A System for Supporting and Managing Same-Time/Different-Place Group Interactions

Pedro A. Antunes

Department of Electrical and Computer Engineering
Instituto Superior Tecnico, Technical University of Lisbon, Portugal
Tel: +351.1.841-74-51, paa@digitais.ist.utl.pt

ABSTRACT

This paper describes a user-interface system developed to support group interactions for same-time/different-place cooperative applications. We address three fundamental aspects of these kind of systems: information sharing, coordination and multiuser-interface. The proposed approach defines four types of objects. *Contents* store application data. *Containers* are dedicated to organise and structure application data. *Connections* manage group coordination. And, finally, *Monitors* are concerned with users awareness of cooperative activities. One important characteristic of the approach is that it identifies and maps into the above objects two basic properties of group interaction support: visibility (public/private information) and durability (durable/transient information). The system eases the design of complex group interaction processes because it defines simple actions that allow programmers and users to define and combine object properties. An example of system usage is given for an application that supports brainstorming activities.

KEYWORDS: Group Interaction, CSCW.

INTRODUCTION

The computer support to group interaction consists of three basic functionalities: information sharing, coordination and multiuser-interface. Information sharing allows to establish a common context between individuals, a functionality that requires the specification of a data consistency model. Data consistency can be preserved through concurrency control mechanisms [5][12], e.g. locking, versioning, history, views, etc.

Group interaction adds the notion of interdependence [23][13] and coordination [24][22] to information sharing. Interdependence means that, in cooperative settings, activities flow from one individual to another, while coordination introduces the requirement of managing the dependencies between activities. Several coordination mechanisms have been proposed [31], e.g. free mechanisms, that rely on the social protocols established by users and do not control the access to the medium, floor-control, semi-formal, based on language and formal mechanisms.

The multiuser-interface is responsible for mediating users and the system. The multiuser-interface must define a public space, shared by all users, and maintain visual consistency of objects which are placed in the public space. It must also manage the interconnection of private and public spaces, since group activities are assembled from a mixture of private and public activities. One more multiuser-interface requirement exists: it must provide users awareness on cooperative activities [7][16][32].

The computer support to group interaction can also be characterised in time/space domains [18][30]. The combination of these domains defines four different types of systems: (1) same-time/same-place, which focus on the computer support to information sharing, since coordination and multiuser-interface can be established face-to-face; (2) different-time/different-place, which minimises multiuser-interface mechanisms, fundamentally because most work is done in the users' private spaces; (3) different-time/same-place, where few cooperative systems can be placed, minimises information sharing and coordination, emphasising single-user interface aspects of interactions; and (4) same-time/different-place, which requires the full spectrum of group interaction support.

In same-time/different-place systems, information sharing is necessary to preserve a shared context between users that are not face-to-face; coordination is essential to manage interventions by users that are simultaneously using the system; and multiuser-interface is essential to preserve the degree of co-presence of cooperative work.

This paper describes support to group interaction addressing in particular same-time/different-place applications. The system is based on a small set of graphical objects that ease the design of complex group interaction processes by hiding information sharing, coordination and multiuser-interface mechanisms through simple manipulations of objects properties. Cooperative applications are programmed by constraining objects properties and users manipulations.

The system has been used to address research issues at system level (e.g. group awareness of network partitions [15]) and also organisational level. At the organisational level, it has been used to implement and study the support to group-decision processes in distributed organisational settings, structured according to decision techniques developed in the social sciences field [3].

The paper is structured in the following way. We start by providing background information on our development effort. The following two sections describe the objects that define the multiuser-interface and support the coordination of users. Next, we dedicate one section to detail the support

Proceedings of the Working Conference on Advanced Visual Interfaces, AVI '98. L'Aquila, Italy: ACM Press, 1998, pp. 216-225.

to information sharing. Then, we provide some implementation details. An example application which uses the proposed system is also presented. Finally, we present some related work, conclusions and future lines of work.

BACKGROUND

The system described in this paper corresponds to the evolution of a set of various systems which have been developed since 1995 till today. Its origin can be attributed to a tool, named NGTool, which was developed in the context of two European Esprit projects (BROADCAST and ORCHESTRA) with the objective of supporting same-time/different-place group decision-making processes in the organisational context [1][2][3]. NGTool was built over a graph editing tool (from where it inherits the look and feel) to implement one particular decision-making technique developed in the social sciences field: the Nominal Group Technique [35][25]. NGTool was developed in the C++ language and Unix environment, had a replicated architecture, and support to information sharing was built over alternative systems, including a socket-based message broadcast system named MBus [21] and a set of reliable group communication systems: ISIS [11], HORUS [29] and NavTech/NavCoop [14][15].

In 1996, NGTool was ported from the Unix to the Windows environment. Furthermore, Most of the functionality related with the cited decision-making technique was generalised and expanded, giving origin to several distinct tools supporting brainstorming, ideas organisation, structured discussions and voting tasks. All these different tools used very similar interaction mechanisms, substantiating our idea that the system could provide generic interaction support for same-time/different-place cooperative sessions [4]. To this second implementation effort was given the name NGMeeting (Network Group Meeting). Given the experimental purposes, reliable communication was not considered important and though we used MBus again for message broadcasting.

The later implementation, developed in 1997 and presented here, results from the identification of several basic properties associated to multiuser-interface objects (visibility and durability, described later) and the definition of simple user actions over those objects, which although being simple can nevertheless be combined in order to create distinct and functional cooperative applications. The current implementation uses a framework for developing distributed applications named DASCo [33][34]. This framework uses an approach for object replication (with separation of concerns: distribution, persistency, concurrency, etc.) different from the group communication paradigm.

The experimental objectives we have followed have been centred on the support to group decision-making processes, their phases (such as forming, storming, norming and performing [41]) and tasks. The focus on group interaction results in less importance given to content and content creation, addressed by classes of systems such as document production environments (e.g. GroupDesk [19]), cooperative editors (e.g. Grove [18]) or group authoring tools (e.g. Dolphin [38]). The reader may later notice that these objectives are complementary and can be integrated in our approach in the future.

MULTIUSER-INTERFACE SUPPORT

We start by defining a set of objects which characterise the multi-user interface: Containers, Contents and Monitors.

