s web browser and accepts external HTTP requests from the browser. When it receives a HTTP request to access a supporter, the web server process invokes and gets results from relevant CGI applications, which perform access or modification on the supporter. A **data management unit** is comprised of a database system handling both persistent storage and transaction management, and server processes managing the security dependent dictionary. The processes also take charge of notifying operation to the notification agent of the changes made to supporters. Lastly, a **notification broadcasting unit** consists of a notification agent and an active dependent dictionary. The notification agent maintains the active dependent dictionary and broadcasts change notification messages issued by the data management unit to adequate dependents.

As contrasted with the three units of the dynamic web server, the client side needs only one additional communication agent, a **delegate agent**. The delegate agent is dynamically added and removed from a client web browser since it is implemented as a JAVA applet [4] that is usually transferred from the server upon being invoked by the client. The primary task of the delegate agent is to enable the server units to initiate communication to the client. More precisely, when a client web browser requests the web service unit to send an HTML document containing a supporter, the service unit sends the client not only the HTML document but also the delegate agent. After arriving at the client web browser, the delegate agent performs two tasks:

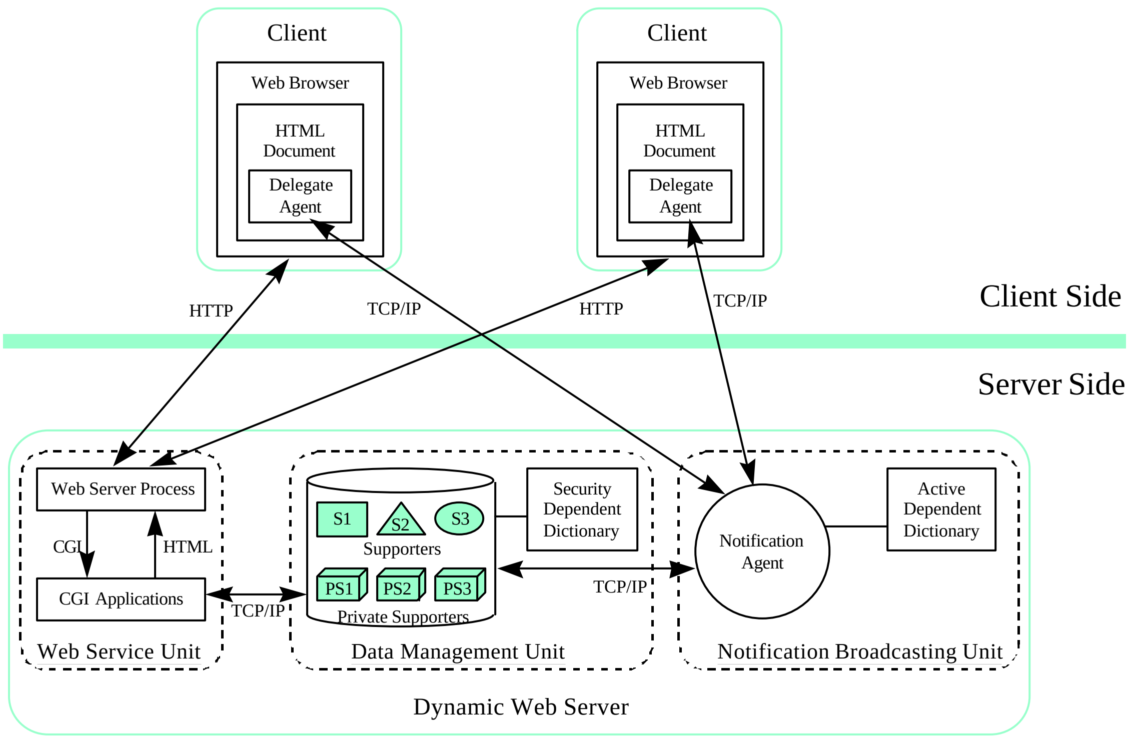


Figure 3. System Components of the Dynamic Web Server.

