# Building Groupwares over Duplicated Object Systems

Hechmi Khlifi[1], Jocelyn Desbiens[1], and Mohamed Cheriet[2]

[1] INRS–Télécommunications, Université du Québec, Place Bonaventure, 900, de la Gauchetière, Ouest, Niveau C, C.P. 644, Montréal (Québec) Canada, H5A 1C6, {khlifi,desbiens}@inrs-telecom.uquebec.ca

[2] École de Technologie Supérieure, Université du Québec, 1100 rue Notre-Dame, Ouest, Montréal (Québec) Canada, H3C 1K3, mohamed.cheriet@etsmtl.ca

**Abstract.** Groupware toolkits let developers build applications for synchronous and distributed computer-based conferencing. Duplicated object systems[1] (or *DoS*), on the other hand, manage distributed objects over the Internet and, since they include high-level features such as communication facilities, load balancing, fault tolerance, and hierarchical messaging, may be leveraged as building blocks for rapidly developing groupware toolkits. This paper describes how we built such a groupware starting from a particular DoS. The system contains a run-time architecture that automatically manages the creation, interconnection, and communications of the distributed processes involved in the working sessions, a set of groupware facilities allowing users to collaborate, to take action on state changes, and to share relevant data, and a session management and user control mechanisms accommodating the group's working style.

## 1 Introduction

Building groupware for synchronous, distributed conferencing can be a frustrating experience. If only conventional single-user GUI toolkits are available, implementing even the simplest systems can be lengthy and error-prone. A programmer must spend much time on tedious but highly technical house-keeping tasks, and must recreate interface components to work in a multi-user setting. Aside from the normal load of developing a robust application, the programmer of groupware must also attend to the setup and management of distributed processes, inter-process communication, state management and process synchronization, design of groupware widgets, creation of session managers, concurrency control, security, and so on.

Consequently, a variety of researchers have been exploring groupware toolkits [9]. Their purpose is to provide tools and infrastructures powerful enough

[1] Object duplication is a distributed object paradigm created and put forward by Quazal Inc., a Montréal based company where one of the author worked for one and a half year.

J.M. Haake and J.A. Pino (Eds.): CRIWG 2002, LNCS 2440, pp. 245–254, 2002.

to let a programmer develop robust, high quality groupware with reasonable effort. Some inroads have been made, but we are far from a complete solution. Realistically, most of today's groupware toolkits are best seen as breakthrough research systems used to either explore particular architectural features of groupware toolkits, or as platforms to build experimental groupware prototypes. While they have not reached the maturity of single-user GUI toolkits, these pioneering efforts have laid a foundation for the next generation of toolkit design.

The solution we propose to this problem is to build the groupware upon a DoS. The DoS handles all the low-level technical network details. This allow us to construct real-time distributed multi-point groupware applications, where two or more people in different locations would be able to visually share and manipulate their online work. Typical applications produced by this system would be electronic whiteboards, games, multi-user text and graphics editors, distributed presentation software, textual chat systems, and so on.

In the second section of this paper we define and describe the principals features of Duplicated Object Systems. In the third section, we describe how *Synchromedia* [2], a distance-learning system, has been built using a DoS.

## 2 Duplicated Object System

### 2.1 DoS: Overview

A *Duplicated Object System*[3] is a distributed object architecture where information is "pushed" across the network rather than using the less scalable "pull" approach. The basic philosophy is that a collaborative application consists of a collection of objects which need to be distributed and duplicated across the stations participating in the session. Furthermore, the object content needs to be coherent and upon modification propagated to other stations.

In a collaborative application, shared objects need to be duplicated across the network so that the participants may see each other. Object duplication is the mechanism used by a DoS to achieve this. The use of duplicated objects gives a DoS significant advantages over other technologies when it comes to the features offered and the ease of programming as they allow high level features such as fault tolerance, load balancing, and object migration to be presented to the user in an easy to use manner.

A duplicated object has the status of either *duplication master* or *duplica*. The duplication master of an object is the controlling instance of the object, while its duplicas are copies of the master object. If a participant joins a session that is already in progress, duplicas of the existing duplicated objects that the new participant requires will be automatically created on that participant's station. The object duplication model used by a DoS allows the programmer to define where objects are required to be duplicated via several different implementations, as detailed below.