Containers and Contents

Containers represent and structure application data. A Container is a functional object which allows users to manipulate its two components:

- A Content object, which stores application data.
- A representative component, which provides distinct visual marks concerning the Content type.

Content objects can be of type Simple, containing one data unit defined by the application, or of type Composed, containing a list of Containers. Content objects are visible to users through a presentation component. In the case of a Simple Content, the presentation component takes the form of a small window, mediating user accesses to the application data, while in the case of a Composed Content it takes the form of a set of arrows linking Containers.

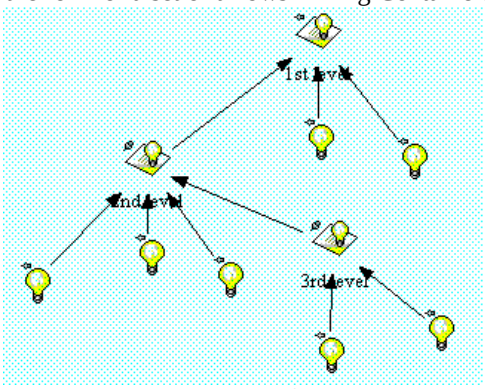

Fig. 1 - Organisation of Containers

The distinction between Containers and Contents accomplishes several purposes. First, it separates application data units from their visual structures and representatives, thus allowing to share application data between users while relaxing graphical properties such as positions or motions in display space. Second, we can preserve display space by allowing users to hide application data units. Figure 1 illustrates how data structures can be represented without showing contents. Finally, the approach focus users' interactions over Containers rather than Contents, avoiding this way distracting users with too many modifications over shared data.

Dynamic Properties of Containers and Contents

Both Container and Content objects present two dynamic properties:

- **Visibility**: public/private
  Objects can be public or private. Private objects support individual activities while public objects are dedicated to support group activities based on coordination and information sharing. Public objects follow a WYSIWIS

(What You See Is What I See) multiuser-interface semantics [37] with the following relaxation: (1) allows spatial discrepancies (users see the same object at different positions); (2) allows presentation discrepancies (the same object may be viewable or hidden to users); and (3) allows time discrepancies. The time relaxation ensures that, although many temporary inconsistencies may arise in the individual perception of content modifications (due to communication delays in the updates), the information that users get in the end is the same.

- **Durability**: durable/transient
  Transient objects manage information intended to disappear after some user manipulation. On the contrary, durable objects manage information that remains after users manipulations. When transient, Containers and Contents can be deleted, moved or reorganised between Contents (between Composed Contents, in the case of transient Containers, to be exact). When durable, Containers and Contents cannot be deleted, moved or reorganised. Figure 2 presents the durability properties for the hierarchy of objects defined by the system.

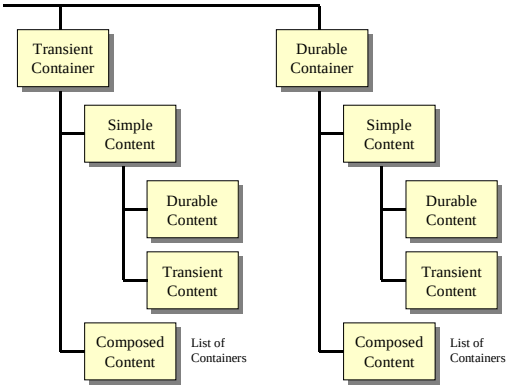

Fig. 2 - Durability property of objects

Public Containers have two more properties:

- **Modification**: modified/unmodified
  This property is necessary to provide users awareness on the modification of Content objects. The property is set whenever users update a Content that is hierarchically bellow a Container.
- **Control**: free/editing/conflict/resolving
  This property identifies a set of states associated to information sharing and concurrency control. A description of the concurrency control mechanisms applied to public objects is given later.

Monitors

Monitors are graphical objects that provide awareness of users' activities and system operations. These objects consist of animated icons that may appear and disappear according to the conditions they are associated to. A Monitor always appears close to a Container and reports on its properties:

- **Visibility**: No monitor is defined for this property (private/public visibility cues are given spatially, as described later).
- **Durability**: A Monitor with the look of a pin informs on the current object state, either durable or transient.

- **Modification**: A Monitor with the look of a glass magnifier identifies if the Content object (including Composed Contents) has been modified by a user.

- **Control**: A Monitor with the look of a semaphore informs on the state of the protocol which manages information sharing and concurrency control (green, the user may edit data; yellow, conflicts may arise; red, the user cannot edit data).

Some Design Options and Observations

This section describes some practical decisions which are independent from the multiuser-interface support described above.

Private and public spaces. We decided to identify the visibility property of Containers and Contents using spatial information, defining different spaces for private and public objects. The application window is divided by a vertical line in two different areas: to the left users find their private space while to the right they find the public space. Furthermore, we restricted the Content objects visibility properties. Contents, including Composed Contents, always inherit their Containers visibility states, which means that hierarchies of Containers cannot cross different spaces.

Containers. We use an icon and an optional label to represent the Content associated to the Container. The durability of a Container can be modified by clicking over the Monitor (pin). Users can show/hide associated Contents by double-clicking on Containers. A Container may change the displayed icon when showing/hiding Contents.

Contents. In the current version of the system, Simple Contents are implemented by a dialog box defined by the Microsoft's Foundation Classes. The Content data is only modified after a user presses the "accept" button. The durability of a Content can be modified by pressing a button (displaying a pin) available in the dialog box. Composed Contents are presented to users as trees of Containers, with directed lines connected to the associated Container (see Figure 1). In future implementations an hypertext approach will also be supported.

Some Observations. We have experimented using the described objects with multiple applications:

- Organise ideas in space, according to perceived relationships.
- Discussion topics, enclosing users positions, arguments and comments.
- Votes concerning different topics.
- Cognitive maps [17].

Casual observations of system usage and discussions with users showed that, although no definition of Container and

Content objects were given, the user interface is easily understood and accepted. It must be noted however that the tasks observed considered mostly private activities with public entries of small phrases (ideas, comments, etc.).

