Structured Cooperative Authoring on the World Wide Web

Dominique Decouchant, Vincent Quint, Manuel Romero Salcedo

Abstract

In this paper we present Alliance, a groupware application that allows several users located on different Web sites to cooperatively produce documents in a structured way. In addition to the local editing functions made available on each site by a structured editor, the application provides such basic functionalities as management of document storage and remote access to distributed documents. It offers services for handling user interaction and cooperation, for dynamically distributing roles to users, for showing documents through multiple views, for controlling the consistency of modifications, and for updating all copies of shared documents. ***Keywords:*** *Cooperative authoring, structured documents, group awareness, computer-supported cooperative work, World Wide Web, HTML*

Introduction

From its inception, the World Wide Web has been perceived as a system that allows people to cooperate and exchange information through a wide area network. This is achieved by several multimedia documents written by different users and made available to others on distributed servers. Each document contains links to some related documents, located on other sites.

In this paper, we take a slightly different approach to cooperation. We consider several persons involved in the task of writing a single document in a cooperative way. Each person has a specific role to play in the common task and should benefit from a software tool. The goal is to finally produce a large, well-structured document, rather than writing several pieces of information interconnected by various links. Since we consider some aspects of the problem to be very close to those considered in the World Wide Web, many solutions developed for the Web can be applied to cooperative editing. For studying the problems posed by cooperative editing, we have developed a distributed application, Alliance, which is a structured distributed cooperative editing tool built on top of the Grif editor [5].

Alliance has been designed to allow several users, distributed on a network, to work on shared structured documents. After the first version was developed for a local area network, it was recently adapted to the World Wide Web [3]. The aim of developing Alliance on the Web was to make this application more widely available, by using long-haul networks and allowing loosely coupled cooperation. For reaching this goal, evolutions of the Alliance cooperation services were needed to take into account most of the problems posed by the Internet and to obtain acceptable performances. From the implementation point of view, the document-management layer, initially based on NFS (Network File System), has been extended to the services provided by HTTP (HyperText Transfer Protocol) [1] on the Web.

The rest of this paper is divided into two main parts. The first part presents the basic principles on which the application is based. The application puts the emphasis on user roles, document fragmentation, and software architecture. The second part considers the specific problems posed by a wide area network for this kind of application and it shows how the Web technology has been used for solving these problems.

The Alliance Application

When developing Alliance, the goals were to study and better understand the specific problems of cooperative editing and to develop techniques that would allow complex structured documents to be handled more efficiently in a collaborative distributed environment.

The application has been developed for networked UNIX workstations. It controls several local instances of the Grif editor, running on different workstations, and allows their users to cooperate. In addition to the local editing functions inherited from Grif, it handles document storage and access to distributed documents; it offers high-level services for handling user interaction and cooperation, for defining shared document fragments, for distributing roles to users, for supporting group awareness, for updating copies of shared documents, and for controlling document consistency.

User Roles and Document Fragments

In the development of a large documentation project, the way in which people interact with each other is well defined: the work is organized, and each team member has a different role to play in the project. Therefore a notion of role has been introduced in Alliance. For each part of a document, a user may have one of the four available roles:

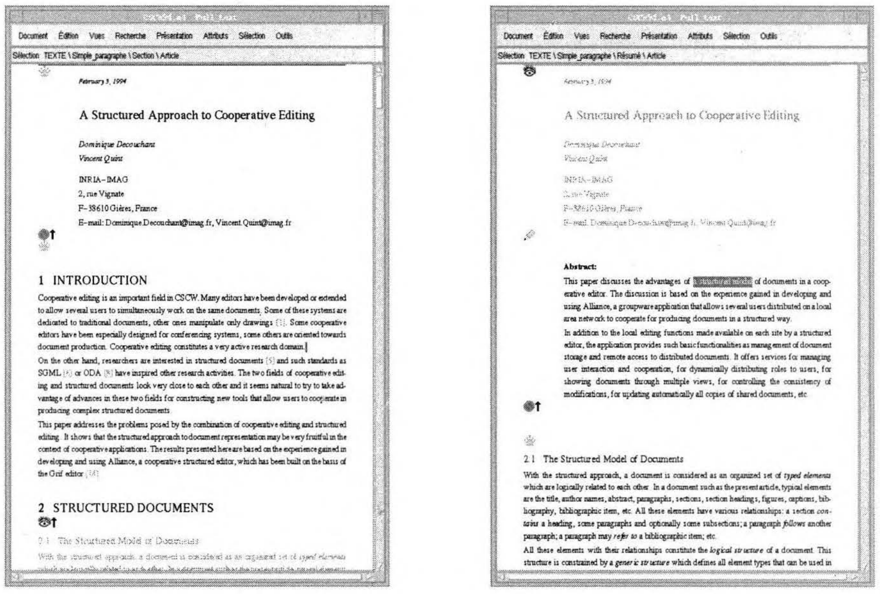
- As the purpose of the application is to edit documents, the first role that appears to be necessary is the *writer* role. A writer is a user who can modify (i.e. change the structure or the content) of (a part of) a document.
- The second role is the one of a *reader*. A reader is a user who can see and read a part of a document, but who cannot modify it. The writer role includes the reader role.
- A third role, called *null role*, has been introduced for preventing some users from even seeing some parts of a document that are confidential for them. Obviously, a user with that role cannot modify the document either.
- The last role, called *manager*, is provided for people who give to other people the possibility of playing roles on documents. This manager role allows the manager to assign roles to new users, and to change the role of existing users. The operations allowed by this role include those allowed to a writer. Several users can play the manager role for a document or a part of a document.

The same user may have different roles on different parts of a document. He can then be authorized to modify some parts of the document, to only read some other parts, and to not even see the rest. These document parts for which user play different roles are called *fragments*.

As an example, Figure 1 shows the displays seen by two users (A and B) editing a shared document. This document is composed of four fragments. The first fragment contains the title, the author names, and affiliations. The second fragment contains the abstract. The third fragment includes the introduction and the heading of section 2. The fourth and last fragment contains all remaining parts. Fragments limits are represented in both views by various icons, which also indicate the current role played by the user on each fragment.

The content of the second fragment (the abstract) only appears in the view presented to user B who plays a writer role for that fragment. This fragment is accessed with the null role by user A (information is invisible). Conversely, the third fragment can only be seen by user A. The fourth fragment can be written by user B, who can also act as a manager, but the fragment can only be read by user A.

As a user may have different roles for different parts of a document, and as these roles can change, the user needs to be notified of the role he can really play on each part. This is achieved by using different colors and by inserting special



Document Édition Vues Recherche Présentation Attributs Sélection Outils

Sélection TEXTE \ Simple_paragraphe \ Section \ Article

February 3, 1994

A Structured Approach to Cooperative Editing

Dominique Decouchant
Vincent Quint

INRIA–IMAG
2, rue Vignate
F–38610 Gières, France
E–mail: Dominique.Decouchant@imag.fr, Vincent.Quint@imag.fr

1 INTRODUCTION

Cooperative editing is an important field in CSCW. Many editors have been developed or extended to allow several users to simultaneously work on the same documents. Some of these systems are dedicated to traditional documents, other ones manipulate only drawings [1]. Some cooperative editors have been especially designed for conferencing systems, some others are oriented towards document production. Cooperative editing constitutes a very active research domain.

On the other hand, researchers are interested in structured documents [5] and such standards as SGML [6] or ODA [8] have inspired other research activities. The two fields of cooperative editing and structured documents look very close to each other and it seems natural to try to take advantage of advances in these two fields for constructing new tools that allow users to cooperate in producing complex structured documents.

This paper addresses the problems posed by the combination of cooperative editing and structured editing. It shows that the structured approach to document representation may be very fruitful in the context of cooperative applications. The results presented here are based on the experience gained in developing and using Alliance, a cooperative structured editor, which has been built on the basis of the Grif editor [14].