[2] Synchromedia: Tele-education and tele-research global university. École de Technologie de l'Information. Université du Québec.

[3] Or DoS for short.

### 2.2 Object Duplication

To enable participants to see each other, duplicated objects need to be duplicated to those stations that require the object. However, to enable a session to be scaled and to operate efficiently, each station connected to a session should only have copies of the objects that it really needs. This ensures that the use of resources such as CPU and bandwidth are minimized, which allows for a greater number of simultaneously connected users. In a DoS, there are possibly four mechanisms used to duplicate objects to the stations where they are required: a duplication master may create a duplica on a particular station, a station may fetch a duplica of a specific duplication master, an object may be automatically duplicated to all participating stations, or duplication spaces may be used to dictate where objects should be duplicated to. The flexibility of the duplication model allows the programmer to use any combination of these four mechanisms to control the duplication of objects across the stations connected to the session. This gives the programmer the ability to decide how many duplicas of each object are published and to which stations so that objects are only duplicated to the stations where they are needed. The level of control that the developer has also facilitates the optimization of resource usage, such as the available memory and bandwidth.

If an object is required to be known globally, the easiest way to be certain that it is duplicated to all stations participating in the session is to ensure that the user-defined class inherits from the appropriate DoS class. This will guarantee that the object will be duplicated to each station connected to the session. If an object is not required to be known globally then the programmer must dictate the conditions under which an object is duplicated. This can be done by simply directly calling the appropriate method to create a duplica on another station or to fetch a duplica from another station. In certain situations, this may be the simplest manner in which to duplicate objects. However, if objects are simultaneously created, migrated, or deleted, due to operations such as fault tolerance and load balancing, object duplication can become very complex and the implementation of duplication spaces will usually present an easier solution to the management of object duplication.

## 3 Building a Groupware over a DoS: Synchromedia

*Synchromedia* is a distance-learning project involving four universities and about ten researchers. It is made up of a collection of collaborative tools (whiteboard, chat, audio/video, shared directories, and virtual laboratory) allowing students to work together and to interact as if they were in a real classroom.

*Synchromedia* is a typical example of a groupware made from a DoS. The Dos we used is **NetZ** [7]. Figure 1 is a snapshot of the user's interface.

### 3.1 Communications in Synchromedia

Communication across a large network presents a number of difficulties that need to be overcome in order to produce an efficient and operable system. The types

**Fig. 1.** User's interface

of problems encountered are the numerous combinations of connection types and processing power that each station may have which leads to non-uniform station capacities and available bandwidth. In addition, the variety of transport types means that stations with no common transport cannot communicate directly with each other. In the case of *Synchromedia*, the DoS communication model deals with these problems thus relieving the programmer of these complex issues.

To solve the problem of different transport types, a DoS performs automatic message routing as required. Therefore, if a station cannot directly communicate with a station to which it wishes to send a message, the DoS will automatically route the message via another station. This is useful to circumvent firewalls and to enable two stations that do not have a common transport type to communicate.

### 3.2 Maintaining Consistency in Synchromedia

*Synchromedia* is made of duplicated objects. Collaborative work consists on the sharing of a collection of duplicated objects. Each duplicated object has a master and duplicas. The master is responsible of updating the itself and its duplicas. When a user perform an action on the master, the master take charge of distributing the effect of this action on all its duplicas. If the action is performed on a duplica, this duplica notify its master about it and the master distribute its effect of the action to all other duplicas.

This approach permit to avoid the emergence of inconsistency rather then allow it and reestablish consistency as some other systems do [3,4]. For each duplicated object, a DoS uses ordering to avoid inconsistency. Ordering consists

on accepting actions that may cause inconsistency and postponing their execution to a moment that will not cause the emergence of inconsistency. This approach is also known as serialization [3]. Various approaches exist that can ensure consistency requirements based on logical time [6] (FIFO, CAUSAL and TOTAL). FIFO order implies that messages from the same source to the same destination are delivered in their order of generation. CAUSAL [6] order implies that messages that are linked by a causal relation are delivered in FIFO order. TOTAL order implies that all destinations receive all messages in the same order regardless of their emission order.

