# TRANSPARENCY AND AWARENESS IN A REAL-TIME GROUPWARE SYSTEM

*Michel Beaudouin-Lafon and Alain Karsenty*

Laboratoire de Recherche en Informatique (CNRS URA 410)
Université de Paris-Sud - Bâtiment 490
91405 ORSAY Cedex - FRANCE
(33) 1-69-41-69-10, mbl@lri.fr
(33) 1-69-41-65-91, ak@lri.fr

## ABSTRACT

This article explores real-time groupware systems from the perspective of both the users and the designer. This exploration is carried out through the description of GroupDesign, a real-time multi-user drawing tool that we have developed. From the perspective of the users, we present a number of functionalities that we feel necessary in any real-time groupware system: Graphic & Audio Echo, Localization, Identification, Age, and History. From the perspective of the designer, we demonstrate the possibility of creating a multi-user application from a single-user one, and we introduce the notion of purely replicated architecture.

## INTRODUCTION

An important part of the work in the field of CSCW concerns case studies in the context of design by a group, and a number of systems have been developed for this purpose. It was stated in a recent article [15] that "there seems to be a focus on technology for the sake of technology, without much thought about what people actually need." It is true that many groupware systems require special facilities like conference rooms equipped with computers, or high-tech gear to support video telepresence. On the other hand, the work on the software aspects of CSCW is still at its beginning: most applications are dedicated, and very few tools exist. We believe that the user-interface issues of groupware systems should be studied more carefully, especially for real-time (or synchronous) systems.

We advocate *transparent* groupware systems, that is, systems that do not require special settings or hardware, and that integrate smoothly with the way computers are used today. Hence, we are interested in extending the most widespread models of interaction to integrate the new dimension introduced by the group. We see the notion of *awareness* as the key to this transparency. By this we mean that each user should be aware of what the others are doing, to facilitate coordination, but not suffer from constraints related to the group, such as floor control.

Proc. ACM Symposium on User Interface Software and Technology (UIST'92, Monterey, California, USA), ACM Press, 1994, pp. 171-180.

To investigate the notions of transparency and awareness, we have developed a real-time multi-user drawing tool called GroupDesign. The reasons are two-fold. First, most existing real-time groupware tools are pixel editors or text editors. Our tool is a structured graphics editor, which we feel more representative of direct manipulation systems. Second, we did not want to create the system from scratch, and we had an existing single-user extensible drawing tool at hand. This was an opportunity to understand the issues involved in turning a single-user application into a multi-user one, and to get some insight into an appropriate architecture for real-time groupware.

This approach led us to explore real-time groupware systems from two perspectives: the users', and the designer's. From the perspective of the users, we present a number of functionalities that we feel necessary in any real-time groupware system. This can be paralleled with the functionalities that have been identified for single-user systems, such as help, undo, semantic feedback, etc. From the designer's point of view, we demonstrate the possibility of creating a multi-user application from a single-user one, and we present the notion of purely replicated architecture.

The paper is organized as follows. The next section describes the related work. We then describe the system and its architecture. In particular, we present the distributed algorithm that implements the purely replicated architecture. The last section discusses the important aspects of this work with respect to the perspectives we have previously described. We close the paper with a conclusion and an outline of future work.

## RELATED WORK

Within the domain of real-time multi-user editors, Grove [10] is the closest to our system, although it is dedicated to text editing. Transparency is supported by concurrent editing at the keystroke level, while awareness is supported by clouds and aged text. Cognoter and Argnoter [23] also share some similarities with our system, in that the members of a group can simultaneously edit a diagram. The concept of WYSIWIS interface, which was invented when these tools were created, is a main theme of the work we present in this article.

A number of other tools are less closely related to our work. rIBIS [21] is a multi-user hypertext system that allows both loosely coupled and tightly coupled group work. Unlike our system, the tightly coupled mode requires a turn-taking floor control. Aspects [3] is a commercial product for shared

editing of structured drawings. However, it provides few groupware features and it requires the locking of objects. BoardNoter [23] and Commune [19] are painting tools dedicated to tightly coupled meetings. With such bitmap editors, there is no need to address the question of conflicts. CaptureLab [12] and Timbuktu [7] use floor control to take over a different computer, and cannot be considered multi-user editors.

Our system is different from many groupware systems in that it addresses loosely coupled groups instead of tightly coupled ones, and distant users instead of face-to-face meetings. This has a significant impact on the groupware features that are needed.

On the side of architecture and tools for creating groupware systems, most authors recognize that a replicated architecture is worth the difficulties it raises. MMConf [8] has an architecture close to ours, but it encourages turn-taking floor control because when running an open floor, the events are not guaranteed to arrive in the same order at all sites. On the other hand, MMConf is a toolkit, whereas the groupware features of our system are currently integrated in the application. DistEdit [16] is a toolkit for programming multi-user text editors. It relies on the ISIS toolkit [4] for the distributed aspect of the architecture. This may cause performance problems, as stated by the authors, because of the concurrency control algorithms implemented by ISIS. Grove also uses a replicated architecture [11], with concurrency control being achieved by a technique called operation transformation: operations which arrive out of date are transformed so that they can be executed without disrupting the session. If there are n operations, this technique requires $n^2$ transformation procedures, some of which are not trivial to write. Our system uses a similar architecture with a simpler concurrency control scheme. Finally, LIZA [14] is a toolkit that uses a central process with replicated clients, and RendezVous [20] is an example of a centralized application.

The notions of transparency and awareness are put in relation by Lauwers and Lantz [18] in the context of shared window systems. We are trying to do, at the application level, something similar to what these authors propose for window systems: identify functionalities, propose implementation, and develop tools.