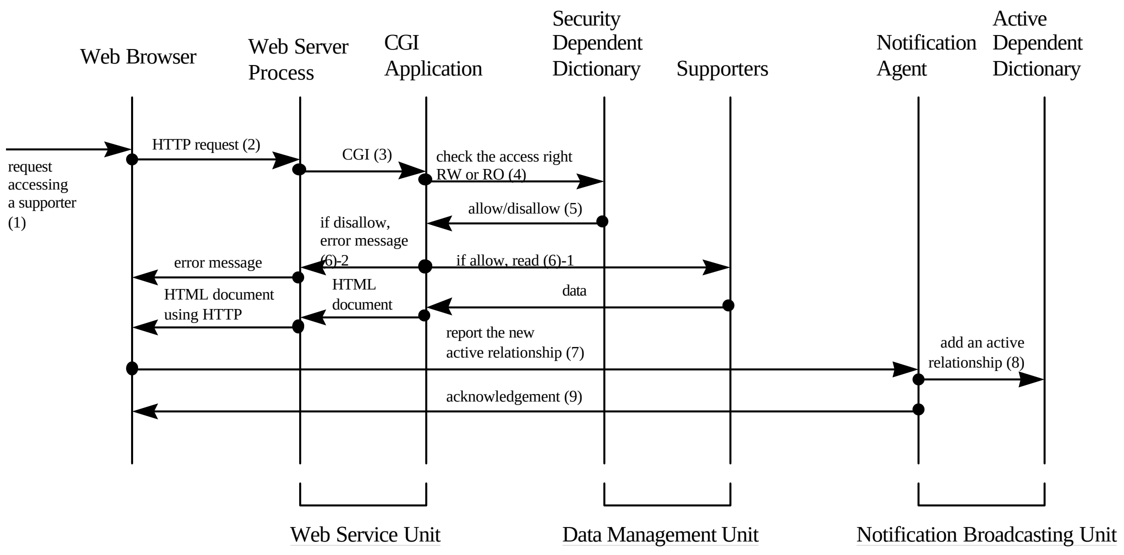
registration of an active relationship into the active dependent dictionary and reception of change notification messages from the notification agent. Firstly, the active relationship is registered as follows. Right after arriving at the client browser, the delegate agent attempts to establish a network channel with the notification agent for subsequent communication with the notification broadcasting unit. If the channel is successfully established, the delegate agent reports the new active relationship between the supporter and the dependent (web browser) to the notification agent for registration of the relationship in the active dependent dictionary. By so doing, the server, i.e., notification broadcasting unit, is now equipped with a channel to recall and actively initiate communications with the client. Once the channel is established, the second task, reception of change notification messages, can be taken in the delegate agent. When the supporter is modified, the notification broadcasting unit can immediately initiate communication with the delegate agent through the established channel and let the delegate agent handle the change notification messages. In this way, the dynamic web server can not only receive requests from the client systems but also have an active means to access the client systems.

Consequently, the dynamic web server overcomes the unidirectional property of the existing HTTP and allows bi-directional communication between server and client. Moreover, the notification unit of the dynamic web server and the web browser equipped with the JAVA applet delegate agent provides strong software compatibility. Existing web server programs can be employed in the web service unit without requiring any modification, since the notification unit communicates with only data management unit and has no direct interaction with the web server process. Similarly, any JAVA-compatible web browser can be used as a client program since the server does not require any particular network protocols except plain TCP/IP.

## 3.2. Dependency Management

In Figure 4, we illustrate the detailed steps of the dependency management procedure in the dynamic web server by providing an event trace diagram describing the interactions between a client web browser and the three units in the dynamic web server. In the diagram, individual participating objects and events are represented respectively by a vertical line and a line of horizontal arrows from the sender object to the receiver object.

In Figure 4, we assume that both security and an active dependent dictionary are in place, and all network channels between the notification agent and delegate agents are successfully established. By typing a precise Uniform Resource Location (URL) or clicking a specific link, a user can access a supporter with his/her web browser ①. The web browser then sends a corresponding HTTP request to the web server process of the web service unit ②, and the web server process starts a CGI application that executes the following three steps to carry out the



Web Browser
Web Server Process
CGI Application
Security Dependent Dictionary
Supporters
Notification Agent
Active Dependent Dictionary
request accessing a supporter (1)
HTTP request (2)
CGI (3)
check the access right RW or RO (4)
allow/disallow (5)
if disallow, error message (6)-2
error message
if allow, read (6)-1
HTML document using HTTP
HTML document
data
report the new active relationship (7)
add an active relationship (8)
acknowledgement (9)
Web Service Unit
Data Management Unit
Notification Broadcasting Unit

Figure 4. An Event Trace Diagram for the Dependency Management Procedure.