We also experimented the system support to highly focussed interactions, where users are expected to converge to the same information. This was implemented by modifying the public Container to open its associated Content to all users and requiring users to press an "accept" button. The approach, however, was a partial failure, not necessarily due to communication delays in large-scale settings, but mostly because users took time to switch from private to public activities and read the information. This observation makes difficult to implement some decision-making techniques that rely on face-to-face interactions, such as the Nominal Group Technique. These techniques have to be modified in order to increase private work.

Different forms of relaxing objects' graphical properties were experimented, for instance, keeping positions in the public space consistent. This, however, resulted in loss of users focus, specially with large object hierarchies. On the contrary, the individual positioning of objects preserves personal spatial references and avoids the need for constantly displaying application data.

COORDINATION SUPPORT

Coordination is addressed by one more type of object defined by the system: the Connection.

Connections

Connections are dynamic objects that structure users interactions over pairs of Containers. Part of the semantics of these objects depends on properties associated to the pair of Containers. A Connection has three components:

- **Origin**: a Container from which the Connection is started.
- **Destination**: a Container where the Connection finishes. If no destination is identified, the system creates one.
- **Transfer**: a temporary Container (may be the origin) which allows to transfer information between origin and destination.

Connections are established by users with drag-and-drop operations. Then, the Connection proceeds in two phases: (1) the initialisation phase, where the temporary Container is created and the origin object is modified according to its properties; and (2) the finalisation, where the temporary Container is removed and the destination object is modified according to its properties. After these two steps, the Connection has accomplished its task and vanishes.

The association between a Connection and the durability property of origin objects is the following (see Figure 3). A transient origin is removed from its Composed Content (if any) by the Connection and becomes the transfer object. If the origin is durable, the Connection preserves the origin and creates a copy of it, that becomes the transfer. Two other alternatives must be considered if the origin is durable: the Content associated to the origin is also durable or, on the contrary, is transient. If the Content is durable, it is preserved, but, if the Content is transient, it is made empty.

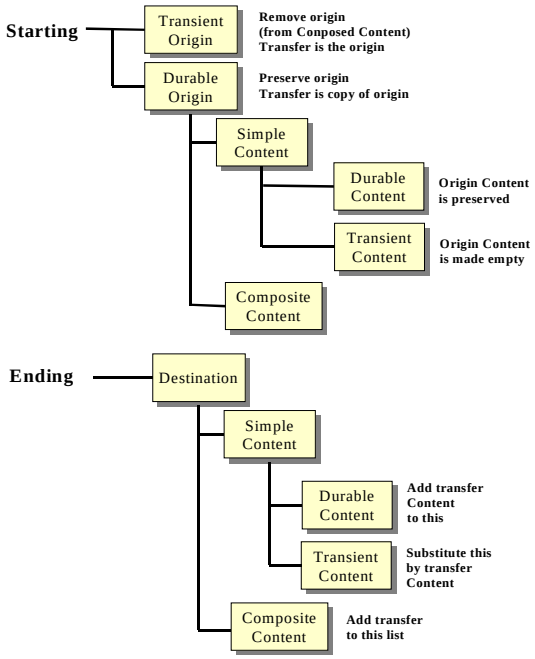

Fig. 3 - Connections and the durability of objects

Now, concerning the destination object. A destination has either a Simple or Composed Content. If Simple and transient, the Connection substitutes its Content by the transfer Content. If Simple and durable, the Connection appends the transfer data to the destination Content. Finally, if the destination is Composed, the Connection adds the transfer to the list of Containers. Figure 3 presents this various actions associated to a Connection.

Connections have also implications in the visibility property of objects. Since application data has a public status in public objects and a private status in private objects, Connections allow users to transfer objects and data across different work spheres. Table 1 presents several examples of Connections and the corresponding functionality.

The functionality of Connections, combined with durability and visibility properties of Containers and Contents, provides the system support to coordination. One particular aspect of the approach is that it avoids the definition of a specific coordination mechanism, but allows the implementation of different coordination mechanisms at the application level. To substantiate this statement, we present some examples that we have experimented so far:

- Establish how data can be created in the public space (restrict the origin objects and require that data be first created in the private space).
- Establish how data can be organised in the public space (restrict the destination objects).
- Establish how data can be transferred from the public space to private spaces (avoid or require that public data be removed).

Origin	Destination	Functionality
Durable Container with Simple Content	Container with Simple Content	Copies data from the origin to the destination
Transient Container with Simple Content	Container with Simple Content	Removes data from the origin and migrates it to the destination
Container with Simple Content	Transient Container with Simple Content	Substitutes previous data at the destination with the origin data
Container with Simple Content	Durable Container with Simple Content	Appends origin data to the destination
Durable Container with Simple Content	Container with Composed Content	Duplicates data, generating a new Container at the destination
Transient Container with Composed Content	Container with Composed Content	Reorganises data, moving the origin Container to the destination
Private and transient Container	Public Container	Migrate data to the public sphere
Public and durable Container	Private and transient Container	Copy data to the private sphere

Tab. 1 - Examples of Connections

- Define who produces and consumes data in the public space (identifying who can establish Connections).
- Define specific roles, for instance, the hierarchy of public objects may only be modified by an editor (verification of who establishes Connections).
- Define protocols, using intermediate objects, e.g. users may send data to an object in the public space from which only a moderator can move data out (restrict Connections to and from that object).

As it may be inferred from the examples above, the implementation of coordination mechanisms at the application level consists on the specification of restrictions applied to Containers:

- Restrictions to dynamic property changes (e.g. they can only be private and durable).
- If they can be used or not as origin and/or destination for Connections.
- Which specific objects they can be connected from/to.
- Which specific users can establish Connections from/to.

INFORMATION SHARING

In our system, information is shared by replicating all objects which have the public property set. The system component that supports object replication and handles concurrency control over distributed replicas is therefore associated to public Contents and Containers. We identify three situations where conflicts due to concurrent operations by users can occur, and present the solutions adopted in the system:

- *Atomic operations, such as modifying the durability or visibility properties of public Containers and Contents.*
  Our approach relies on the serialisation of messages which are broadcasted to all replicas in order to inform on property changes. No concurrency control is performed by the system: all replicas acquire the latest property change performed by any user.
- *Operations associated to coordination, i.e. establishment of connections between Containers.*
  The probability of occurrence of such conflicts is low, since they happen when users select the same objects as origin or destination of Connections, and Connections vanish im-mediately after these operations. However, the probability of conflicts increases with large hierarchies of Containers, since some Connections may involve all children in the hierarchy. A semi-optimistic approach for concurrency control was selected: we allow users to freely establish Connections but immediately suspend the operation if a conflict is detected. If a conflicting Connection is completed locally by the user (because no conflicting information was received by the local replica during the drag-and-drop) the local operation is rolled back and the initial state is recovered.