## SYSTEM OVERVIEW

GroupDesign is a multi-user drawing tool for structured graphics. It runs on Apple Macintosh computers connected by AppleTalk. The architecture is replicated: an instance of the application (a *replica*) runs on the computer of each user.

A GroupDesign diagram is a set of pages, either independent or connected in an hierarchical way. Within a page, graphical objects (rectangles, texts, etc.) and connectors (links between objects) can be edited as in MacDraw. The complexity of a diagram is such that one can have a large number of users working on different pages. GroupDesign uses a relaxed WYSIWIS paradigm [22]. A strict WYSIWIS approach would not have allowed users to work independently on different diagram areas. The document is the same for all replicas but each user has his or her own view of the diagram (e.g. users have an independent control over the scroll bars and window placement).

Whereas a meeting involves a few users during a short time, GroupDesign sessions can last indefinitely with participants entering and leaving during the session. This new freedom created a new problem however. For instance, a user may enter in the middle of a session without knowing the recent history of the document. If changes have been made to the diagram, and the user does not agree with the changes, he or she needs a means of identifying the user(s) who made the change to discuss it with them. The features we provide to compensate for the lack of group memory are History, Age and Identification, which allow one to be informed of when, how and by whom changes have been made to the diagram.

We have also developed real-time features to provide the group with a means of understanding simultaneous actions on the document. Those features are Graphic & Audio Echo, Localization, and Teleconference. Finally, we developed a feature that we call Time-Relaxed WYSIWIS. It gives users a way to have privacy while working.

Most of the groupware features use color to identify each user. The name of the users and their associated colors appear in a menu. In the following paragraphs are more details about the groupware features.

### Graphic & Audio Echo

We define Echo as the representation of a user's action upon other users' interface. We designed the echo to be comprehensible yet not too distracting. Thus, one can choose to observe the group at work or decide to focus on a task without being disturbed. The echo is graphic if users share a common view of the diagram or audio if operations occur out of the window.

Graphic Echo is a two-phase process, as illustrated below. The first phase takes place when an operation is initiated,

| | user | group |
|---|---|---|
| phase 1 | presses the mouse on the rectangle. | “move” icon displayed on top of the rectangle (Fig.1). The icon has the user's color attribute. |
| | drags the rectangle to the new position. | the icon is still displayed (Fig. 1). |
| phase 2 | releases the mouse. The rectangle is drawn at the new position. | the icon is erased and the rectangle is moved smoothly to the new position (Fig. 2). |

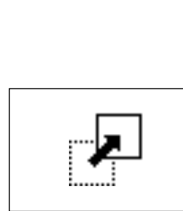

Figure 1 - busy icon

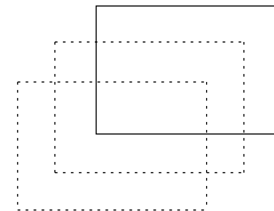

Figure 2 - animation

and uses an icon which has the same function as the busy signal of the Colab [23]. The second phase takes place when the operation is completed, and uses animation to make it easy to understand the operation.

An icon displayed in an object indicates that a change is about to be made by another user. Its shape indicates which operation is underway, and its color indicates the author of the operation. While the icon is displayed, the object is partially locked. A user can still modify the object unless the operations are not compatible - such as moving an object being moved.

We developed Audio Echo to cover non-visible operations, since Graphic Echo does not cover modifications occurring outside of the window view. The Arkola bottling plant simulation [13] showed that sound can play a significant role in groupware systems. In GroupDesign a sound is associated with every graphical operation.

The graphic and audio echo are modes that users can enable or disable. Users can also control the set of operations for which an echo is provided.

### Localization and Identification

Localization makes it possible to coordinate views with another user. It is a strict WYSIWIS feature for a relaxed WYSIWIS interface. In this mode the participants' front window is displayed as a rectangle with their assigned color so that users can see each other's current view. Selecting a user's name from the menu “teleports” one’s view to the area currently viewed by that user. Thus, one can have both an independent view of the diagram and be able to synchronize views with another user. This gives the users a sense of territory - when they modify objects, they are aware if other users are currently viewing the changes.

Figure 3 is an example of three users in the same session viewing an overlapping part of a page. Heather is in Localization mode and thus can see Alain's and Michel's views. Localization alone is not sufficient to identify who has just modified a diagram, since several users can view the same part of the diagram. Moreover, once the changes are done, there would be no means to identify each user's operations. The Identification feature is used to identify the name of the users who modified the diagram. Identification is done through colors: every object is displayed with the color corresponding to either the user who created it or to the last user who modified it. This is useful to discuss design issues on a particular area of the diagram without disrupting the session participants. Using Identification, one can contact the specific users who made the changes.

Both Localization and Identification are modes. They do not interfere with the drawing activity, but add information about users to the participants.