HTTP request ③. First, referring to the security dependent dictionary of the data management unit, it checks the requesting user’ s authority with which he/she could access the supporter ④. Second, if the user is allowed to access the supporter (RO or RW access right) ⑤, the CGI application retrieves the supporter from the data management unit ⑥-1. If not, the application sends an error message to the user via the web server process and stops instantly ⑥-2. Third, the application makes an HTML document containing the supporter and a delegate agent, which will be sent to the requesting user’ s web browser via the web server process. On arriving at the web browser, the delegate agent generates a network channel over TCP/IP with the notification agent of the notification broadcasting unit. Then, to register the new active relationship between the supporter and the dependent (web browser window), it sends a message to the notification agent through the established network channel ⑦, and the network channel is made alive over the browsing session accessing the HTML document that contains the supporter. On receiving the message from the delegate agent, the notification agent adds the active relationship to the active dependent dictionary ⑧ and sends an acknowledgment message back to the delegate agent ⑨.

## 3.3. Change Notification

In Figure 5, we illustrate the steps of the change notification procedure in the dynamic web server using an event trace diagram. We also assume that some active relationships in the active dependent dictionary and network channels between the notification agent and delegate agents are already established.

When a user attempts to modify a supporter with his/her web browser ①, the web browser sends a corresponding HTTP request to the web server process of the web service unit ②. The web server process then starts a CGI application, carrying out the HTTP request ③. This CGI application executes the following three steps. First, it checks the requesting user' s authority with which he/she could make changes on the supporter ④. Second, if the user is allowed to make changes on the supporter (RW or WO access right) ⑤, the CGI application performs change operations for the supporter through the data management unit ⑥-1. Otherwise, the application sends an error message to the user via the web server process and stops instantly ⑥-2. Third, the application makes an HTML document containing the results of the change operations (e.g. changing details or system messages), which will be sent to the requesting user' s web browser via the web server process when the change operations are committed. Meanwhile, the data management unit sends a change notification message containing the identity of the changed supporter to the notification agent ⑦. Then, referring to the active dependent dictionary, the notification agent chooses the dependents currently accessing the changed supporter ⑧ and broadcasts the change notification message to them through existing network channels ⑨. Of course, each client side recipient of the change notification message is the delegate agent residing in the dependent accessing the changed supporter.

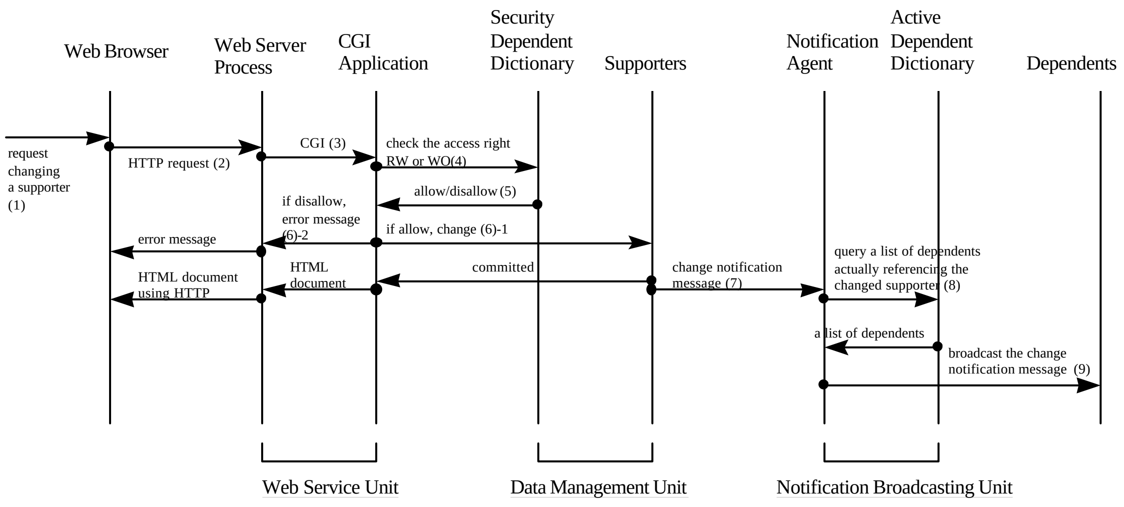

Web Browser
Web Server Process
CGI Application
Security Dependent Dictionary
Supporters
Notification Agent
Active Dependent Dictionary
Dependents
request changing a supporter (1)
HTTP request (2)
CGI (3)
check the access right RW or WO(4)
allow/disallow (5)
if disallow, error message (6)-2
if allow, change (6)-1
error message
HTML document using HTTP
HTML document
committed
change notification message (7)
query a list of dependents actually referencing the changed supporter (8)
a list of dependents
broadcast the change notification message (9)
Web Service Unit
Data Management Unit
Notification Broadcasting Unit

Figure 5. An Event Trace Diagram for the Change Notification Procedure.

