Fuks, H. & Moura L. (1994) A Document Based Approach for Cooperation, Journal of the Brazilian Computer Society, V1, N1, Julho 1994, pp. 36-45

A Document Based Approach for Cooperation

Hugo Fuks

Leonardo Mendonça de Moura

hugo@inf.puc-rio.br

Departamento de Informática

Pontifícia Universidade Católica do Rio de Janeiro

Rua Marquês de São Vicente, 255 - CEP 22453-900

Rio de Janeiro - RJ - Brasil

Abstract

This article outlines a document based approach for cooperation. It proposes a cooperation model, which states the way team members access the documents of a project. It is the underlying model for a cooperative system, where developers concentrate on the application's domain, ignoring the aspects related to the cooperative process.

Keywords: cooperative work, documents,

1. Introduction

This research is oriented towards developing an Electronic Meeting Room—a cooperative working environment. Our two main objectives are to provide desktop conferencing support and whiteboard meetings tools—synchronous groupware—for working together within the scope of a project. Inspiration for the former comes mostly from the GroupKit [5, 16], and for the latter, from Liveboard [3] and Colab [18].

Before starting the development of cooperative applications for this environment, a cooperation model is proposed. This paper outlines a document based approach for cooperation, which states explicitly the way team members access project documents. It is the underlying model for a cooperative system where developers concentrate on the application's domain, ignoring aspects related to the cooperative process such as access control, locks, document creation. The examples presented are from cooperative writing and drawing, although the cooperative system encompasses more facilities.

The paper is organized as follows. The second section concerns the cooperation model, and describes the access control methods, together with other features of the cooperative system. The third section presents an object oriented framework for the document based approach for cooperation. The fourth section shows the cooperative system architecture and a few implementation issues. Finally, the conclusion and future work are presented.

2. The Cooperation Model

The document based approach for cooperation leads to a cooperation model. The purpose of this model is to relate people to documents, for example a group of people working together to write a paper. People are inherently different, and working together implies that they must cooperate performing a variety of roles to achieve their aim. With this model it is necessary to define each role's relationship to the document.

A cooperative system is proposed to accommodate the document based approach for cooperation. The cooperative system comprises members, teams, projects and cooperative documents, as well as demonstrates their relationship. A team has a set of members (which is a subset of members of the system), and a set of projects that will be executed by the team. There are

two especial members, namely the system administrator and the head of the team [8]. A project consists of a set of cooperative documents.

During project execution—which involves document editing—members perform different roles. There are three possible roles to be performed in document editing: author, commentator and reader. The author owns the document, and may do whatever he wishes with it. The commentator may only make comments and offer suggestions about the document to its authors, who can accept them or not. The reader may only read the document.

There are various types of documents, e.g, graphics and text, therefore, there must also exist many kinds of applications in the cooperative system. In this way, each document may have one (or more) specific applications assigned to it.

2.1. Team Formation

The head of the team is assigned by the cooperative system administrator—whose sole task is to authorize the formation of new teams. The head takes responsibility for all team's actions, enrolls team members and defines the access control hierarchy. He is also responsible for project creation and for defining members who are allowed to create documents. These members and the head of the team are in charge of role definition for a specific document. For example, in computer supported cooperative design [4], the project leader - an author - will be the head of the team, the technical writer will be an author as well, while the consultants will be commentators. The access control hierarchy defines the way team members access project documents.

2.2. Cooperative Documents

Every access control structure is based on the notion of minimal unit. It is difficult to give a precise definition to the concept of minimal unit, but one could think of it as a part of a document—an object—to which ownership may be given. Minimal units vary according to the application, such as a paragraph for a word processor or a graphic object (e.g., point, line, circle) for a drawing program. Other possible instances of minimal units are textual messages, voice messages, screen snapshots, still images, video, and others.

CES, a distributed collaborative system [6] uses the notion of document section, where each section contains among other fields, the name of the user holding the lock. *GroupDraw* [5] establishes ownership for its objects by defining coupling levels: private, public and shareable.