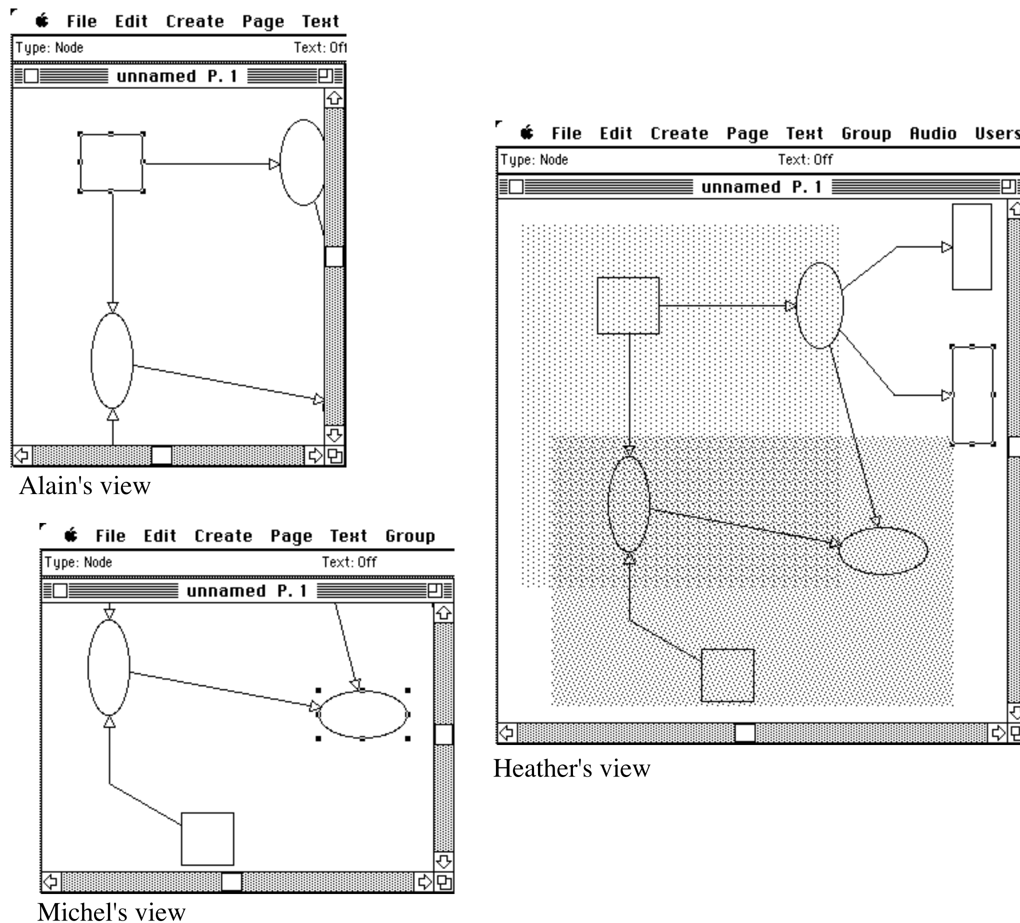


Figure 3 - Localization

### Age and History

The only time-related information usually available concerning a diagram is the last time it was saved into a file. No information is given about the last date of modification of the objects in the diagram. In a multi-user application this information is important. For instance, if many objects in the same area have been recently modified, this area is probably a "hot spot" of the group's activity.

The age of objects is displayed using colors which vary from red (recently modified) to blue. Age allows one to find out when objects have been modified - but not how. This is why we also provide the History feature. The last actions of the group can be replayed using a control panel similar to a tape recorder. The type of operation and the name of the user who did it are displayed when an operation is replayed either forward or backward. Whenever one enters this mode, the system is off-line. It gets other users' operations but keeps them in a queue to process them as soon as the user closes the control panel.

History is a command, thus one cannot change the diagram when replaying the last actions. On the other hand, Age is a mode. This mode is incompatible with Identification because both use the color of objects to convey their information.

### Teleconference

This feature is used when users want to work in a tightly coordinated way. Teleconference allows a subset of the group to work under an almost strict WYSIWIS interface. By "almost strict" we mean that the users see each other's windows modifications (resize, scroll, etc.), but not the movements of the cursor (for performance reasons). Note, however that the application still runs an open floor, unlike most systems that operate in strict WYSIWIS mode.

Teleconference is useful when some members of the session want to discuss a particular design issue. In this situation, additional communication channels are needed (e.g. audio/video links). We could also consider adding a telepointing facility to the system. In the current version it is possible to use a GroupDesign object as a telepointer.

### Time-Relaxed WYSIWIS

This feature is complementary to the Teleconference mode, in that it gives a user some privacy during a session. In this mode, the user's current modifications do not appear on the participants' windows. When satisfied with the resulting diagram one can decide to commit himself or herself and the participants' diagram is then updated. This feature is useful in a variety of circumstances. If a user is not familiar with the application, it might be inhibiting to show his or her clumsiness to the group, for instance. Another example is that users can modify the diagram, and if they are not satisfied with the result, they can cancel their modifications without disrupting the session.

## ARCHITECTURE

We have implemented what we call a *purely replicated* architecture. A replica of the application runs on each computer. Unlike many systems that use a replicated architecture, GroupDesign does not use any central process for the coordination of the replicas, nor does it give a special role to the user who first launches a session. All replicas are strictly identical, and no other process is required. The replicas communicate by sending each other events through LocalTalk, using the facilities of the Apple Event Manager of Apple's new System 7.0 [1].

### From Design to GroupDesign

We neither built GroupDesign from scratch nor modified the code of an existing program; rather we used an existing application, MetaDesign, that we modified through its programming interface, called Design/OA.

MetaDesign is a graphical editor from MetaSoftware Corp. used to create logical diagrams made of nodes and connectors. Nodes automatically maintain their connections when they are moved, making graphics easy to change. A diagram can be made of different pages, either independent or connected in an hierarchical way. A page can be attached to a node, so that when double-clicking in the node, the page opens with a subdiagram in it.

Design/OA [9] is a set of functions to customize, extend and tailor MetaDesign. It contains three categories of functionalities: to install filter functions to intercept and control user actions, such as creating a node; to extend and customize MetaDesign's menus; and to create and modify diagrams, the same way one would do manually with MetaDesign.