A DoS uses a TOTAL ordering scheme for each duplicate object. This scheme guarantee that all copies of any duplicated object remain consistent during the collaborative work. In that way the whole collaborative work remain consistent.

Although the TOTAL ordering scheme usually increase response times and notification times, its use within a DoS improves this weakness. In fact, each duplicated object is treated separately so that the queuing time is reduced. In addition, masters of duplicated objects are distributed all over the network so that the the processing charges is shared between many stations.

### 3.3 Management of Presence in Synchromedia

DoS don't provide appropriate mechanisms for the control of presence. Although they provide a joiner-based mechanism for the management of sessions, they don't allow the control of groups and users. Groupware developers have to implement their own mechanisms for controlling users and groups.

As part of the project *Synchromedia*, we designed and implemented a mechanism for the management of presence in groupware. This mechanism manage sessions according to the joiner-based approach, though we added the option of launching sessions in the absence of users. This option give more independence to the system towards its users. As far as the management of users and groups are concerned, we adopted an organization of work meeting the needs of collaborative work at the university level. Users are organized in groups, and are administrated and controlled according to their *roles*.

**Organization of Work.** The work in *Synchromedia* is organized according to the four next guidelines:

1. a single *work space* includes all participants in the collaborative works. This space correspond to the university in the real context;
2. in this *work space*, there may be as many *work groups* as one wishes. Each *work group* includes all users working on the same theme and having the same objectives (students and teacher). *Work groups* are not necessarily mutually exclusive. This means that a user may belong to more than one *work group*;
3. the users of a *work group* collaborate during a *work meeting* (i.e., a course). A users can work simultaneously in many work meetings if he is member of the corresponding *work groups*;

4. the users have different rights. These in turn depend on the user's *role*. For instance, a teacher may perform some actions that are not permitted to student. During a *work meeting*, each *work group* is administrated by an *administrator* (i.e., the teacher).

Figure 2 illustrates the organization of work mentioned above.

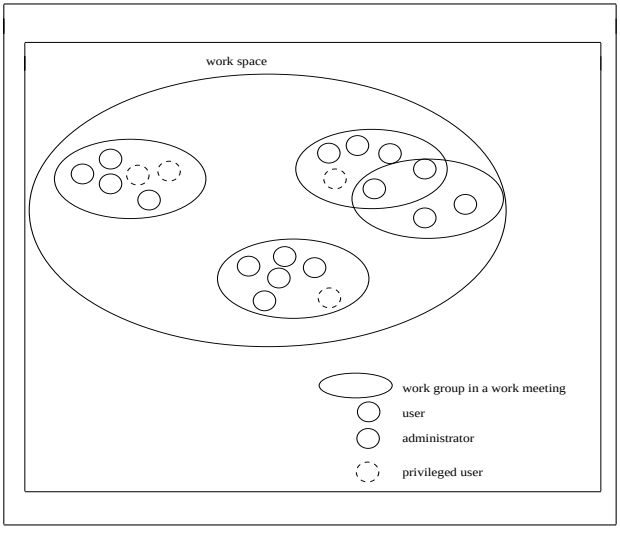


**Fig. 2.** Organization of work

**Administration of Users and Groups in Synchromedia.** The administration of users and groups in *Synchromédia* is made according to a new model that we called DRBAC (Dynamic Role Based Access Control). This model is derived from the Role Based Access Control model (RBAC) [8,1,2,5]. RBAC is used to describe security mechanisms that mediate users' access to computational resources based on role constructs. A role defines a set of allowable activities for users authorized its use. It can be thought of as a job title or position within an organization, which represents the authority needed to conduct the associated duties. The introduction of roles in the security mechanisms considerably reduces the cost and complexity of administration ([1] presents an exhaustive study of this issue). As shown in figure 3, RBAC associates users with roles and roles with permissions. A permission is the right to perform a given action on a given object. A user who is authorized to a role, is consequently authorized to all permissions underlying of this role.