## 4. Applications of the Dynamic Web Server to Collaborative Systems

In this section, we apply the dynamic web server to two typical intranet-based collaborative systems, concurrent engineering design systems and electronic approval systems. Though these collaborative systems are typical applications, their characteristics are different. To better understand the different characteristics of the two systems, we can compare them from several perspectives of collaborative systems such as content, time, process, and coupling in common tasks. In typical concurrent engineering design systems, members of an engineering design team divide the full drawing of a certain product into their several sub-drawings and bring it to completion by individually making the sub-drawings [2, 21, 28]. Due to the increasing need for rapid preparation of prototype product, they may work in a parallel but want to have synchronous collaboration. Thus, though handling sub-drawings individually, each member wants to be kept notified of other colleagues' recent progresses before the completion of the work. In such a design system, coupling around the shared tasks is usually tight. Meanwhile, an electronic approval system is a kind of workflow system that helps to define, execute, coordinate, and monitor the flow of approval tasks along a sequence of processes defined in an organization. Examples of the electronic approval system include order handling systems, credit checking systems, and tax return filing systems [8]. In such capacity, collaborative tasks are executed in a sequential manner, since a member in a department cannot carry out any task unless his/her preceding members complete and pass the task. For a task to be accomplished, individual members work in an asynchronous manner, and thus their coupling is rather loose. The comparison between the two is summarized in Table 1. We use these two different applications to show that our framework can be applied to diverse application domains.

| Application System | Content | Time | Process | Coupling in Common Tasks |
|---|---|---|---|---|
| Concurrent Engineering Design Systems | Design Drawings | Synchronous | Parallel | Tightly Coupled |
| Electronic Approval Systems | Text, Charts, Graphs | Asynchronous | Sequential | Loosely Coupled |

Table 1. Characteristics of Concurrent Engineering Design Systems and Electronic Approval Systems.

## 4.1. Concurrent Engineering Design Systems

To better support such individualized sub-drawing tasks and at the same time achieve higher workgroup productivity in full drawing, concurrent engineering design systems need to offer each member synchronized and consistent views of the sub-drawings that are being made by others [2, 21, 28]. Since the members usually access or modify the same drawings in a synchronous manner, the drawings are suitable to be stored in our framework as supporters. Suppose that the full drawing consists of four sub-drawings. Each sub-drawing is stored as an individual supporter (S1, S2, S3, or S4) in the data management unit of the dynamic web server and is assigned to one user (U1, U2, U3, or U4) for separate discretionary design work step. Table 2 shows the security dependent dictionary of this example. For instance, only user U1 can modify supporter

S1 (with the RW access right), whereas users U2, U3, and U4 can read it (with the RO access right). When U2 attempts to access S1 with the web browser, the CGI application handling read operation in the server validates U2' s access right for S1 and provides U2' s web browser with an HTML document containing S1 and a delegate agent. On arriving at the web browser, the delegate agent asks the notification agent of the notification broadcasting unit to register the established active relationship between S1 and U2' s web browser window in the active dependent dictionary. Afterwards, when U1 attempts to modify S1, the CGI application handling modification first checks U1' s access right for S1 and performs modification operations on S1. After these operations have been completed, the change notification described in Section 4 takes place. That is, the data management unit sends the change notification message concerning the change of S1 to the notification agent of the notification broadcasting unit. Then, with the help of the active dependent dictionary, the notification agent chooses the users currently accessing S1 and broadcasts the change notification message to them. On the client side, the delegate agent receives this message and informs the user of it. If U2 is still accessing S1, the user will recognize the change of S1 and get the recent image of S1 through the automatic update of the web browser window. By providing consistent design views in such a manner, the dynamic web server facilitates the construction of the concurrent engineering design systems, thereby enriching synchronous collaboration on intranet environments.

|  | U1 | U2 | U3 | U4 |
|---|---|---|---|---|
| S1 | RW | RO | RO | RO |
| S2 | RO | RW |  |  |
| S3 |  |  | RW | RO |
| S4 |  |  | RO | RW |

Table 2. Security Dependent Dictionary of the Concurrent Engineering Design System.