We have built GroupDesign with Design/OA as follows. Filter functions have been installed to intercept the user actions and broadcast them to the other replicas. We have added two menus, one for the control of the groupware features, the other to display the user names and their associated colors. Finally, we have used the functions that modify the diagram to handle the events coming from the other replicas.

Using Design/OA made our task much easier than starting from scratch. Design/OA makes it possible to completely change the features, look, and feel of MetaDesign so that we have not been limited by the interface when building GroupDesign.

### Managing the replication

The main advantage of a replicated architecture is the possibility of having an interface with a response time as short as for a single-user interface. This is achieved by using a high-level asynchronous protocol. This protocol is implemented in a software layer which is mainly independent of the application. This means that the technology we have used to turn a single-user application into a multi-user applies to many other applications, although we have not come to the point where we can provide a groupware toolkit for that purpose, like DistEdit [16] or MMConf [8].

The protocol consists of events being sent by the replicas to describe the commands carried out by each user. An event contains an operation, defined as a triple (object, op-code, arguments). The protocol is asynchronous because a replica never waits for a reply from the other replicas. This requires a distributed algorithm that ensures that the events are processed at each site in a compatible way.

The basic idea is to use a logical clock [17] at each site to timestamp events sent by this site. Each replica keeps a list of received events. When a replica receives an event with timestamp *t*, it should undo any operations triggered by events with a more recent timestamp, then handle the received event, and finally redo the operations. However, if the operation contained in the received event commutes with the more recent ones, then it is not necessary to undo and redo those operations. Similarly, if the received operation is masked by a more recent one, i.e., if applying *op1* and then *op2* is identical to applying *op2*, then the received operation can be discarded. For instance, if one user moves a rectangle and another changes its color, the operations commute and thus can occur in different orders at different sites. If a user sets the color of an object to red (event *e1* at time *t1*) and another sets it to green (event *e2* at time $t2 > t1$), a third replica receiving *e2* then *e1* will handle *e2* and discard *e1*.

In GroupDesign, except for the creation of objects, for any couple of operations *(op1, op2)*, if *op1* and *op2* are equal then *op2* masks *op1*, otherwise *op1* and *op2* commute. Two operations are equal if their object and op-code are equal. In particular, this means that objects are independent, i.e., for any two different objects *A* and *B*, any operation on *A* commutes with any operation on *B*. This also means that operations are independent, i.e., the result of applying two different operations to the same object does not depend on the order of the operations. Under the above condition, a received event is always handled immediately, without having to undo and redo operations. The creation of objects is a special case since it is impossible to apply operations on objects which have not been created yet. Events carrying such operations are stored in a separate list which is examined each time an object is created.

The management of logical clocks makes it possible for two events to be sent by different replicas at the same logical time, causing a conflict. Fortunately, the probability of such a conflict is quite low because the time window within which it can occur is very small. For instance, user A should change the color of an object to green, and user B should change it to red before the event is received from A. Would this happen, we use the total ordering of events defined by Lamport [17]. As a consequence, the operation of one of the users is executed while the other one is discarded and the user notified.

In the Time-Relaxed WYSIWIS mode, conflicts may occur more often because they are detected only when the user commits. Hence, there is a conflict for each object modified by a user in this mode which has also been modified by another user. We have decided that the user in the time-relaxed mode has his or her conflicting operations canceled, in order to disrupt the least number of users. This is not a satisfactory solution, and we need more experience with the usage of this feature to devise a better one. These conflicts can be minimized, though. A user in time-relaxed mode is signaled to the group. Using the localization feature, one can avoid modifying the objects viewed by the user in time-relaxed mode.

### Description of the algorithm

The types and data structures used by the algorithm are shown in figure 4. Each event contains a description of the operation (object-id, op-code, arguments) and bookkeeping data (clock, sequence number, and site number of sender). The object-id is assigned by the replica that first creates the object. It is a unique identifier obtained by concatenating a unique local identifier and the site number. Each replica manages an associative table that maps identifiers to objects. Although we use an array objectTable in the declarations of figure 4, a real implementation should use a better scheme, like a hash-table, because an array would be too large and too sparse.

The implementation of the algorithm is shown in figures 5 and 6. It is straightforward, except that it ensures that the set of received events does not grow indefinitely. This set is used when "old" events arrive, in order to detect conflicts and to determine whether the operation has been masked. Events can be discarded from this set if their logical clock is

```
type
  Site = positive integer;   (* site numbers *)
  Time = positive integer;  (* clock values *)
  Seq = positive integer;   (* sequence numbers *)
  Identifier = integer;        (* object identifiers *)
  OpCode = (Alive, Create, …);(*application operations *)
  Args = (* description of operation arguments *);
  ObjectPtr = (* pointer to an application object *);
  Event = record
    op : OpCode;               (* operation to carry out *)
    objId : Identifier;        (* id of object, or nullId *)
    args : Args;               (* arguments of operation *)
    time : Time;               (* logical time of sender *)
    seqno : Seq;               (* seq number of sender *)
    sender : Site;             (* site number of sender *)
  end;
  EventList = list of Event;
const
  timeOut : Time := 10;      (* timeout for Alive events *)
  nullId : Identifier := 0;    (* id for ops without object *)
  nullArg : Args := (* an 'empty' argument record *);
var
  sites : set of Site;       (* sites in the session *)
  me : Site;                 (* identifier of this site *)
  localTime : Time;          (* logical clock of this site *)
  seq : Seq;                 (* seqno of last event sent *)
  (* associative table mapping obj-ids to objects *)
  objectTable : array [Identifier] of ObjectPtr;
  (* list of recent events, sorted by ascending time *)
  recent : EventList;
  (* list of events for objects not yet created *)
  delayed : EventList;
  (* seqno of last event received in sequence *)
  lastSeq : array [Site] of Seq;
  (* time of last event received in sequence *)
  lastClock : array [Site] of Time;
(* list manipulation procedures *)
procedure Insert (var el: EventList; before, e: Event);
procedure Append (var el: EventList; e: Event);
procedure Remove (var el: EventList; e: Event);
(* send an event to a site *)
procedure SendEvent (s: Site; e: Event);
(* application-dependant functions *)
procedure Apply (op: OpCode; o: ObjectPtr; a: Args);
function Create (id: Identifier; a: Args) : ObjectPtr;
```