A cooperative document is a set of minimal units, each one having its owner. In order to provide room for cooperation, members must have access rights to minimal units owned by other members. There must be a way to specify who will have the ability to change, read, propose suggestions or make comments on minimal units belonging to others.

2.2.1. Access Control Strategies

A crucial aspect of this document based approach for cooperation concerns the way in which team members and non-team members have access to documents and to parts of documents. In this approach all access details must be specified. We have listed three different control strategies namely: *privilege, direct and mixed access*.

Privilege Access

This access control strategy associates members to privilege levels. Level 1 is the top level, and members at this level are the most privileged ones. Members at level *n* have access to all minimal units owned by members at levels higher than *n*. Its main disadvantage is that, in cases like those suggested by the hierachical configuration presented in Figure 1, manager C for example could interfere with workers D and E that are not under C's command.

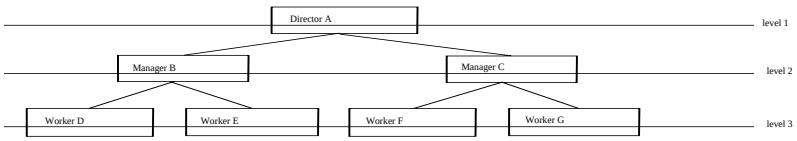

Figure 1. Problems with the Privilege Level Access Strategy

Direct Access

For this strategy every member that creates documents must state explicitly which other members have access to its minimal units. The main disadvantage is that it neither accepts the definition of an access control hierarchy nor accepts a pre-defined standard access control structure.

Mixed Access

This strategy satisfies both the definition of a hierarchy and accepts a pre-defined standard access control structure, but it is not limited by them. It is based on two access control modes: the global access rights, which are used to define an access control hierachy, and the local access rights, which allow members to individualize access to its minimal units, breaking the pre-defined standard access control structure.

The head of the team defines the access control hierarchy. Every access to documents created by the team is based on the hierarchy. This hierarchy is defined via global access rights using commands that state which minimal units of member X can be accessed by member Y. Thus, the access control hierarchy is a set of global access rights defined by the head of the team. In the access control hierarchy presented in Figure 2, everybody can see everyone else's minimal units, but may not change them.

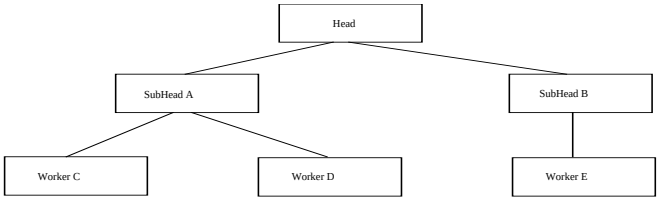

Figure 2. Access Control Hierarchy

Local access rights are defined in order to bypass the access control hierarchy defined via global access rights. Using local access rights members that create documents give other members individualized access to their minimal units. Local access rights are defined in a similar way to the global access rights, the difference being that here specific minimal units should be indicated—like selecting a specific paragraph with the mouse within a word processor application.

It is important to emphasize that commands which conflict with the access control hierarchy defined by the head of the team are prohibited. For example, looking at the figure above, it is not possible for worker C to forbid SubHead A to change his minimal units.

2.2.2. Undo

The cooperative system offers an undo mechanism which is more complex than the one available for non-cooperative applications. In order to provide this feature it is necessary to associate an action list for each member participating in the making of the document. The access control hierarchy and the local access rights should be taken into account, because actions taken over a minimal unit by its owner prevail upon actions taken by other members. CES [6] implements a different undo mechanism using a version stack, (for each document section), which contains old versions of the section.

Consider that Tom creates minimal unit x and then Martin changes minimal unit x. If either of them performs an undo at this point, minimal unit x will return to its former state. But, instead of the undo, imagine that Tom changes minimal unit x. Now, if Tom performs an undo at this point,

minimal unit x will return to exactly how it was when Tom left it—after the previous change. On the other hand, if Martin performs an undo, nothing will happen because Tom's actions prevail over his minimal units.

The following undo rules implement the undo mechanism. This first rule states that an action taken by the owner of a minimal unit, can only be cancelled by himself.