2 STRUCTURED DOCUMENTS

2.1 The Structured Model of Documents

With the structured approach, a document is considered as an organized set of *typed elements*

Document Édition Vues Recherche Présentation Attributs Sélection Outils

Sélection TEXTE \ Simple_paragraphe \ Résumé \ Article

February 3, 1994

A Structured Approach to Cooperative Editing

Dominique Decouchant
Vincent Quint

INRIA–IMAG
2, rue Vignate
F–38610 Gières, France
E–mail: Dominique.Decouchant@imag.fr, Vincent.Quint@imag.fr

Abstract:

This paper discusses the advantages of structured model of documents in a cooperative editor. The discussion is based on the experience gained in developing and using Alliance, a groupware application that allows several users distributed on a local area network to cooperate for producing documents in a structured way.

In addition to the local editing functions made available on each site by a structured editor, the application provides such basic functionalities as management of document storage and remote access to distributed documents. It offers services for managing user interaction and cooperation, for dynamically distributing roles to users, for showing documents through multiple views, for controlling the consistency of modifications, for updating automatically all copies of shared documents, etc.

2.1 The Structured Model of Documents

With the structured approach, a document is considered as an organized set of *typed elements* which are logically related to each other. In a document such as the present article, typical elements are the title, author names, abstract, paragraphs, sections, section headings, figures, captions, bibliography, bibliographic item, etc. All these elements have various relationships: a section *contains* a heading, some paragraphs and optionally some subsections; a paragraph *follows* another paragraph; a paragraph may *refer to* a bibliographic item; etc.

All these elements with their relationships constitute the *logical structure* of a document. This structure is constrained by a *generic structure* which defines all element types that can be used in

Figure 1: Two views of the same document

active icons at each fragment limit. When the user has a null role, the corresponding parts are not displayed, but the user may notice that they exist, as numbers (section numbers, figure numbers, etc.) take hidden parts into account. Parts where the user has the reader role are displayed with a specific color that each user can choose. Parts where the user has a writer or manager role are displayed with their normal color.

Roles allocated by a manager to other users are *potential* roles: users are allowed to play these roles, but they are not guaranteed to be able to play them at any time. When acting on a shared document, a user plays *effective* roles, which may be different from his/her potential roles. This is due to two different causes:

- For ensuring document consistency, only one user can play the writer or manager effective role at a time. Other users owning the writer potential role are then limited to a reader effective role. This policy is applied by the application and cannot be bypassed by users.
- For improving cooperation, a user with the writer potential role may decide to play only the reader or null effective role on some fragments that he/she does not intend to modify for some time.

When a user changes his/her role, when a user leaves the session, or when a manager allocates new roles, the roles of other users are subject to changes. This is the reason why the icons marking fragment limits are useful and may change dynamically.

Alliance is an *asynchronous* application: all users do not see exactly the same state of the document at the same time. When a writer types a single character, this character is not displayed immediately on the screen of other users. Instead, each author must validate the changes he/she has made, in order to make them available to other users. The other users may decide whether they want these changes to be displayed automatically on their screen at validation time or later. In the second case, the icons indicate that a new version of the corresponding fragment is available and the user simply clicks the icon to get the latest version of that fragment.

The document is automatically divided into fragments by the application, according to the roles assigned by managers. This division is performed in such a way that the potential role of each user having a role for a given fragment does not change along the fragment. Changing potential roles of users can then lead to a different fragmentation of a document: fragments can be divided or merged by the application. Each fragment is stored in a separate file and these files can be located on different sites.

Document fragmentation is based on document structure. Being based on Grif, Alliance uses a structured model of documents. These documents can include a variety of components such as tables and equations, which can be shared in the same way as the rest of the document. The rich logical structure allows managers to handle document sharing efficiently. It also allows to dynamically change sharing granularity, thus permitting users to change role easily. As sharing is based on a well defined structure, document consistency can be guaranteed by simple ways.

Alliance Architecture

Basically, Alliance allows each user to edit locally an instance of a shared document and it allows all users sharing a document to communicate in order to be aware of the work performed by others. Therefore, two main functions can be identified in this application:

- *The editing function* allows users to process documents locally, in the core memory of their workstation,
- *The document-management function* provides support to the editing function for document naming, document storage, concurrency control, document replication, and group awareness.

The distributed application is constituted by several *instances*. An instance of Alliance is the piece of software that runs on a workstation for a single user. If several users share the same workstation, then several independent Alliance instances run on it.