Figure 4 - Declarations for the algorithm

such that no older event will ever arrive. The algorithm stores in the array lastClock, the logical clock of the last event received *in sequence* from each site, using the sequence number sent with each event and the auxiliary array lastSeq. Any event older than the minimum value of lastClock can be discarded. However, this is not sufficient to ensure a bounded size to the set of received events. Indeed, if a site is idle, it is not sending events, therefore the minimum value of lastClock stays the same and the size of the set never decreases at the other sites. Therefore, if the current site has not been sending events in the last timeOut logical clock ticks, then it sends an "Alive" event to signal that this site is still alive. This event will update the lastClock array of the other sites, giving them the opportunity to reduce the list of recent events.

For the sake of simplicity, we have not described the handling of deleted objects in the algorithm. When an object is deleted, it should be marked as such in the objectTable. The “Execute” procedure should discard operations on deleted objects and be cautious with deleted objects that have not been created yet.

### Designing the set of operations

The events carry operations which describe the modifications made by users to the document. However, a distinction must be made between the commands that the user can use and the actual operations that are sent by means of events between sites. The design of the set of commands must be driven by the task space that the application addresses and the conceptual model of the application, as for any interactive system. The design of the set of operations, however, must be driven by the constraints of the distributed algorithm, namely the fact that identical operations must mask each other and non-identical ones must commute. Although this was achieved quite easily with GroupDesign, it might not be the case in other applications. In such cases, one must resort to the undo-execute-redo scheme for operations that do not commute nor mask. Another possibility consists in transforming the operation, as done in Grove [11].

Some commands are specified instantaneously, such as the selection of an object, the activation of a menu command, etc. Each such command corresponds to a single operation. Other commands need a certain amount of time to be specified. For instance, creating or moving an object involves dragging the mouse, some menu commands actually open a dialogue box for the specification of additional arguments, and the command is executed only when the user hits the OK button. In GroupDesign, such commands are translated into two operations. The first one is sent when the commands starts, so that the other sites can provide an echo of the command and lock the appropriate objects for the operation being specified, thus reducing the risk of conflict. The second operation is sent when the command finishes, or when it is aborted. It describes the command itself, as if it had been specified instantaneously. This is how GroupDesign implements the two phase echo described in a previous section.

```
(* broadcast an operation executed locally *)
procedure Send (op: OpCode; id: Identifer; args: Args)
var
  s : Site;
  event : Event;
begin
  (* construct event *)
  event.op := op;          event. objId := id;
  event.args := args;      event.time := localTime;
  event.seqno := seq;      event.sender := me;

  (* send it to other sites *)
  for s in sites do
    if s ≠ me then SendEvent (s, event) endif;
  endfor;

  (* update local state *)
  localtime := localTime + 1;
  lastClock [me] := localTime;
  seq := seq + 1;
  lastSeq [me] := seq;
end;

(* handle an incoming event *)
procedure Receive (event: Event)
var
  e, recentEv : Event;
  found : boolean := false;
begin
  if event.time > localTime then
    (* event newer than anything in recent *)
    Execute (event);
  else
    (* find a more recent event with same operation *)
    for e in recent do
      if e.time ≥ event.time and e.op = event. op
        and e.objId = event. objId then
        found := true;
        recentEv := e;
        break;   (* exit for-loop *)
      endif;
    endfor;

    if found then
      if recentEv.time = event.time then
        (* conflict : use total ordering of Lamport *)
        if recentEv.site < event.site then
          Execute (event);
        else
          (* conflicting operation discarded *)
          (* notify user *)
        endif;
      else
        (* masked operation ignored *)
      endif;
    else
      (* event commutes with all events in recent *)
      Execute (event);
    endif;
  endif;

  (* update local state *)
  Update (event);
end;
```

Figure 5 - Main procedures of the algorithm

```
(* update local state after receiving an event *)
procedure Update (event: Event)
var
  e : Event;
  s : Site;
  found : boolean := false;
begin
  (* update local time *)
  localTime := max (localTime, event.time) + 1;

  (* update lastClock, and insert event in recent *)
  for e in recent do
    if e.sender = event.sender
    and e.seq = lastSeq [e.sender] + 1 then
      (* revent eceived in sequence *)
      lastSeq [e.sender] := e.seq;
      lastClock [e.sender] := e.time;
    endif;
    if not found and event.time > e.time then
      (* insert event, keeping recent sorted *)
      found := true;
      Insert (recent, e, event);
    endif;
  endfor;

  (* discard useless events from recent *)
  for e in recent do
    if e.time ≤ min (lastClock)
      then Remove (recent, e);
      else break;
    endif;
  endfor;

  (* send alive event if inactive for a long time *)
  if localTime > lastClock [me] + timeOut
    then Send (Alive, nullId, nullArgs); endif;
end;

(* execute the operation contained in an event *)
procedure Execute (event: Event)
var
  e : Event;
  object : ObjectPtr;
begin
  if event.op = Alive then (* do nothing  *)
  elsif event.op = Create then
    (* create object and store it *)
    object := Create (event.objId, event.args);
    objectTable [event.objId] := object;
    (* execute delayed operations on this object *)
    for e in delayed do
      if e.objId = event.objId then
        Apply (e.op, object, e.args);
        Remove (delayed, e);
      endif;
    endfor;
  else
    (* delay operation if object does not exist yet *)
    object := objectTable [event.objId];
    if object = NIL
      then Append (delayed, event)
      else Apply (event.op, object, event.args)
    endif;
  endif;
end;
```