Given that member A takes a generic action over a minimal unit x of his own Then the action is inserted in member A's list and all actions taken over minimal unit x are removed from all other members' lists

The next allows the owner of a minimal unit to undo the actions taken by other members over his minimal unit. This rule also permits members, other than the member who owns the minimal unit, to undo actions taken over it, until a new action is taken by the owner.

Given that member A takes a generic action over a minimal unit of member B Then the action is inserted in both member A's and member B's lists

The following rule forbids more than one undoing of an action taken over a document, in order to mantain its consistency.

Given that member A performs an undo over a generic action (over a minimal unit of his own) Then the action is removed from all members' lists

This last rule forbids a member who has lost his access right over a specific minimal unit to undo actions taken over it.

Given that member A denies member B access to minimal unit x Then all actions taken over minimal unit x are removed from member B's list

2.2.3. Lock & Unlock

Lock and unlock are necessary when one team member is editing a part of a document and does not want to share it with another member. In such situations the part being edited should be locked in order to stop other team members from changing it. At the end of the editing process this part of

the document should be unlocked. Locks are imposed over minimal units, i.e. while editing a part of a document, the lock will be applied to the minimal units that comprise it. There are two different kinds of lock and three different kinds of unlock. All of them are determined and constrained by the cooperative application which has control over the user interface.

Implicit Lock	the lock automatically applies to a selected minimal unit
Implicit Unlock	the unlock takes place when the user de-selects a minimal unit
Explicit Lock	the lock is forced by the user. This is quite useful when the user wants to make a change to the document that involves more than one minimal unit
Explicit Unlock	the unlock is commanded by the user
Timeout Unlock	the unlock is applied to a set of minimal units, after some pre-defined user inactivity period

Our locking policy is going to be based on a strict two-phase locking protocol (as adopted by most commercial database systems). This policy has as main advantage avoiding cascading rollback [20]. If this protocol is enforced, then a member will only see a change performed by another member on the shared work surface if and only if this change successfully comes to its completion, i.e., no abort action is taken during the process.

Colab [18] is an experimental face-to-face meeting room set up at Xerox PARC that has a voice locking mechanism. Verbal cues are used by the participants to coordinate their actions, avoiding conflicts while using the shared work surface. There are other groupware applications like GROVE [2], which is a real-time group text editor with a no locking policy.

2.2.4. Version Control, Backup and History Lists

The cooperative system keeps many versions for each document. At any time, the member who created a document can return to a previous version. The system also provides automatic backups of the documents based on a pre-defined number of actions taken in the documents. The maximum number of backups per document is defined by its owner. An interesting implementation for this feature is given in [6] where a version stack is associated to each document.

The cooperative system supports two different kinds of history lists, namely local history and global history. The former keeps a document action list which shows how the document evolved. The following are examples of actions: version handling and minimal unit creation. Global history refers to a project, keeping a project action list where actions are for example, document creation

and deletion. This feature enables the reconstruction of a complete project and its documents. These lists will be extended to accommodate relevant information for providing decision rationales [4].

2.3. Cooperation Policies

For members who do not have the right to alter documents the system offers the possibility of making suggestions. In order to make a suggestion, a member selects and changes the part of the document according to his wish, together with a brief text justifying the change. All the owners of the minimal units involved in the change will receive the justification and approve it or not.

Comments, like suggestions, exist to let members without access to specific minimal units propose changes to them. They differ from suggestions for lacking the mechanism to actually make the changes, only the justification text is broadcasted to the respective minimal unit owners. Comments are also sent to commentators and readers if the document has the *public reading* attribute.

The member who created a document may define it as one of *public reading*, letting non-team members read the document (readers). The member who created the document may also define it as one of *public comments*, letting non-team members make comments on the document .

During document editing time it is important to provide a synchronous text based communication channel for team members. There they can discuss the work, clarify positions, pose questions, give answers, and provide rationales, etc. This could be implemented via an extra window in each user's screen. In order to follow the conversation precisely, labelled multiple cursors [5] are offered: one can see the other's cursor showing the relevant part of the document.