## 4.2. Electronic Approval Systems

Electronic approval systems usually support asynchronous collaboration among members who are involved in approval tasks in distributed environments. In such environments, it is frequently the case that an approval task does not start, even though all prerequisite tasks have been completed. If accumulated, such delay in the electronic approval system could result in a serious problem in the organization by making the organizational decision-making processes frequently delayed and long. To resolve this, electronic approval systems need to be able to immediately notify the members of newly arrived tasks, and to make the approval tasks move more timely. Such semi-synchronous collaboration is thus useful in electronic approval systems, although not seriously required as in the concurrent engineering design systems.

In defining the work sequence of the workflow, workflow systems including electronic approval systems can be represented by workflow maps specifying the structure of work procedures (i.e., predefined work steps and partial ordering of these steps) and activities constituting the bodies of the work steps [8]. If "order entry", "credit checking", "billing", and "shipping" are work steps in the order processing system, then the work step "send out letter" may be an work step of these steps such as "order entry" and "billing". A hypothetical workflow map for an electronic approval system is provided in Figure 6. The ellipse of Figure 6 exhibits a work step, and the number inside the ellipse represents the identity of the user responsible for the corresponding work step. A set of arrows and work steps represent a work procedure that can be the whole electronic approval system such as order processing system. The decision status of the individual work step is represented by a value (either 'approved' or 'rejected') associated with an arrow.

In Figure 6, the work step of user U6 receives a task that have been approved by user U4. Then, if U6 approves it, the task is passed to the next work steps of both U7 and U8. Otherwise, it redirects this task to the work step of user U5. As time goes by, tasks may be stacked for each work step. To manage such tasks accumulated in individual work steps, every work step in electronic approval systems has its own to-do list. The to-do list is private and only readable by the owner of the work step, but is modified by others who are allowed to assign new tasks to the work step. The to-do list can be compared with the shared drawing in the concurrent engineering

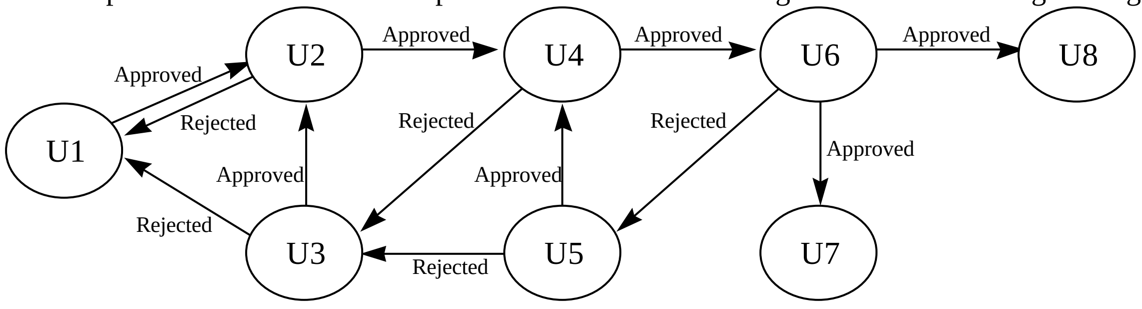


Figure 6. A Workflow Map of the Electronic Approval System.

design systems. Both are similar in that they become shared supporters. However, the shared drawing cannot be modified by others. Only the owner can modify the shared drawing, while others can read it. The to-do list can be adequately accommodated by the private supporter; only the owner can read arriving tasks in the private supporter, while both owner and non-owner members can change the contents of the private supporter by assigning tasks. This is specifically suitable in the intranet environments, since each member can access his/her own private supporter or modify other members' private supporters to assign new tasks through the web browser.