Figure 6 - Auxiliary procedures for the algorithm

### Newcomers

A newcomer is a site entering a session which is already running. Unless the session starts synchronously for all users and no newcomer is accepted, every site but the first one is a newcomer. Handling a newcomer means transferring a known state to it, to let every other site know of its existence so that it can receive events from them, and to decide when the newcomer can start to behave like a normal site. Moreover, several newcomers entering the same session simultaneously must be handled correctly. In the rest of this section, we sketch out the algorithm for handling newcomers.

The algorithm is based on the fact that the state S of a site can be characterized by stateTime, an array of clock values indexed by site numbers, as follows: S corresponds to the execution of a set of events E such that each event e of E verifies e.time ≤ stateTime [e.sender].

The algorithm proceeds as follows: the newcomer multicasts the other sites and gathers their logical clocks. The other sites can then start to send events to the newcomer. These events are stored by the newcomer in a list and processed once the newcomer is initialized. The newcomer then asks a site to transfer a state that corresponds at least to the clocks it has gathered. The transferring site sends its state and the corresponding stateTime array to the newcomer. It also commits to forward to the newcomer any events older than the stateTime array that it has not yet received. When the transfer is complete, the newcomer can process the events received from the other sites, and start to behave normally.

## DISCUSSION

The context in which a groupware system is to be used has an important impact on its features. A number of systems have been designed to support a group activity in face-to-face meetings, or with video telepresence. In such a context there are few users, and a tight coupling between them. As a result it is quite easy to monitor the activity of the whole group.

The context we have chosen is one of a loose coupling with a potentially large number of users. With this setting, the granularity of the group work is larger: instead of several users working on the same task, each user is assigned a different task, in the context of the more global activity of creating the document.

In the next section we analyze how transparency and awareness support the group work under this loose coupling. We then analyze the features that we propose from the user's point of view. Finally we discuss the impact of a purely replicated architecture on the engineering of groupware systems.

### Transparency and Awareness

For the end-user, transparency means that the system does not bring obstacles in the way of the task he or she is carrying out. In the context of a groupware system, this means that the system should not impose unnecessary constraints. Floor control is such a constraint: in a system that requires one active user at a time, a user must wait to take the floor to do something. An open floor system relieves this constraint. Another constraint is strict WYSIWIS. Relaxing WYSIWIS creates some well-known

problems [23] due to the fact that the reference to an object by a user may not be understandable to another user.

Transparency in GroupDesign is characterized by an open floor, and by a range of relaxed WYSIWIS features. We have not implemented turn-taking protocols, because we believe that an open-floor supported by social protocols and a good awareness of the group is more efficient and closer to real life. Also, it is much easier to implement floor control in an open-floor system, than to implement parallel activities in a system designed for turn-taking.

The variations of WYSIWIS that we offer range from almost strict WYSIWIS in Teleconference mode, to a very loose WYSIWIS in time-relaxed mode. We believe that strict WYSIWIS is only meaningful in a face-to-face meeting, where one wants to see another user's screen at the level of cursor movements; most of the time such a tight coupling justifies a floor control. The *almost* strict WYSIWIS provided by GroupDesign is adapted to distant users, because it works at the level of views and operations. We think that a more strict approach would not give much more transparency, and would be extremely difficult to implement efficiently (if even possible in our setting).

Referring to the four dimensions of WYSIWIS identified by Stefik et al. [22], we can say that our system gives control over relaxing WYSIWIS in time, population, and congruence of views, but not in display space. Relaxation of the time constraint is controlled by the time-relaxed mode, in which a user explicitly commits his or her changes. Relaxation of the population is possible to some extent because one can use the teleconference mode with only a subset of the group. Relaxation of congruence is provided when the teleconference mode is not active, because each user can independently move and resize a window, and scroll and zoom its contents. Moreover, the fact that each user can be in a different mode (Localization, Identification, Age, Echo) is also a relaxation of the congruence. The display space to which WYSIWIS applies is fixed, however, because only GroupDesign windows are subject to WYSIWIS.

Another aspect of transparency is the ability to observe the document. Not only is a user able to navigate geographically in the document, but he or she is also able to navigate in time and over the dimension of users. Navigation over time is achieved by the Age and History functions. Navigation over the dimension of the users is carried out by means of the Identification and Localization functions. These features actually provide the articulation between transparency and awareness. Transparency is conveyed by the navigational aspects of these functions, while the awareness is conveyed by their modal aspect. For instance, the Localization mode displays the areas viewed by the other users. This supports awareness in the sense that the information is brought to the user without interfering with his or her task, so that it is taken into account subconsciously.

Awareness of the other users of the group is further conveyed by the echo of commands, which we see as essential. The challenge of a good echo is to provide an accurate, non-disturbing and efficient feeling of what the other users are doing. Because echo must be in real time, it also raises implementation issues, so that the choice of a particular echo is a trade-off between what is feasible and what is desirable.

What would be an ideal echo? The first idea that comes to mind is to provide other users with the feedback given by the user carrying out the operation, following a strict WYSIWIS approach. For instance, if a user moves an object around, then the shadow of that object should move on all the screens. Beside the problem of the efficient implementation of this, we do not believe this is the best echo. When I am moving an object, the feedback is appropriate because I can anticipate it: I know that I am moving the mouse. But if a user sees an object moving around by itself he will probably be quite surprised, because he cannot anticipate it.