Privacy should always be a very important prerogative at work, whether one is working alone or in a group. A team member may wish to sketch some graphics or write some text without it being seen by the others. If it comes out the way he wishes , then the draft can be shown to the group. For this purpose each member has a private area associated to each document that he is working in. Through the copy and paste mechanism, drafts are moved from the private area to the document and vice versa. Finally, a team member could also bring a version of the document to his private area and make a private version of it.

3. The Object Oriented Framework

This section introduces an object oriented framework for the document based approach for cooperation. The choice for using an object oriented approach for design, and more specifically to employing ADV's [1] (Abstract Data Views), lies in their suitability for separately developing the user interface and the core of the application's specification through prototyping.

The proposed framework follows Rumbaugh's [17] understanding of a framework as an object oriented description of system components and how they are connected to each other. The framework comprises a Data Dictionary which is presented in the appendix and Object Diagrams that are shown below.

3.1. Object Diagrams

The framework's static structure is presented using Object Diagrams. These diagrams illustrate the relationship between entities as described in the previous section.

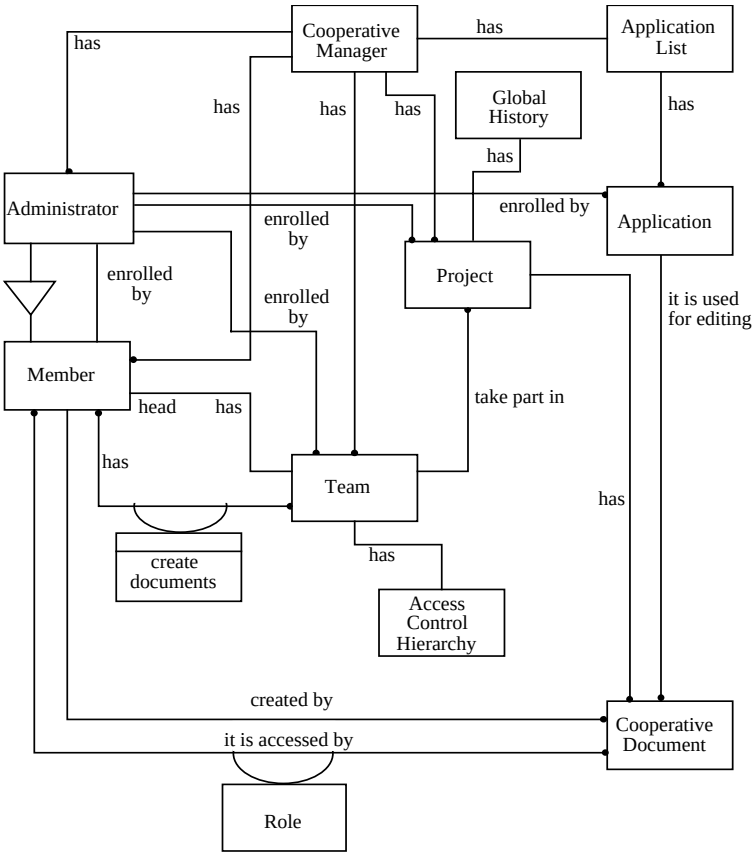

Figure 3. Cooperative System

The diagram in figure 3 shows relationships between the classes which comprise the cooperative system. The Cooperative Manager keeps lists of members, teams, projects, administrators and cooperative applications existing in the system. A member may participate in different teams. In some of them he may create documents, in others he may not. There is a set of documents

associated to each member which is created by him, and a set of documents accessible to the member; a member may have a different role for each document. The administrator is an especial member who may create teams and projects and enroll new members, etc.

A team consists of many members, one of them having overall responsibility for the team—the head of the team. There is an access control hierarchy associated to each team. A team may participate in various projects.

Each project is run by one team. For each project there is an associated set of documents. A project has an access history list of documents (creation, editing and deletion), that indicates the date and the time the members accessed them. A cooperative document has an associated set of members that may edit it.