DRBAC is derived from RBAC. It takes advantage of RBAC to reduce the administration cost. In addition, it provides new features that meet the groupwares requirements. DRBAC distinguishes between two states of the system:

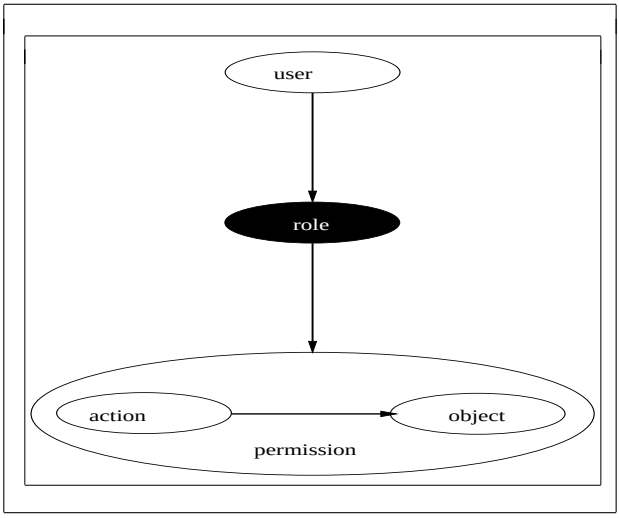


**Fig. 3.** RBAC approach

*static* state and *execution* state. The static state is the state of the system before initiating the collaborative work. The execution state is the state of the system during the collaborative work. The figure 4 presents these two states.

In the static state, we introduce a new association action/user that doesn't exist in the RBAC. We call this association *administration* as it refers to actions that one user may perform on another user. For instance *preventing a user from writing on the whiteboard* is an administration that a teacher can perform on a student. In this way, DRBAC associates, in one hand, users to roles and, in the other hand, roles to permissions and administrations. Those associations are immutable during the static state, however they may change during the execution state. A new association appears also in the execution state. This association links users directly to permissions and administrations. This association is very important as far as it results of the administrations performed by privileged users on others users. For instance if the teacher allows a student to write on the whiteboard, a new association between this student and the permission *write on the whiteboard* appears even though this permission is not allowed to the role *student*. The associations between users in one side and permissions or administrations in the other side are temporary and don't last after the end of the collaborative work.

The introduction of the execution state with the DRBAC model permit a better control of users while working. It permits a real time administration of users. For instance a teacher is given a mean to modify student's rights and privileges, i.e., he may allow or prevent a student from writing on the whiteboard.

DRBAC predicts that users are organized in groups. The creation and destruction of groups are also permissions that can be linked to some roles. In

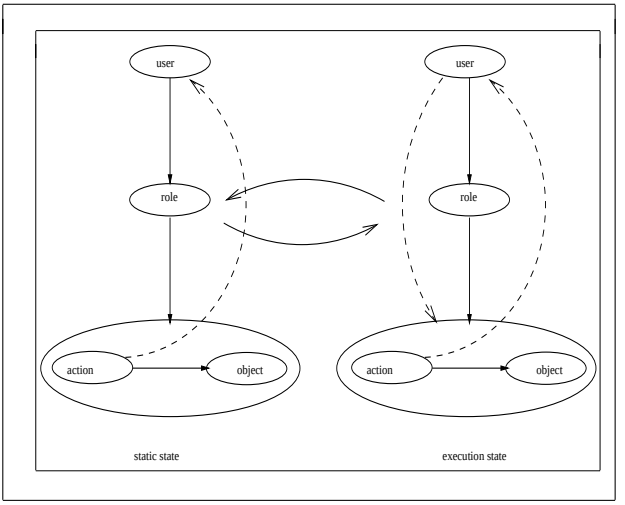


**Fig. 4.** DRBAC approach

addition, groups may be definite statically and users are given or not the right to gain access to them.

**Integration of DRBAC.** In order to integrate the DRBAC to *Synchromedia*, we have developed the following three components:

- Static DRBAC database: this database is used to specify the users, their roles and their memberships. To each role is assigned a set of permissions and administrations. If a user is active in a role, he is allowed to perform all actions linked to this role.
- Activation manager: this manager is responsible for controlling users within the execution state of the system. It is directly integrated in *Synchromedia*. To each duplicated object *User*, are associated two structures : a *permission table* and an *administration table*. The permissions table contains the permissions of the corresponding user and the administrations table contains the administrations of the corresponding user. After login, user's permissions and administrations are downloaded from the static control database to the structures and distributed to all users of his/her group. When the user attempts to perform an action, the system verifies his permissions table before allowing the action to be done. Permissions could be changed at runtime by an authorized user. The administration table of authorized users contains the administrations that allow them to modify others' permissions. Permissions are actually implemented in the form of a matrix. Rows of a permission matrix correspond to objects and columns to actions performed on those objects. The content of a cell $(i, j)$ is a variable that may take one of the three following values: 0, 1, or -1, 0 meaning that the corresponding

user is not allowed to perform the action $i$ onto the object $j$, 1 meaning that he is allowed to perform that action, and -1 meaning that the action is not defined for the object. Table 5 gives an example of the permissions matrix of a student in *Synchromedia*.