  As previously noted, Connections established from or to Containers with Composed Contents may involve a large set of Containers. Under those circumstances, the system detects possible conflicts by: (1) at the initialisation phase, starting at the origin object, checking up and down all objects referenced by Composed Contents; and (2) at the finalisation phase, starting at the destination object, checking up and down all objects referenced by Composed Contents. If any object checked is origin of a Connection or is a Simple Content being edited, the Connection is aborted.
- *Editing operations, concerning Contents modifications.*

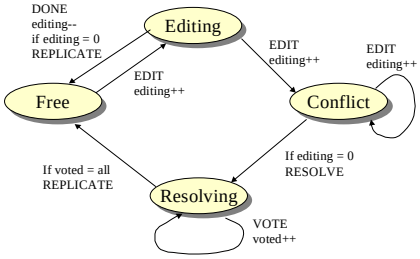

Fig. 4 - States machine for Content editing

For Content editing, we use an optimistic approach based on a versioning mechanism: users are allowed to freely edit the public Content. If a conflict is detected, different versions are generated by the system, and users are later requested to vote on which version should be selected (a majority rule is used). Figure 4 presents the state machine used to handle editing operations.

IMPLEMENTATION

Figure 5 presents the UML (Unified Modelling Language [28]) diagram of the classes that implement Contents, Containers and Connections associated to multiuser-interface and coordination support.

The following classes are implemented:

- Container: The visibility and durability properties allow to define coordination and information sharing. The methods preLinkAsOrigin and postLinkAsDestiny indicate that the Container participates, respectively, as origin or destination of a Connection. The method generate implements the functionality that allows a Container to generate other Containers. This method is used when a Connection does not identify a destination object. Methods openContentView and CloseContent View allow users to open or close the associated Content object.
- Content: Has two derived classes:

- SimpleContent: Is a wrapper for a dialog box from Microsoft's Foundation Classes. Implements the Clone method for duplicating objects.
- ComposedContent: Implements the list of Containers. Has algorithms for placing new Containers, avoiding juxtapositions, and also for opening/closing the listed Containers as a group.

- Connection: Receives the identification of origin and destination Containers. Method getDestiny creates a destination when it is not identified by users. The connectionAllowed method verifies if the connection is valid. This method can be redefined in order to implement specific coordination mechanisms for each application.

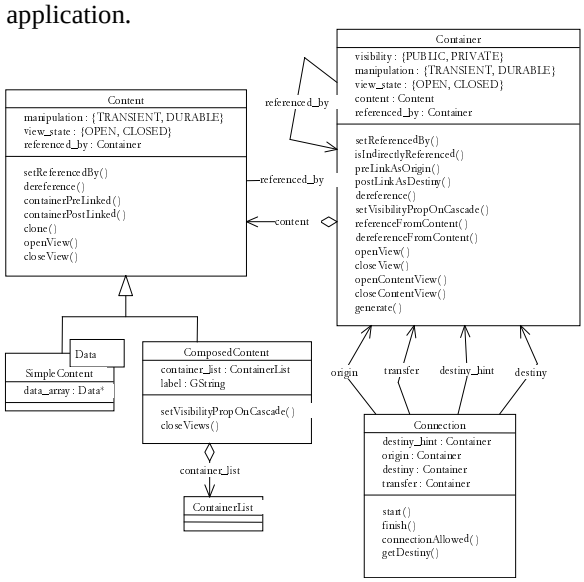

Fig. 5 - UML description of system classes

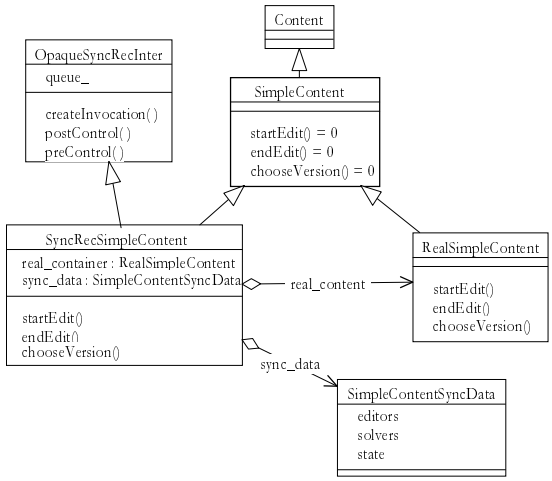

Fig. 6 - UML description of information sharing classes

Support to object replication and concurrency control is implemented at the Simple Content object, and operates only when the visibility property has the public state. The following objects are involved (see Figure 6):

- SimpleContent: Derived from Content, it defines startEdit and endEdit for starting and finishing content editing.
- RealSimpleContent: Implements the SimpleContent without object replication. It is used when the Content visibility property is private.
- SyncRecSimpleContent: Implements object replication and concurrency control using data stored by SimpleContent SyncData. It maps startEdit and endEdit into preControl and postControl. These two methods are provided by the DASCo framework for starting and finishing replicated object invocations. The method preControl does not perform concurrency control, which would be pessimistic, but informs if a possible conflict is detected.

Architectural Details

The current version of the system is based on a replicated architecture, where one client version of the system is instantiated at each user's machine running the Window 95/NT operating system. As previous referred, public objects are replicated at each client site. A server must be running in order to support object replication and invocation. Basically, the server receives information from one replica and disseminates that information to the other replicas. Client/server communication uses TCP/IP sockets. The server runs the DASCo framework, dedicated to support object invocation, serialisation, concurrency and concurrency control over invocations. Due to DASCo specific requirements, the server must run on a Unix machine.