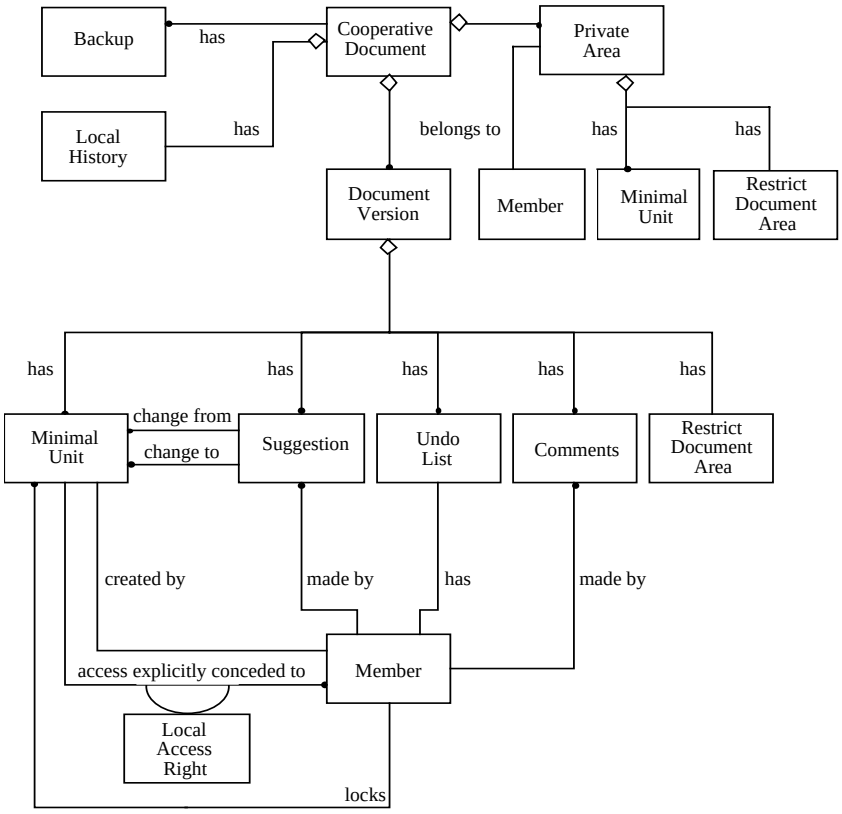

Figure 4. Cooperative Document

The diagram in figure 4 presents the cooperative document structure, showing the components which comprise it. A cooperative document consists of a set of versions of itself, a set of backup copies, a local history list and private areas. The local history list keeps member access information on minimal units (creation, change and deletion), that differs from the global history list, which has access information on documents, not minimal units.

A document version comprises: a set of minimal units, a set of suggestions, a set of comments, a restrict document area and an undo list for each member who has access to the document.

A minimal unit is created by one member, who may associate access rights to it. Imagine that Martin does not have access to Tom's minimal units, but Tom wants Martin to change one of his minimal units. In order to do that, Tom has to give Martin a local access right for that specific minimal unit, breaking the access control hierarchy. It is important to realize that Tom is allowed to give access to Martin, who is not enabled to access because of the access control hierarchy. On the other hand, Tom cannot prevent Dov, who is hierarchically above him, from accessing his minimal units.

A restrict document area is used by the cooperative application to store data which is not associated to any member.

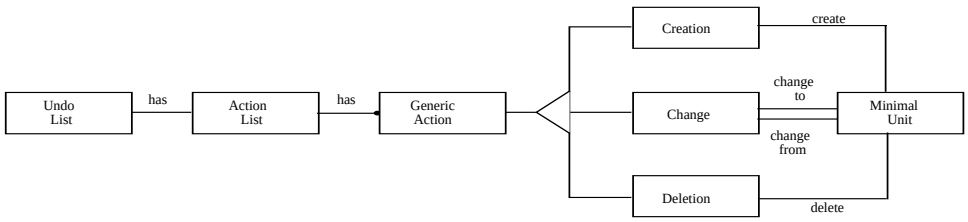

Figure 5. Undo List

The diagram above shows the undo list structure, which has three possible kinds of actions, namely: creation, change and deletion. It is important to realize the following difference between the undo list and the local history list. When a member undoes an action, the cooperative manager removes the action from the undo list, but leaves the action recorded in the local history list, together with the requested undo.