The decision-making process in Figure 6 involves eight members (U1 to U8). The corresponding to-do lists for individual members are respectively stored in private supporters (PS1 to PS8) in the data management unit of the dynamic web server. Table 3 shows the security dependent dictionary of the example illustrated in Figure 6. For instance, only user U4 can read his own to-do list by accessing the private supporter PS4 (due to RW access right) and perform subsequent task handling, whereas only users U2 and U5 can pass the task to U4, since they are allowed to modify PS4 (due to WO access right). In such capacity, the change notification mechanism can bring enhanced synchronicity to the task approval process. More

|  | U1 | U2 | U3 | U4 | U5 | U6 | U7 | U8 |
|---|---|---|---|---|---|---|---|---|
| PS1 | RW | WO | WO |  |  |  |  |  |
| PS2 | WO | RW | WO |  |  |  |  |  |
| PS3 |  |  | RW | WO | WO |  |  |  |
| PS4 |  | WO |  | RW | WO |  |  |  |
| PS5 |  |  |  |  | RW | WO |  |  |
| PS6 |  |  |  | WO |  | RW |  |  |
| PS7 |  |  |  |  |  | WO | RW |  |
| PS8 |  |  |  |  |  | WO |  | RW |

Table 3. Security Dependent Dictionary of the Electronic Approval System.

precisely, when U2 or U5 passes a new approval task into U4’s to-do list, i.e., PS4, the data management unit sends a change notification message concerning the change made in PS4 to the notification agent of the notification broadcasting unit. The agent then refers the active dependent dictionary and checks whether U4 currently has a live session accessing PS4 or not. If U4 does, the notification agent sends the change notification message to U4. In response to this notification process on the server side, the client delegate agent on U4’s web browser receives this message and informs U4 of the change made in PS4. As such, the dynamic web server also enables the electronic approval system in the intranet environments to monitor the flow of approval tasks and notify the users of the change of their own to-do lists. Such benefits of the dynamic web server would help electronic approval systems to reduce the task transition time between work steps and consequently contribute to decrease of the total time required to perform the decision-making process in the intranet environments.

## 5. Summary and Conclusions

In the intranet-based collaborative systems, synchronous collaboration is increasingly requested to achieve higher workgroup productivity as well as to reduce overall completion time for collaborative work. For synchronous collaboration in the intranet environment, dependency management with a certain security policy and change notification mechanisms are necessary. However, research on the intranet-based collaborative systems has not been yet satisfactory largely due to the lack of effective change management support and limitations of current web technologies such as unidirectional HTTP communication protocols. Having recognized the problems of existing intranet systems, we present a web-based change management framework and a dynamic web server that can serve as a generic underlying platform in making web-based collaborative systems which incorporates the proposed change management framework.

The web-based change management framework is to support the dependency mechanism and the change notification mechanisms between collaborative shared tasks and web browser user views. In providing the dependency management mechanism, our framework supports a user-level security policy to prevent unauthorized access to the supporters. It also manages active dependency relationships between the supporters and their dependent users' views that are currently alive over the corporate intranet. To manage both dynamically changing relationships effectively, the framework uses dependency dictionaries maintaining dependency pairs between supporters and authorized users or dependent user views. In terms of change notification mechanism, the framework provides an web-based broadcasting mechanism that allows the web server actively propagate change messages to all dependent user views when a supporter is changed. By virtue of these mechanisms, all the web-based collaborative workgroup users can have consistent synchronized views of the shared tasks that keep evolving. To present the generality as well as the practicality of the framework, we implemented the prototype of the dynamic web server on a commercial ODBMS called OBJECTSTORE under the Microsoft Windows NT operating system. We used WebSite 1.1 [26] as a web server program and Netscape Navigator 3.0 [19, 23] as a web browser.

Contributions of the research can be summarized as follows. Firstly, the dynamic web server developed on the web-based change management framework provides a generic system platform, which, we believe, would facilitate the development of intranet-based collaborative systems. Specifically, application system development will be made easier since our framework are developed on the basis of an object-oriented database model in which all the core mechanisms including agents, dictionaries, service units, supporters and dependents are designed as self-contained objects, implemented in C++ classes. As seen in the prototypes of concurrent engineering design system and electronic approval system, the collaborative systems adopting the proposed dynamic web server can immediately take advantage of the dynamic dependency management capability as well as real time-based change notification facility so that all collaborating users in the intranet can have synchronized views on the shared tasks. Consequently, the dynamic web server resolves the limitations of the unidirectional communication protocol in existing web technologies and thus makes possible the full-fledged collaboration in the intranet computing environment. Moreover, the embedded support for the user-level security offers a means to make the collaboration system to be well-protected against unauthorized accesses in the intranet.

Second, since software compatibility has been emphasized in designing the dynamic web server, both existing web browsers and servers can be used as collaborative user view tools and servers without any serious modification. This is largely because the proposed dynamic web server consists of three well-defined autonomous modules and adopts an agent approach which can effectively coordinate all the intermediate dependency management and bi-directional communications between server and collaborative user views.