| | Write | Erase | Change entries | Stop |
|---|---|---|---|---|
| Whiteboard | 1 | 1 | -1 | -1 |
| Virtual lab | -1 | -1 | 0 | 0 |

**Fig. 5.** Student's permission table

As well as permissions, administrations are also implemented in the form of matrix. Rows of an administration matrix correspond to roles and columns to actions performed on the users of those roles. The content of a cell $(i, j)$ is a variable that may take the values: 0 or 1, 0 meaning that the owner of this administration table is not allowed to perform the action $i$ onto the users of the role $j$, and 1 meaning that he is allowed to do so. Table 6 gives an example of the administration matrix of a teacher in *Synchromedia*.

| | Allow writing | Prevent writing | Eject |
|---|---|---|---|
| Assistant | 1 | 1 | 0 |
| Student | 1 | 1 | 1 |

**Fig. 6.** Teacher's administration table

– Administration tool: the administration tool is an interface designed for the administration of the static control database. It's purpose is to let the administrator define and modify users, groups, roles, and permissions for the current session.

## 4 Conclusion

By abstracting the network layer, a DoS allows groupware developers to concentrate their development efforts on content and tools rather than network issues. Via high-level features such as communication facilities, load balancing, fault tolerance, and hierarchical messaging, an DoS-enabled groupware effectively deals with the Internet's inherent unreliability, high latencies, bandwidth limitations, and resource constraints.

In *Synchromedia*, we have designed and implemented our own mechanism for the management of presence. The main idea behind this mechanism is to allow privileged users to administrate their groups during the collaborative work.

Our future work will be focused on the following two directions:

- Comparison between the three run-time architectures used to build *groupwares* (centralized, replicated, and DoS architecture). The criteria of comparison are: response time and notification time. This comparison can be made using a mathematical model, a simulation or by making some experiences and taking measurements.
- Further development of the DRBAC model, especially to formally define the different components of this model and to introduce the static and dynamic constraints that may apply. The existence of a well-defined model will make easier the implementation of groupwares and give more flexibility to their use.

**Acknowledgments.** Netz product was provided by Quazal Inc. from Montréal, special thanks to all the team up there. This research work was supported by grants from FODAR (University of Québec Academic Fund and Development Fund) and the MEQ (Department of Education of Québec).

## References

[1] D.F. Ferraiolo, J.F. Barkley, and D.R. Kuhn. A role based access control model and reference implementation within a corporate intranet. *ACM Transactions on Information Systems Security*, 1, February 1999.

[2] D.F. Ferraiolo, J. Cugini, and D.R. Kuhn. Role based access control: Features and motivations. 1995.

[3] S. Greenberg and D. Marwood. Real time groupware as a distributed system: Concurrency control and its effect on the interface. In *Proceedings of the ACM CSCW 94 Conference on Computer Supported Cooperative Work*, pages 207–217, October 1994.

[4] A. Karsenty and M. Beaudoin-Lafon. Slice: a logical model for shared editors. In *Real Time Group Drawing and Writing Tools*, pages 156–173, McGraw-Hill, New York, 1995.

[5] D.R. Kuhn. Mutual exclusion of roles as a means of implementing separation of duty in role-based access control systems. In *Second ACM Workshop on Role-Based Access Control*, 1997.

[6] L. Lamport. Time, clocks and the ordering of events in a distributed system. *Communications of the ACM*, pages 558–565, july 1978.

[7] Quazal. Net-z 2.0 - technical overview.

[8] R.S. Sandhu, E.J. Coyne, H.L. Feinstein, and C.E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.

[9] T. Urnes and R. Nejabi. Tools for implementing groupware: Survey and evaluation. Technical Report No. CS-94-03, York University, 1994.