EXAMPLE

In order to illustrate the system previously described in usage, we present an example application. This application is intended to allow users to generate and organise ideas, using a technique similar to the brainstorming technique [20][26]. The requirements of this application are:

- A public space is necessary to increase synergy, by allowing users to observe ideas already generated by the group.
- Users can freely generate ideas in their private or public spaces. Ideas can be moved from private to public spaces.
- Ideas can be structured in folders.
- In the public space, users can freely move ideas around folders, remove them from folders or change their contents, independently of their author.

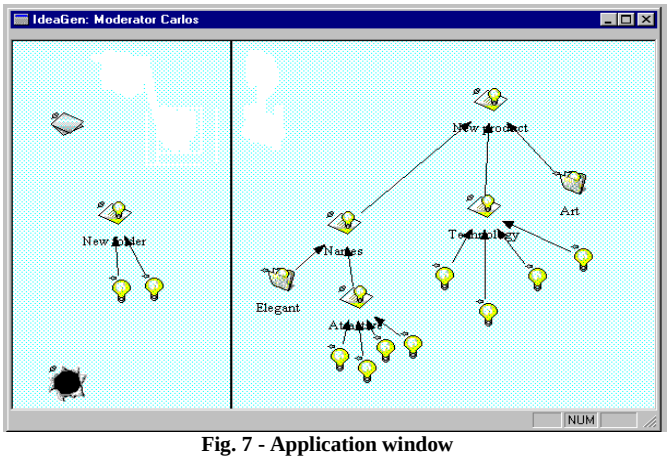

Fig. 7 - Application window

The implementation of this tool required the definition of the following objects (see Figure 7):

- *A Container dedicated to generate folders for storing ideas (top left).*
  This Container is configured to not be movable, be permanently durable, and do not accept connections as a destination.
- *A Container of type folder (e.g. top right).*
  This is a standard Container with a Composed Content. It has two icons, a "briefcase" when the list of ideas is hidden, and a "light bulb over a paper" when the list of ideas is visible.
- *A Container of type idea (e.g. bottom right).*
  This is a standard Container with a Simple Content where users write one idea. It has a "light bulb" icon.
- *A Container for deleting ideas or folders (bottom left).*
  This container has a "black hole" icon and is configured to be permanently durable and only accept connections as destination. Origin objects are deleted independently of their durability.

Figure 8 illustrates how users create ideas from folders in private spaces. The user establishes a connection from the folder into the private space (no destination). Given that the folder is durable, the connection creates a new Container of type idea.

Figures 9 and 10 illustrate the usage of the durability property. In Figure 9, the connection of a transient idea moves the idea in space while, in Figure 10, the connection of a durable idea creates a new idea in space.

Figure 11 displays ideas which have been moved to the public space. Two users, Carlos at the left and Daniel at the right can see differently the same information. Carlos is editing one idea while Daniel sees the folder closed. Figure 12 shows that Carlos finished editing and so Daniel sees the modification Monitor. In Figure 13, Daniel opens the folder to see which idea was edited.

Figures 14 to 16 illustrate how the system handles concurrency control. In Figure 14, Carlos starts editing and idea. Daniel, in Figure 15, starts editing the same idea. Carlos and Daniel are informed that a possible conflict exists but are allowed to proceed editing. The system creates versions and waits for both users to finish editing. Finally, in Figure 16, users vote on which version should be selected.

RELATED WORK

A set of recent works can be related with some aspects of the system presented in this paper:

Visual Obliq. Is a GUI-builder for distributed multiuser applications [10]. Interaction with Visual Obliq is based on forms, and the unit of distribution is the form. Aspects pertaining to distributed computing, replication, sharing and communication, are explicit to Visual Obliq programmers.

MEAD. Is a system that supports the creation of multiuser interfaces [9]. The basic system component of MEAD is the User Display Agent which manages information display and multiple user interactions at the graphical object level. MEAD does not supply support to coordination mechanisms, as defined by Connections.