The two-phase echo we provide has proved more appropriate: when an icon appears in an object, the user knows that something is going to happen to it, and the shape of the icon indicates what. Moreover, the object is locked for the operations that are incompatible with the one displayed by the icon. Then the object moves, or changes shape, or disappears. The animation provided by GroupDesign in this second phase is important, because other users are not anticipating the effect of the command (unlike the user who initiated that command). As demonstrated by Card et al. [6], immediate response animation gives enough time to the other users to understand what is about to happen.

### Existentialism at the Interface

In a single user application, a user should always be able to answer the following questions: Where am I? How did I get here? What can I do now? This should also be the case in a multi-user application. Each user should be able to answer a variant of these questions: Where are we? How did we get here? What can we do now?

*Where are we?* This question is related to the global state of the document, and the individual state of each user. The global state of the document can be observed by the usual navigation commands (scrolling, zooming). The individual state of each user consists in the part of the document he/she is currently editing. The Grove editor [10] used clouds to visualize a part of a text under modification by another user. This is appropriate for a text document, where the modifications are localized. This is not the case with a drawing tool. The Localization mode of GroupDesign makes the areas of the document being visualized by the other users visible. It provides less information than the clouds, which track both the location and the activity of each user. On the other hand, the Localization gives a sense of territory which is more stable over time than the clouds.

Another feature to answer the question "Where are we?" is Identification. This mode shows who created or last modified each object. We do not provide any ownership feature that would give access rights and the like. Rather, we prefer to give the users simple and efficient ways to know who did what, so that they can decide by themselves whether they should or should not modify an object. Identification, like other features (especially History), tend to make the document transparent in the sense that everybody can understand why the document is like it is.

*How did we get here?* Because a document can be edited by several users over a long period of time and possibly across several sessions, keeping the history of the document becomes an important issue. In a single-user application, this is less important because the user easily remembers the sequence of operations that led to the current state of the document: he only has to manage his own time. In a multi-user application, the times of each user are intertwined to form the global time of the document, which cannot be easily reconstructed by a user. The Age and History features are meant to help users reconstruct this global time.

The Age mode is comparable to the aged text of Grove [10]. In the same way as Localization creates a map of territories of the different users, Age gives a time map which is easy to read. In addition to this dynamic display of the time map, the History command gives a way to navigate in time.

*What can we do now?* Answering this question means that a user can know which actions are possible in the current context, and which are not. The partial locking that occurs at the first phase of echo keeps the user from modifying an object being modified by another user. This shows the user what he cannot do. Moreover, the non-intrusive aspect of echo works at the subconscious level and prevents most conflicts, as already noted with Grove [10].

The ability to completely avoid such conflicts is a trade-off, and in GroupDesign conflicts are greatly reduced by the echo, but are still possible. Our experience to date is that conflicts never happen in practice, thanks to the response time of the network.

### Engineering groupware systems

Implementing real-time groupware systems is a challenge. Not only has one to devise new interaction techniques and artifacts, but one also has to face the difficulty of implementing a distributed system. We see as essential for a real-time groupware system to provide an immediate response to each user's actions. This requires some degree of replication, implemented by a distributed algorithm. We have chosen in GroupDesign to replicate the whole application. Beyond the immediate response time that our distributed algorithm ensures, this has three main advantages. First, it is possible to transform an already existing application into a groupware one. Second, it is fault-tolerant since each replica is autonomous. Finally, a user can seamlessly switch between a single-user and multi-user usage of an application. This last property is important for the acceptance of groupware by end-users, as explained by Baecker [2].

Replicating the whole application might not be feasible with other systems, or it might not be desirable. In such situations, the best architecture probably is to centralize the functional core of the application and to replicate its interface. The notion of pure replication can still be applied to the user interfaces of such a system, encouraging a better separation between user interface and functional core. Indeed, the communications between the replicas of the user interface will bypass the functional core in order to achieve the kind of functionalities that we have introduced in this article (Echo, Localization, etc.), which are independent of the functional core.

Before gaining more experience with such mixed architectures, pure replication, as implemented by the distributed algorithm that we have presented, is best suited to shared editing tools. In order for the properties of independence of objects and operations to be met, the documents being edited must be structured in some way. A shared bitmap editor could not be implemented, because each pixel should be an object, which is not realistic. A shared text editor could be implemented, provided that the granularity of editing be defined. For instance, each sentence could be an object. The partial locking that occurs at the first step of the two-phase echo would then prevent two users from editing the same sentence at the same time, but a user would be able to edit a sentence while another changes its font or format. This is not completely satisfactory since the granularity is imposed by the replication algorithm and cannot be changed during the session. We are currently specifying a more general version of the algorithm in order to handle dependent objects and/or operations, so as to do away with these constraints.

## CONCLUSION AND FUTURE WORK

We have presented a real-time groupware system for the shared editing of structured diagrams. We have emphasized two important characteristics of this system: transparency of the system and awareness of the group. We consider these two properties as central to any real-time groupware editor. We have shown that the implementation of the system with a purely replicated architecture is both simple and powerful.

Preliminary user testing has shown that the features we propose, once explained to the users, are easily understood. Conflicts never happen in practice, except in time-relaxed WYSIWIS, because the protocol is lightweight compared to the network bandwidth. Finally, GroupDesign alone cannot support the whole coordination task. This led us to develop a separate application to allow users to communicate with each other by typing text in a shared window.