Third, the dynamic web server brings in various types of transparencies in managing dependency relationship and broadcasting the notification messages. Persistence and concurrence transparency are attained since the ODBMS is used as an underlying system and thus concurrent transient dependent views and persistent shared tasks are seamlessly accommodated on a single formalism. Location transparency is also achieved since dependent web browsers are managed in terms of dependency relationship and are thus kept synchronized wherever they are located in the intranet.

As a future research direction, the dynamic web server has strong potential in various internet application areas, specifically in the electronic commerce area. We think that automated product selection aid or agent-based commerce in the electronic commerce can have more powerful functionality and intelligence if they are integrated with the proposed dynamic web server.

## References

[1] T. Berners-Lee and D. Connolly, "Hypertext Markup Language - 2.0", *Request For Comments 1866*, 1995.

[2] Tzy-Shuh Chang and Allen C. Ward, "Conceptual Robustness in Simultaneous Engineering: A Formulation in Continuous Spaces", *Research in Engineering Design*, vol. 7, no. 2, pp. 67-85, 1995.

[3] David Coleman, "Collaborating on the Internet and Intranets", in *Proceedings of the Thirtieth Annual Hawaii International Conference on System Sciences*, January 7-10, Maui, Hawaii, vol. 2, pp. 350-358, 1997.

[4] Gary Cornell and Cay S. Horstmann, *Core JAVA*, SunSoft Press, California, 1996.

[5] Flaviu Cristian, "Synchronous and Asynchronous Group Communication", *Communications of the ACM*, vol. 39, no. 4, pp. 88-97, 1996.

[6] John Desborough, *INTRANET Web Development*, New Riders, Indiana, 1996.

[7] C. Ellis, S. J. Gibbs, and G. L. Rein, "Groupware: Some issues and experiences", *Communications of the ACM*, vol. 34, no. 1, pp. 38-58, 1991.

[8] C. Ellis and G. Nutt, "Modeling and Enactment of Workflow Systems", in *Proceedings of 14th International Conference on Application and Theory of Petri Nets*, June 21, Chicago, Illinois, pp. 1-16, 1993.

[9] J. W. Erkes, K. B. Kenny, J. W. Lewis, B. D. Sarachan, M. W. Sobolewski, and R. N. Sum, Jr., "Implementing Shared Manufacturing Services on the World-Wide Web", *Communications of the ACM*, vol. 39, no. 2, pp. 34-45, 1996.

[10] Michael R. Genesereth and Steven P. Ketchpel, "Software Agent", *Communications of the ACM*, vol. 37, no. 7, pp. 48-53, 1994.

[11] Martin Gleeson and Tina Westway, "Beyond Hypertext: Using the WWW for

Interactive Applications", in *Proceedings of The First Australian World-Wide Web Conference*, April 30 to May 2, Ballina, Australia, 1995.

[12] A. Goldberg and D. Robson, *Smalltalk-80 The Language and Its Implementation*, Reading, MA:Addison-Wesley, 1983.

[13] Kevin J. Goodman, *Windows NT: A Developer's Guide*, M&T Books, New York, 1994.

[14] Martin Hardwick, David L. Spooner, Tom Rando, and K. C. Morris, "Sharing Manufacturing Information in Virtual Enterprises", *Communications of the ACM*, vol. 39, no. 2, pp. 46-54, 1996.

[15] Robert M. Hinden, "IP Next Generation Overview", *Communications of the ACM*, vol. 39, no. 6, pp. 61-71, 1996.

[16] Soon-Young Huh and David A. Rosenberg, "A Change Management Framework: Dependency Maintenance and Change Notification", *Journal of Systems Software*, vol. 34, no. 3, pp. 231-246, 1996.

[17] Norman P. Hummon, "Organizational Structures and Exchange Processes", *Intelligent Systems in Accounting, Finance and Management*, vol. 2, no. 4, pp. 235-246, 1993.

[18] Charles Lamb, Gordon Landis, Jack Orenstein, and Dan Weinreb, "The ObjectStore Database System", *Communications of the ACM*, vol. 34, no. 10, pp. 50-63, 1991.

[19] Gavin McMahon, *Hyper Media Case Study: Netscape Communications Corporation*, THESEUS Institute and London Business School, 1995.