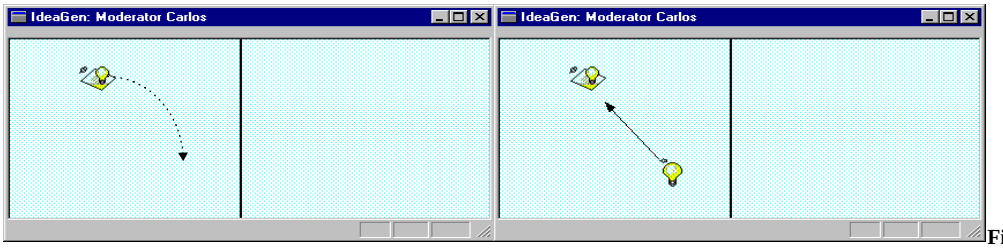

Fig. 8 - Create ideas from folders

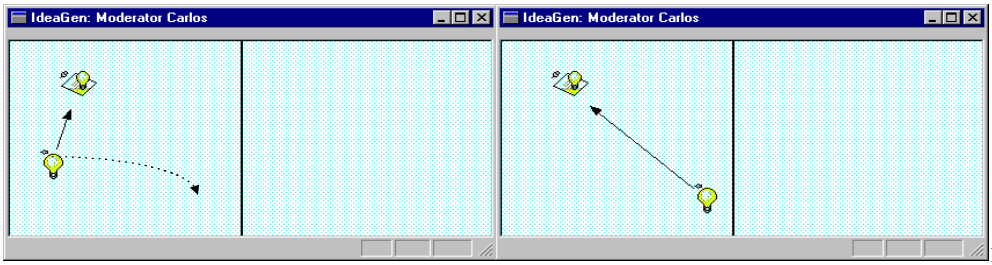

Fig. 9 - Move ideas

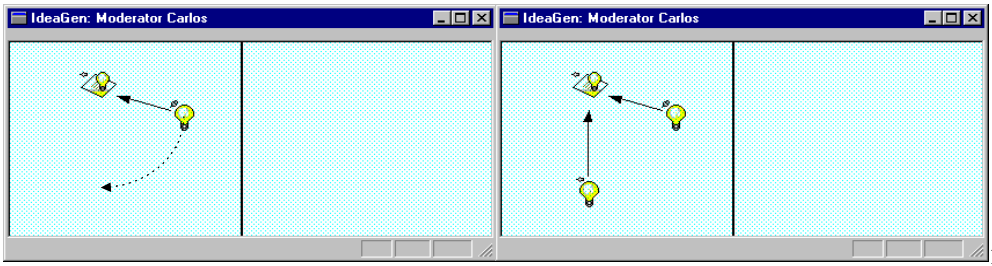

Fig.10 - Create ideas from other ideas

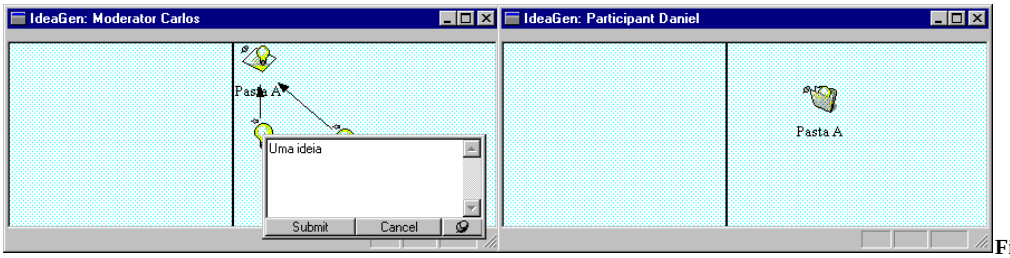

Fig. 11 - Ideas in public space

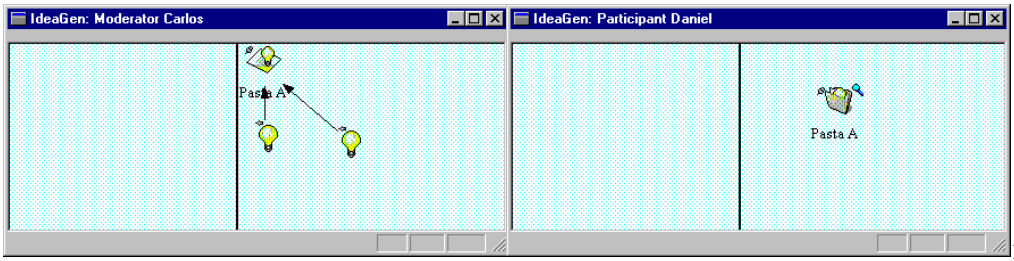

Fig. 12 - Monitoring modifications

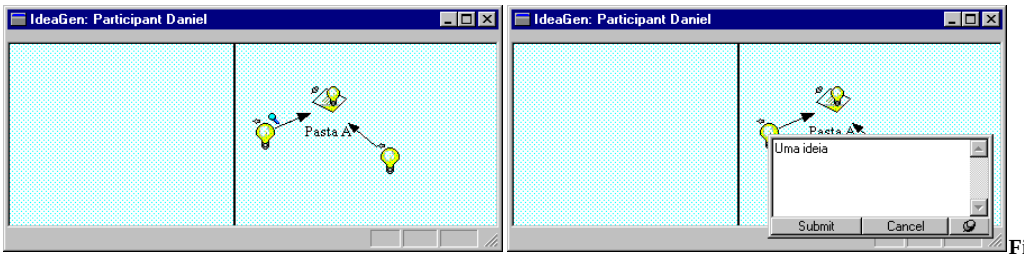

Fig. 13 - Looking for modification

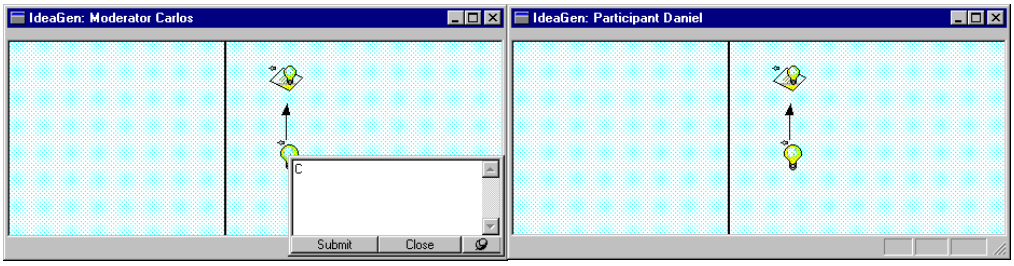

Fig. 14 - User starts editing Content

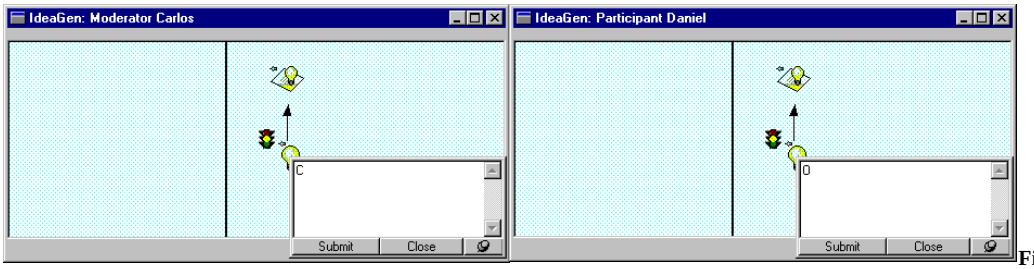

Fig. 15 - Another user starts editing

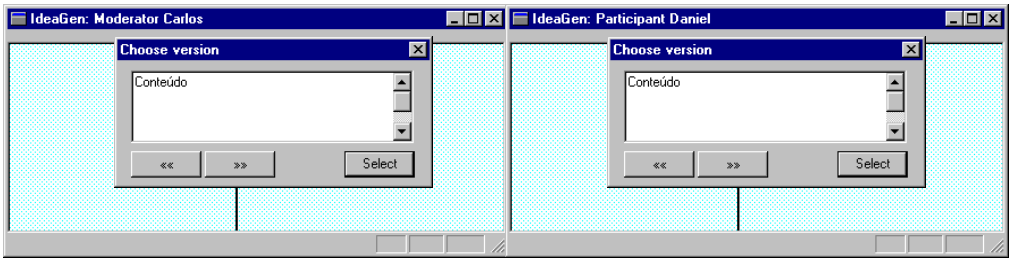

Fig. 16 - Choosing content version

CoSARA. Is a platform to specify and prototype multiuser interactions [39]. The focus is on data sharing (implementing several concurrency control protocols). The system also provides activity monitoring. The interaction mechanisms are defined at programming level, requiring knowledge of intricate specification models. CoSARA does not address coordination at the multiuser-interface level.