In our approach, we rely on the users to structure their group work around the features of the tools they use, in the same way users adapt their work to the applications they have at hand. We expect new working processes to emerge from the group that may not have been anticipated by an *a priori* analysis.

Our future work will take three directions. The first is the investigation of further general functionalities for real-time groupware. The second is the evaluation of these features in real-life settings. The last is the definition of a groupware toolkit to support the purely replicated architecture.

## ACKNOWLEDGMENTS

This work is partially supported by Apple France. We thank MetaSoftware for providing us with MetaDesign and Design/OA, and Heather Sacco for enhancing the readability of this article.

## REFERENCES

1. Apple Computer, Inside Macintosh, Volume VI, Addison Wesley, Reading, MA, 1991.
2. Baecker, R., New Paradigms for Computing in the Nineties. In *Proc. Graphics Interface '91*, (Calgary, Alberta, 3-7 June 1991), pp. 224-229.

3. von Biel, V., Groupware Grows Up. In *MacUser*, June 1991, pp. 207-211.
4. Birman, K., Cooper, R., Joseph, T., Kane, and K., Schmuck, F., The ISIS System Manual, June 1989.
5. Buxton, W., and Moran, T., Europarc's Integrated Interactive Intermedia Facility (IIIF): Early Experiences. In *Proc. IFIP WG8.4 Conference on Multi-User Interfaces and Applications* (Heraklion, Greece), S. Gibbs and A. A. Verrighn-Stuart (eds). North Holland, 1990.
6. Card, S. K., Mackinlay, J. D., and Robertson, G. G., The Information Visualiser, an Information Workspace. In *Proc. CHI'91* (New Orleans, LA, April 1991), pp. 181-188. ACM, New York, 1991.
7. Coleman, Dale, Timbuktu vs. Carbon Copy Mac: Close Race. In *MacWeek* (September 11, 1990), pp. 181-188.
8. Crowley, T., Milazzo, P., Baker, E., Forsdick, H., and Tomlinson, R., MMConf: An Infrastructure for Building Shared Multimedia Applications. In *Proc. Third Conference on Computer-Supported Cooperative Work* (Los Angeles, CA., October 1990). ACM, New York, 1990.
9. Meta Software Corporation, *Design/OA Manual*, 150 CambridgePark Drive, Cambridge, MA, March 1989.
10. Ellis, C.A., Gibbs, S.J., and Rein, G.L., Groupware Some Issues and Experiences. In *Communications of the ACM*, January 1991, Vol. 34, No 1, pp. 39-58.
11. Ellis, C.A., and Gibbs, S.J., Concurrency Control in Groupware Systems. In *Proc. ACM SIGMOD'89 Conference on the Management of Data*, (Seattle WA, May 1989). ACM, New York, 1990.
12. Elwart-Keys, M., Halonen, D., Horton, M., Kass, R., and Scott, P., User Interface Requirements for Face to Face Groupware. In *Proc. CHI'90* (Seattle, WA, April 1990), pp. 303-312. ACM, New York, 1990.
13. Gaver, W. W., Smith, R. B., and O'Shea, T., Effective Sounds in Complex Systems: The Arkola Simulation. In *Proc. CHI'91* (New Orleans, LA, April 1991), pp. 85-90. ACM, New York, 1991.
14. Gibbs, S. J., LIZA: An Extensible Groupware Toolkit. In *Proc. CHI'89* (Austin, TX, May 1989), pp. 29-35. ACM, New York, 1989.
15. Henninger, S., Computer Systems Supporting Cooperative Work: A CSCW'90 Trip Report. In *SIGCHI Bulletin*, July 1991, Vol. 23, No 3, pp. 25-28.
16. Knister, M. J., and Prakash, A., DistEdit: A Distributed Toolkit for Supporting Multiple Group Editors. In *Proc. Third Conference on Computer-Supported Cooperative Work* (Los Angeles, CA, October 1990). ACM, New York, 1990.
17. Lamport, L., Time, Clocks and the Ordering of Events in a Distributed System, *Communications of the ACM*, July 1978, Vol. 21, No. 7, pp. 558-565.
18. Lauwers, J. C., and Lantz, K., Collaboration Awareness in Support of Collaboration Transparency: Requirements for the Next Generation of Shared Window Systems. In *Proc. CHI'90*, (Seattle, WA, April 1990), pp. 303-312. ACM, New York, 1990.
19. Minneman, S. L., and Bly, S. A., Managing a Trois: a Study of a Multi-User Drawing Tool in Distributed Design Work. In *Proc. CHI'91* (New Orleans, LA, April 1991), pp. 217-224.
20. Patterson, J. F., Hill, R. D., and Rohall, S. L., Rendezvous: An Architecture for Synchronous Multi-User Applications. In *Proc. Third Conference on Computer-Supported Cooperative Work* (Los Angeles, CA, October 1990). ACM, New York, 1990.
21. Rein, G. L., and Ellis, C. A., rIBIS: A Real-Time Group Hypertext System. In *International Journal of Man Machine Studies*, Vol. 34, No 3, March 1991, pp. 349-368.
22. Stefik, M., Bobrow, D.G., Foster, G., Lanning, S., and Tartar, D. WYSIWIS revised: Early Experiences with Multiuser Interfaces. In *ACM Transactions on Office Information Systems*, Vol. 5, No 2, April 1987, pp. 147-186.
23. Stefik, M., Foster, G., Bobrow, D. G., Keneth, K., Lanning, S., and Suchman, L., Beyond the Chalkboard: Computer Support for Collaboration and Problem Solving in Meetings. In *Communications of the ACM*, January 1987, Vol. 30, No 1, pp. 32-47.