[20] Piero Migliarese and Emilio Paolucci, "Improved communications and collaborations among tasks induced by Groupware", *Decision Support Systems*, vol. 14, no. 3, pp. 237-250, 1995.

[21] Arturo Molina, Ahmed H. Al-Ashaab, Timothy I. A. Ellis, Robert I. M. Young, and Robert Bell, "A Review of Computer-Aided Simultaneous Engineering Systems", *Research in Engineering Design*, vol. 7, no. 1, pp. 38-63, 1995.

[22] Krishnamurty Muralidhar, Dinesh Batra, and Peeter J. Kirs, "Accessibility, Security, and Accuracy in Statistical Databases: The Case for the Multiplicative Fixed Data Perturbation Approach", *Management Science*, vol. 41, no. 9, pp. 1549-1564, 1995.

[23] Netscape Communications Cor., URL *http://www.netscape.com*, 1996.

[24] Adrian Nye, *Xlib Programming Manual for Version 11 of the X Window System*, CA:O' Reilly & Associates, Inc., vol. 1, 1992.

[25] Daniel E. O' Leary, Daniel Kuokka, and Robert Plant, "Artificial Intelligence and Virtual Organizations", *Communications of the ACM*, vol. 40, no. 1, pp. 52-59, 1997.

[26] O'Reilly & Associates, Inc., URL *http://website.ora.com*, 1996.

[27] Charles J. Petrie, "Agent-Based Engineering, the Web, and Intelligence", *IEEE Expert*, vol. 11, no. 6, pp. 24-29, 1996.

[28] Richard W. Quadrel, Robert F. Woodbury, Steven J. Fenves, and Sarosh N. Talukdar, "Controlling Asynchronous Team Design Environments by Simulated Annealing", *Research in Engineering Design*, vol. 5, no. 2, pp. 88-104, 1993.

[29] Andreas Scherer, "Supporting Concurrent Engineering Using an Intranet Approach", in *Proceedings of the Thirtieth Annual Hawaii International Conference on System Sciences*, January 7-10, Maui, Hawaii, vol. 2, pp. 589-595, 1997.

[30] Y.-P. Shan, "An Event-Driven Model-View-Controller Framework for Smalltalk", in *Proceedings of Object-Oriented Programming Systems, Languages, and Applications : OOPSLA*, October 1-6, New Orleans, Louisiana, pp. 347-352, 1989.

[31] HongHai Shen and Prasun Dewan, “Access Control for Collaborative Environments”, in *Proceedings of ACM 1992 Conference on Computer-Supported Cooperative Work*, October 31 to November 4, Toronto, Canada, pp. 51-58, New York, 1992.

[32] B. Stroustrup, *The C++ programming language*, MA:Addison-Wesley, 1986.

[33] Oliver E. Theel and Brett D. Fleisch, “A Dynamic Coherence Protocol for Distributed Shared Memory Enforcing High Data Availability at Low Costs”, *IEEE Transactions on Parallel and Distributed Systems*, vol. 7, no. 9, pp. 915-930, 1996.

[34] Vijay Varadharajan and Claudio Calvelli, “An Access Control Model and Its Use in Representing Mental Health Application Access Policy”, *IEEE Transactions on Knowledge and Data Engineering*, vol. 8, no. 1, pp. 81-95, 1996.

[35] Kwok-Kee Wei, Bernard C Y Tan, Choon-Ling Sia, and Krishnamurthy S Raman, “Hypertext: A New Approach to Construct Group Support Systems”, *International Journal of Information Management*, vol. 16, no. 3, pp. 163-181, 1996.

[36] Michael Wooldridge and Nicholas R. Jennings, “Intelligent Agents: Theory and Practice”, *Knowledge Engineering Review*, vol. 10, no. 2, pp. 115-152, 1995.

[37] Second World Wide Web Conference ’94 *Electronic Proceedings of the Second World Wide Web Conference ’94: Mosaic and the Web*, October 17-20, Chicago, Illinois, 1994.

[38] Kun-Lung Wu and W. Kent Fuchs, “Recoverable Distributed Shared Virtual Memory”, *IEEE Transactions on Computers*, vol. 39, no. 4, pp. 460-469, 1990.

[39] Budi Yuwono and Dik Lun Lee, “WISE: A World Wide Web Resource Database System”, *IEEE Transactions on Knowledge and Data Engineering*, vol. 8, no. 4, pp. 548-554, 1996.