COLA. Is a system that supports cooperative work based on a model which defines activities (cooperation), roles (coordination) and events (awareness) [40]. COLA does not address the multiuser-interface level of group interaction support.

DIVA. Is an environment for group work that provides support for communication, cooperation and awareness [36]. DIVA uses a virtual office abstraction, based on rooms, desks and documents types of objects. DIVA is aimed at the integration of different CSCW tools, rather than support to cooperative applications. DIVA provides group awareness on users activities in the system: users change activities and coupling by moving through the virtual office, i.e. dragging icons around rooms and desks. The system shares the information space and establishes audio/video channels for users at the same desk.

DIVE/CyCo. Are, respectively, three and two dimensional systems that support cooperative interactions [8]. These systems use a spatial model that defines levels of awareness based on spatial metrics (orientation and distance). Levels of awareness are used to fire different events. For instance, a far object is able to notice a shareable piece of text; becoming closer, the object is able to read text; even closer the object may be able to cooperatively manipulate text. Group interaction is defined by moving objects.

Ramonamap. Is an interactive map that serves as interface to a shared database [6]. The way Ramonamap displays resources, using iconic information, is comparable to the one provided by Containers/Contents. No details are given on multiuser-interface, coordination or information sharing functionality.

GroupDesk. Is an environment for the coordination of cooperative document production that addresses the awareness requirement [19]. Awareness support is based on the distribution of object manipulation events and user registering of interests on particular events.

DOLPHIN. Is a groupware application that provides computer support to group meetings [38]. It defines private and shared spaces. The shared space supports whiteboard usage (gestures). DOLPHIN provides hypermedia objects and accepts pen-based inputs. Object transfers between public and private spaces use clipboard cut/paste operations. In remote cooperations, awareness is only supported through dedicated audio/video channels.

ESC. Is a text-based environment that demonstrates the use of innovative communication channels for cooperation [27]. In some sense, one Container/Content pair may implement a ESC channel.

CONCLUSION AND FUTURE WORK

In this paper we present a system which supports the construction of applications and tools for same-time/different-place group interactions. The system addresses three common functionalities of this kind of applications: information sharing, coordination and multiuser-interface aspects.

The proposed system is based on four object types: Contents, Containers, Connections and Monitors. Contents store application data while Containers represent and structure application data. Connections are dedicated to address users coordination: they allow users to manipulate Containers and Contents in a structured and cooperative way. Finally, Monitors are dedicated to provide users awareness concerning the actions performed with the system.

Two properties are fundamental to the functionality of Contents and Containers: visibility and durability. The visibility property addresses information sharing. Public objects are shared by all users while private objects can only be manipulated by one user. The visibility property hides the necessary support to object replication and concurrency control. The durability property allows users to define dynamically how information is manipulated in the system: durable objects are intended to persist in the system after being manipulated while transient object are intended to be combined into other objects or disappear.

The combination of visibility and durability properties with the functionality of Contents, Containers and Connections provides a wide – yet simple – range of group interactions. Although the system does not implement any specific coordination mechanism, generic support is provided allowing application programmers to construct such mechanisms by specifying constraints to objects defined by the system. Details concerning public spaces and visual consistency, interconnection of public and private spaces, object communication, replication and concurrency control are removed from the application programmers' concerns.

The paper illustrates the above comments with an example application. The application, which is intended to support a social process commonly known as brainstorming, allowing users to generate, modify and organise their ideas, requires minimum programming to accomplish its objectives.

We should however recognise that several issues could have been, but were not, addressed by the system. Notably, an hypertext class should have been derived from the Composite Content, supporting one more powerful way of organising application data. Also, Simple Contents are currently of type text only, although, for instance, a drawing object would be useful for implementing cooperative design applications.

Other aspects that were not considered but will be addressed in the future are:

- Remove the private/public space dichotomy, i.e. avoid the identification of public and private properties based on spatial information of objects.
- Allow private Contents inside public Containers and vice-versa.

And finally, we intend to address the issue of visually programming the constraints associated to system objects. The intention is to allow users to create and personalise on-line their own objects of cooperation, in a scenario

characterised by long (and possibly different-time/different-place) cooperative sessions.

ACKNOWLEDGEMENTS

The author wishes to acknowledge and thank the contributions for this work from: Tânia Ho, Carlos Alves and Daniel Silva.

REFERENCES

[1] P. Antunes and N. Guimaraes. NGTool - Exploring Mechanisms of Support to Interaction. In 1st CY-TED-RITOS Intern. Works. on Groupware - CRIWG '95. CYTED-RITOS. Lisboa, Portugal. Sep. 1995.

[2] P. Antunes and N. Guimaraes. Structuring Elements for Group Interaction. In 2nd Conf. on Concurrent Engineering, Research and Applications (CE95). Concurrent Tech. Corp. Washington, DC. Aug. 1995.

[3] P. Antunes, N. Guimaraes, J. Cardenosa and J. Segovia. Beyond Formal processes: Augmenting Workflow with Group Interaction Techniques. In Conf. on Organisational Computer Systems - COOCS '95. ACM. Aug. 1995.

[4] P. Antunes and N. Guimaraes. User-Interface Support to Group Interaction. In 2nd CYTED-RITOS Intern. Works. on Groupware - CRIWG '96. CYTED-RITOS. Puerto Varas, Chile. Sep. 1996.

[5] N. Barghouti and G. Kaiser. Concurrency Control in Advanced Database Systems. ACM Computing Surveys, 23(3). Sep. 1991.

[6] J. Bartlett. Ramonamap - An Example of Graphical Groupware. In Proc. of the ACM Symposium on User Interface Software and Technology - UIST '94. Marina Del Rey, California. Nov. 1994.

[7] M. Beaudouin-Lafon and A. Karsenty. Transparency and Awareness in a Real-Time Groupware System. In Proc. of the ACM Symposium on User Interface Software and Tech. Monterey, Cal. Nov. 1992.

[8] S. Benford. A Spatial Model of Interaction in Large Virtual Environments. In Proc. of the 3rd European Conf. on Computer-Supported Cooperative Work - ECSCW '93. Milan. Sep. 1993.