In the LAN version of Alliance, an instance is a single process. As shown in Figure 2, it may be viewed as a main module that controls the functions performed by other modules.

The main module controls the editing module via the Grif API. It can update a fragment for which a new version has been sent from another instance; it can ask the editor to store a local fragment in a file; it can ask the editor to prevent the user from modifying some parts of the edited document; etc.

Editing events are transmitted via the ECF (External Call Facility) by the editing module, to inform the main module about some commands performed by the user. For instance, that is the way the main module knows what part of the document is currently selected, when the user playing the manager role changes the roles of other users for the selected part. More details about the API and the ECF mechanism provided by Grif are given in related documentation [6].

The main module also receives events (callbacks) from the user interface module (OSF/Motif) when the user issues commands that concern not editing, but cooperation, such as changing the role of other users.

Finally, the main module calls the document-management module when it needs to access the fragments stored in local or remote files.

Cooperative Editing on a Wide Area Network

In this section, we present the main issues that have been addressed when extending the LAN

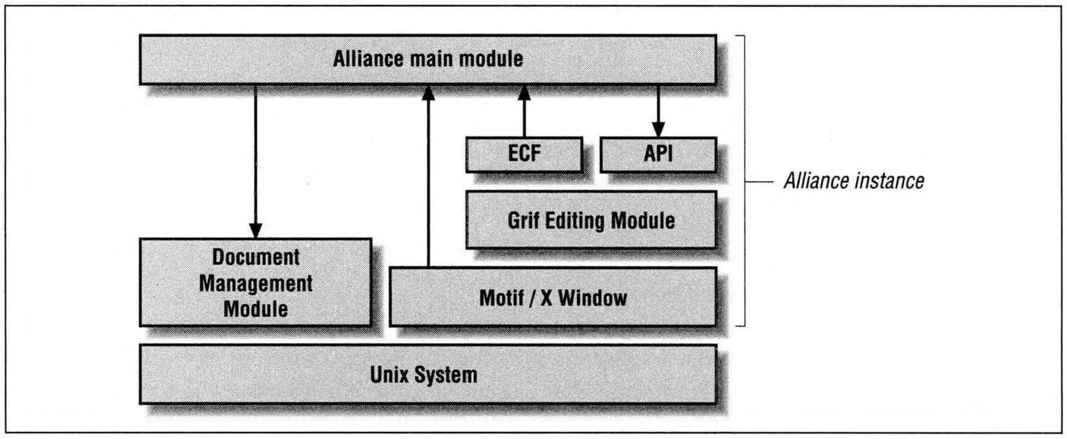


Figure 2: Architecture of an Alliance instance

version of Alliance to the Web. We focus first on the problems posed by the delays and failures that can occur in wide area networks and we discuss the impact of these problems on the architecture. Then, we discuss the issue of user identification and management in the the context of the Internet.

Finally, we describe the document storage mechanism that allows each user to work on shared documents when all sites can communicate and exchange information, but also when some users are isolated by network failures or simply because they are away.

From a LAN to the Internet

Porting Alliance to a wide area network would, one must think, allow users to benefit from the specific features of such networks. A WAN, such as the Internet, is composed of many sub-networks that communicate via gateways. This kind of network is inherently unreliable because of unpredictable communication delays, link failures, or computer crashes. In those cases, the network can be temporarily partitioned leading to disconnection of subnetworks. For an application such as Alliance, these failures make some remote-files unreachable and transmission delays unpredictable. All actions involving remote resources can cause long delays or even locks, when these remote resources are treated as if they were local. Therefore the communication support of Alliance has been separated from the part that is in charge of user interface and local editing.

Another issue in a wide area network is remote-file access. We first considered a simple solution, which would avoid making changes to the way in which files are accessed. The same service as NFS is offered in WANs by other systems, such as the Andrew File System (AFS) for instance, which hide distribution and present remote files in the same way as local files. Although this solution would not change the architecture, it was rejected—we were looking for a file system that was widely available and used; one that would make the installation of Alliance on any site a simple task, even when used occasionally.