[9] R. Bentley, T. Rodden, P. Sawyer and I. Sommerville. Architectural Support for Cooperative Multiuser Interfaces. IEEE Computer. May 1994.

[10] K. Bharat and M. Brown. Building Distributed, Multi-User Applications by Direct Manipulation. In Proc. of the ACM Symposium on User Interface Software and Technology - UIST '94. Marina Del Rey, California. Nov. 1994.

[11] K. Birman, T. Joseph and F. Schmuck. Isis - A Distributed Programming Environment, User's Guide and Reference Manual. Tech. Rep., Dept. of Computer Science, Cornell Univ. Ithaca. Mar. 1988.

[12] G. Blair and T. Rodden. The Opportunities and Challenges of CSCW. Journal of Brazilian Computer Society, 1(1). Jul. 1994.

[13] R. Butler. Designing Organizations. Routledge. 1991.

[14] F. Cosquer, L. Rodrigues and P. Verissimo. Using Tailored Failure Suspectors to Support Distributed Cooperative Applications. In Proc. of the 7th Intern. Conf. on Parallel and Distributed Computing and Systems. Washington, DC. Oct. 1995.

[15] F. Cosquer, P. Antunes and P. Verissimo. Enhancing Dependability of Cooperative Applications in Partitionable Environments. Proc. of the 2nd European Dependable Computing Conf. (EDDC-2). Taormina, Italy. Oct. 1996.

[16] P. Dourish and V. Bellotti. Awareness and Coordination in Shared Workspaces. In Proc. of ACM CSCW '92 Conf. on Computer-Supported Cooperative Work. Toronto, Canada. Nov. 1992.

[17] C. Eden. Strategy Development and Implementation: Cognitive Mapping for Group Support. In Strategic Thinking: Leadership and the Management of Change. John Wiley & Sons, Ltd. 1993.

[18] C. Ellis, S. Gibbs and G. Rein. Groupware: Some Issues and Experiences. Comm. of the ACM, 34(1). 1991.

[19] L. Fuchs, U. Pankoke-Babatz and W. Prinz. Supporting Cooperative Awareness with Local Event Mechanisms: The GroupDesk System. In Proc. of the 4th European Conf. on Computer-Supported Cooperative Work - ECSCW '95. Stockholm, Swe-den. Sep. 1995.

[20] C. Hwang and M. Lin. Group Decision Making under Multiple Criteria. Springer-Verlag. 1987.

[21] S. Kaplan, A. Carrol and K. MacGregor. Supporting Collaborative Processes with ConversationBuilder. In Conf. on Organizational Computing Systems. Atlanta, Georgia. Nov. 1991.

[22] R. Kraut and L. Streeter. Coordination in Software Development. Comm. of the ACM, 38. Mar. 1995.

[23] T. Malone and K. Crowston. What is Coordination Theory and How Can it Help Design Cooperative Work Systems? In Proc. of the Conf. on Computer Supported Cooperative Work (CSCW '90). Los Angeles, Cal. Oct. 1990.

[24] T. Malone and K. Crowston. The Interdisciplinary Study of Coordination. ACM Computing Surveys, 26(1). Mar. 1994.

[25] C. Moore. Group Techniques for Idea Building. SAGE Publications. 1994.

[26] J. Nunamaker, L. Applegate, and B. Konsynski. Facilitating Group Creativity: Experience with a Group Decision Support System. Journal of Management Information Systems, 3(4). 1987.

[27] D. Patel and S. Kalter. Low Overhead, Loosely Coupled Communication Channels in Collaboration. In Proc. of the 3rd European Conf. on Computer-Supported Cooperative Work - ECSCW '93. Milan. Sep. 1993.

[28] Rational Software Corporation. Unified Modelling Language v 1.1c – Notation Guide. World-Wide-Web http://www.rational.com. Jul. 1997.

[29] R. Renesse et al. The Horus System. Tech. Rep. Cornell Univ. Jul. 1993.

[30] T. Rodden. A Survey of CSCW Systems. Interacting With Computers, 3(3). 1991.

[31] T. Rodden and G. Blair. CSCW and Distributed Systems: The Problem of Control. In Proc. of the 2nd European Conf. on Computer Supported Cooperative Work - ECSCW '91. Amsterdam. 1991.

[32] T. Rodden. Populating the Application: A Model of Awareness for Cooperative Applications. In ACM 1996 Conf. on Computer Supported Cooperative Work - CSCW '96. Cambridge, Mass. Nov. 1996.

[33] A. Silva, L. Gil and J. Martins. Three-Layered Framework with Separation of Concerns. In OOPSLA '96 Works. on Exploration of Framework Design Principles. San Jose, Cal. Oct. 1996.

[34] A. Silva. Development and Extension of Frameworks. In Handbook of Object Technology. Saba Zamir Editor. CRC Press. 1998.

[35] D. Sink. Using the Nominal Group Technique Effectively. National Prod. Review. Spring 1983

[36] M. Sohlenkamp and G. Chwelos. Integrating Communication, Cooperation, and Awareness: The DIVA Virtual Office Environment. In ACM 1994 Conf. on Computer Supported Cooperative Work - CSCW '94. Chapel Hill, North Carolina. Oct. 1994.

[37] M. Stefik et al. Beyond the Chalkboard: Computer Support for Collaboration and Problem Solving in Meetings. Comm. of the ACM, 30(1). 1987.

[38] N. Streitz, J. Geissler, J. Haake and J. Hol. DOLPHIN: Integrated Meeting Support Across Local and Remote Desktop Environments and Liveboards. In ACM 1994 Conf. on Computer Supported Cooperative Work - CSCW '94. Chapel Hill, North Carolina. Oct. 1994.

[39] I. Tou et al. Prototyping Synchronous Group Applications. IEEE Computer. May 1994.

[40] J. Trevor, T. Rodden and G. Blair. COLA: A Lightweight Platform for CSCW. In Proc. of the 3rd European Conf. on Computer-Supported Cooperative Work - ECSCW '93. Milan. Sep. 1993.

[41] B. Tuckman. Development Sequence in Small Groups. Psychological Bulletin. 1965.