The World Wide Web satisfies these conditions. A number of servers run on the Web provide remote file access. A Web server can be easily installed on any computer and the software is widely available.

Web servers are based on the HTTP protocol [1]. HTTP allows on-line access to distant information using the client/server model. In the WAN version, Alliance fully exploits this model for allowing instances of the application to cooperate:

each instance acts as a client and each site acts as a server. Clients send requests to servers in order to:

- Obtain information about shared documents (list of users, effective role played by a user in a fragment, etc.)
- Get a new version of a fragment

On the server site, the work is carried out through the Common Gateway Interface, by executing scripts. Basically, a script is a program that can be executed for a HTTP client to perform some specific work on a server site. A script can receive information through input parameters and it can send results back to the calling client. Alliance uses a set of scripts that allow clients to get information about shared documents, or about fragments.

HTTP proposes three methods: Get, Post and Put. Get allows a client to request a document identified by its URL. Post is used to trigger execution of an existing script; a Post request is composed of the script URL and a list of input parameters to be provided to the script. Put is supposed to allow a client to write a file remotely, but very few servers implement this method, mainly for security reasons.

As the Put method is not widely available, a client must combine two methods for writing a file remotely (see Figure 3). It sends to the server a Post which starts a script sending back a Get message to the client. Then the client returns the file as the result of the Get message and the script that issued the Get message writes this file locally on the server. Due to the lack of the Put method, a HTTP server is also required on the client site.

The following example explains how a client can obtain the list of documents owned by a remote user. First, the client builds a request containing the URL of the script GetUserDocumentList, and the login name of the remote UNIX user as parameters. Then, it invokes the HTTP Post method and the script is executed on the server site. The script first checks that the user is registered on the server, then it loads the list of documents belonging to that user. Finally, the list is returned to the client through the standard output. On error, the script returns an appropriate diagnostic.

Executing scripts on a remote server poses several problems concerning message security and authentication of users and applications. These security issues are not addressed here, but they are considered for a future version of Alliance.

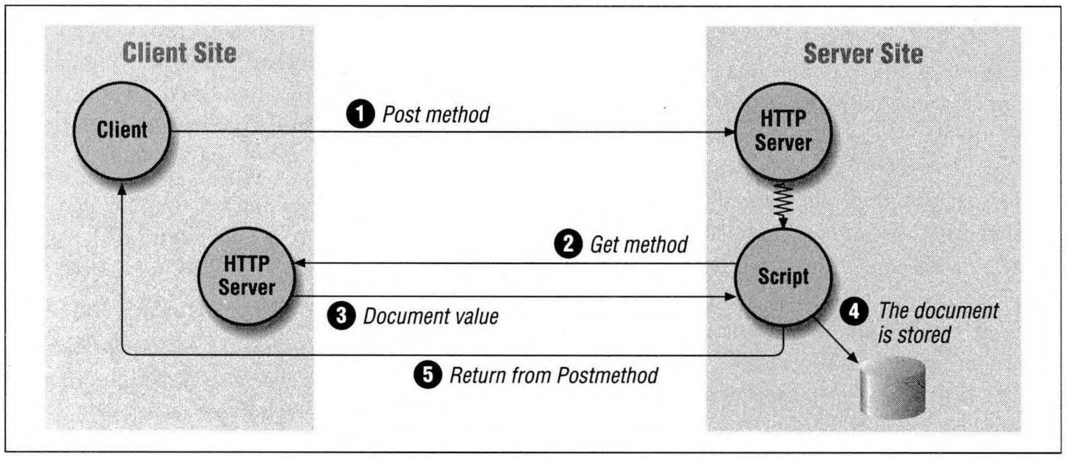


Figure 3: Implementation of the Put method

In order to cope with the specific constraints of the Internet, each instance of Alliance is now divided into two processes (see Figure 4), the Editor and the Assistant:

- All editing functions for which the user is expecting an immediate feedback, and that do not depend on remote resources, are performed by the Editor.
- All functions that can cause transmission delays or locks are performed by the Assistant. These functions are almost the same as those performed by the document-management module in the LAN version. The Assistant takes care of all functions regarding the client side of remote access.

Each user on a site is served by an Editor process and an Assistant process. In addition to these processes, which are in charge of the local users, each Alliance site runs a daemon, even if no local user is active. This daemon process is a HTTP server that contains the Alliance scripts implementing all operations required by the remote Alliance Assistants. Each Alliance script is executed in response to an Alliance client request.

User Management

The notion of a user is fundamental in a groupware application. Some applications rely on the notion defined by the host operating system. These users are considered as persons who can login to make actions with well specified privileges on computer resources (disks, CPU, files, display, applications, drivers, etc.). Such a user definition is not adapted to the shared environment of distributed groupware applications, as it does not consider remote users, nor the specific resources of the application.

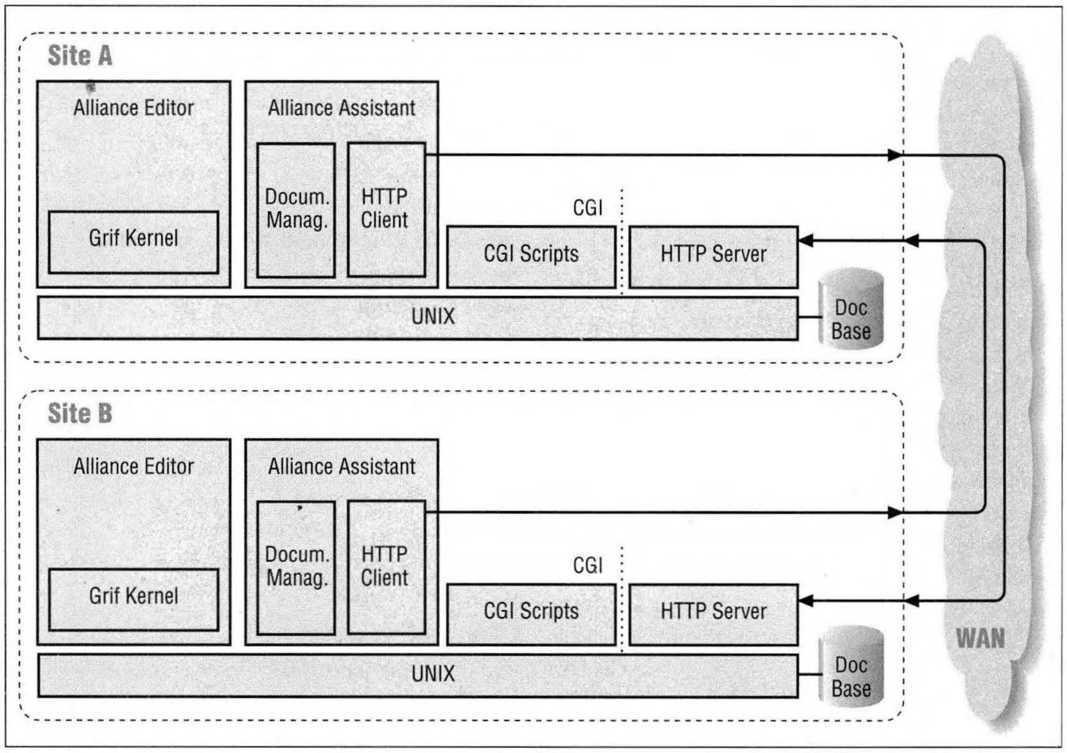


Figure 4: Architecture for a wide area network

In Alliance, a user is simply a name (called "external name" below) that identifies a person. A list of users is associated with each document. This list is built by the document owner and is independent of the lists associated with other documents. The list indicates the users who are authorized to act on the document. The document owner can update this list at any time.

The document owner is the user who creates the document or makes it available in the Alliance environment. Initially, the owner plays the role of a manager for the whole document. He/she is free to assign various roles to other users.

A notion of group is also used in Alliance. A group is a name that represents several users playing the same role on documents. A user may be a member of several groups. A group can appear in the list associated with a document. It indicates that all users belonging to that group are allowed to act on the document.

All users of the list associated with a document can be involved in the commands for sharing documents. They will also be involved in future tools which are under development for negotiation, messaging or annotation. For instance, the document-sharing tool used by managers allocates roles to the users and groups according to the list associated with the document.

An Alliance user must be registered in the host system of one site involved in the application. This site contains the information needed for managing all documents belonging to that user. As the scope of a user is restricted to the host system where it is registered, users have not only an external name, but also an internal name that contains both the identification of their site and their login name on that site. The user list associated with each document contains both the external name and the internal name for each user.

When the document owner creates or updates the list of users, he provides the following data for each entry:

- The URL [2] (Universal Resource Locator) or the machine name (in a local area network) that identifies the site of that user
- The login name of the user on that site
- The directory where the document base is located on that site (user home directory by default)

After the list has been initialized, the owner sends a message to all users working with the document and gives them his/her URL, his/her login name and the local document name. Each remote user can then get a full copy of this document, and know the associated users and groups. Thus, all users are able to contact each other, even if they meet for the first time.

Document Distribution and Replication

In Alliance, a document is represented by a set of files which contain: document fragments, user roles for each fragment, the order of all fragments in the document, and the current state of each fragment.

In order to allow each user to work on a shared document, even in case of network failure, documents are copied on each site where they are needed. All document fragments and the corresponding management information are replicated among different sites (see Figure 5), where they are stored in local files. Each user can then work independently. In order to allow cooperation and group awareness, local copies must be updated when remote users have made modifications. But, as Alliance is based on an asynchronous model of cooperation, these updates do not need to be done in real time. Nevertheless, a mechanism is required for maintaining the consistency of all those copies.

Document consistency is based on a simple principle. In the whole system, there always exists one *master copy* for each fragment, which is the reference; there are as many *slave copies* as needed. On a given site where at least one user works on the document, all fragments of that document are available in local files and each fragment copy is either the master copy for that fragment or a slave copy. These two types of copies allow different effective roles to be played by a local user:

- The master copy (in grey in Figure 5) allows the user to act on it with any role, including the writer or manager role. As the master copy is unique, only one user can play these roles for a given fragment.
- A slave copy only allows the reader or null roles, even if a higher potential role is assigned to the user.

According to this principle, the set of all master copies constitutes the current document state. As a site usually does not own the master copies of all fragments of a given documents, each site, and then each user, may have a delayed perception of the document state. However, these different perceptions of the document are updated when a site owning the master copy of a fragment produces a new version of that fragment.

Master copies can migrate from one site to another. This operation is based on a transaction mechanism in order to avoid loss or duplication of the master copy that could be caused by communication failures. The possibility of moving the master copy of a fragment is evaluated each time a user tries to act on a fragment with the writer or manager role, in accordance with his potential role. The transaction fails if the master copy is unreachable or is already locked in reader or manager mode by its current owner. If no user is currently playing the writer or manager role on the site of the master copy, or if the user playing these roles accepts to change role, the master copy can move.

When a fragment is updated by a user acting as a writer, a copy of this fragment is not automatically sent to all sites working on the document. Only a short message is sent: it informs the remote sites that an updated version of the fragment is available. With the replication policy presented above, communication between sites is needed only to transmit these short messages, to transfer updated copies to the sites which ask for it, and to get remote user lists.

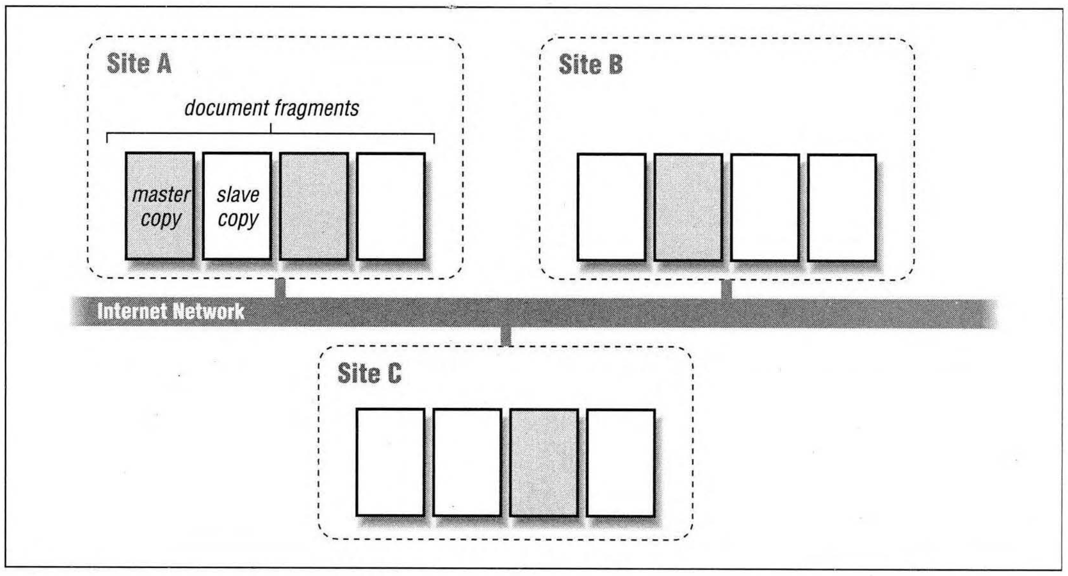


Figure 5: Distribution of the fragment copies of a document

An advantage of document replication is that it allows disconnected cooperative work. We might think for instance of a user accessing the network and working in a collaborative way on a document. When he/she leaves his/her office, he/she disconnects his/her workstation and continues to work in disconnected mode, with the full editing environment. When the station is connected again to the network, Alliance automatically updates all document fragments. This kind of operation mode is the result of an intended disconnection. In case of any network failure, users can work on their documents in the same way. Obviously, disconnection should not last too long.

Conclusion and Perspectives

The current implementation of Alliance includes all functions presented in this paper. Additional services are under development for helping users to communicate with each other through an annotation mechanism and additional dialog options. Annotations will permit users to associate some comments with the document parts for which they have only the reader role. Private and shared annotations will be available.

The wide-area version of Alliance uses only one part of the services provided by the Web, the HTTP protocol, and servers. It does not take advantage of all the possibilities offered by the World Wide Web. For the moment, document fragments are stored in a specific format, not in HTML [4], which is the usual document format on the Web. The next version of Alliance will also integrate HTML. In fact, this work is in progress. The Grif editor has already been adapted to the Web and the result of this adaptation, Symposia, obviously handles HTML documents [7]. Alliance and Symposia are converging in order to provide Web users with a collaborative tool for producing HTML documents on the Web. ■

References

1. T. Berners-Lee, *Hypertext Transfer Protocol: A Stateless Search, Retrieve and Manipulation Protocol,* Internet draft, CERN, November 1993.
2. T. Berners-Lee, *Uniform Resource Locators—A unifying syntax for the expression of names and addresses of objects on the network,* Internet draft, CERN, January 1994.
3. T. Berners-Lee, R. Cailliau, A. Luotonen, H. Frystyk Nielsen and A. Secret, "The World Wide Web," *Communications of the ACM,* vol. 37, num. 8, pp. 76-82, August 1994.
4. T. Berners-Lee, *Hypertext Markup Language Specification—2.0,* Internet draft, CERN, November 1994.
5. V. Quint, I. Vatton, "Grif: an Interactive System for Structured Document Manipulation," *Text Processing and Document Manipulation, Proceedings of the International Conference,* J. C. van Vliet, ed., pp. 200-213, Cambridge University Press, 1986.
6. V. Quint, I. Vatton, "Making Structured Documents Active," *Electronic Publishing—Origination, Dissemination and Design,* vol. 7, num. 1, pp. 55-74, March 1994.
7. V. Quint, C. Roisin, I. Vatton, "A structured authoring environment for the World Wide Web," *Proceedings of the Third International World Wide Conference,* Computer Networks and ISDN systems, ed., pp. 831-840, The International Journal of Computer and Telecommunications Networking, vol. 27, April 1995.

About the Authors

Dominique Decouchant
CNRS-IMAG
Dominique.Decouchant@imag.fr

Vincent Quint
INRIA RhÔne-Alpes
Vincent.Quint@inria.fr

Manuel Romero Salcedo
INRIA RhÔne-Alpes / IMAG
Manuel.Romero@imag.fr