4. Architecture and Implementation Issues

A prototype of the cooperative system is being built using Tcl-DP, which is an extension of Tcl [14]. It supports distributed processing via RPC [13]. Tcl-DP was chosen based on the following:

- *GroupKit* [15, 16], a toolkit for building real-time conferencing systems is written in Tcl/Tk.
- Tk is an Tcl extension which is appropriate for user interface development. Tcl/Tk handles user interface and application development separately [1]. Tk is faster and easier to understand than Motif and Xt.
- Tcl is a cross-platform development language. It runs in Unix, Windows and MacOs.

- Tcl commands handle strings in a way that simplifies the implementation of communication protocols.

- Possible bottlenecks in Tcl may be avoided using C code instead.

Unlike *GroupSketch* and *GroupDraw* [5] which use fully replicated architectures less vulnerable to network and machine failure, our cooperative system prototype architecture is a centralized one—easier to keep the work surfaces and the user requests synchronized.

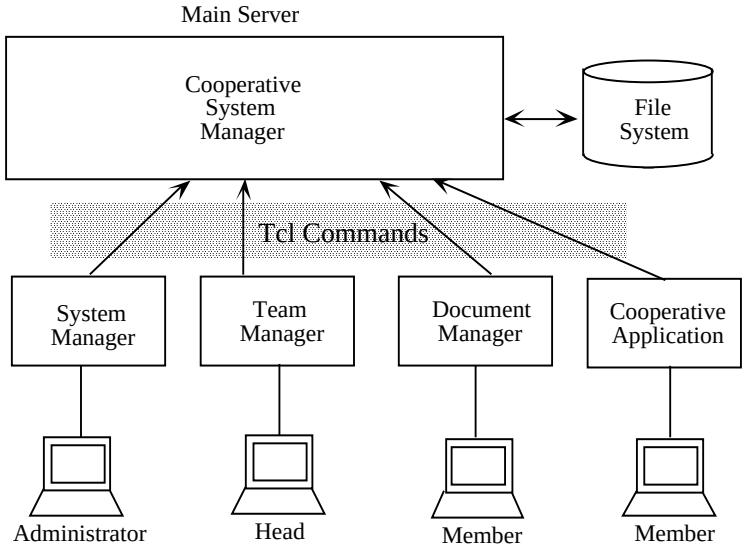

Figure 6. Cooperative System Prototype Architecture

In order to provide a user friendly interface, the cooperative system has three different managers, each of them performing a specific role. The first application is the System Manager, which is used by the administrator to enroll members and for team formation. The second application is the Team Manager, which is used by the head of the team to enroll team members, to define the access control structure and to create projects. The third application is the Document Manager, which is used by team members to access, create and delete project documents. Finally there are the Cooperative Applications like multi-user editors.

Tcl commands implement the asynchronous communication protocol between clients—managers and cooperative applications—and server. The table that follows shows one (there are many) command names and its description for each client:

CreateProject	Administrator creates a project for a team	System Manager
EnrollMember	Head of the team enrolls a member to a team	Team Manager

OpenDocument	Member opens a specific version of a document of a project	Document Manager
CreateMinimalUnit	Member creates minimal unit, returning its Id	Cooperative Application

5. Conclusion & Future Work

This paper focuses on system level support for real-time desktop conferencing and whiteboard meeting tools. We propose a document based approach for cooperation, and use it for modelling a preliminary cooperative system. We outline an object oriented framework for the cooperative system implementation. Based on this framework, our next step is to develop a cooperative text processor and a cooperative graphic editor for the electronic meeting room. The user interface design, or better saying, the human human interface design for the cooperative system and its applications will meet Tang's [19] design criteria for common work surface tools. Regarding the cooperative text processor and the graphic editor interface, we will pay especial attention to the insights and advice coming from [2, 5].

Our main objective is to set up a cooperative environment where software experiments like, cooperative design [4], cooperative requirements capture [12], design recovery sessions [10] and program derivation sessions [7] may take place.

Acknowledgments

The authors would like to thank Professor Carlos José Pereira de Lucena for his comments. We also wish to thank Marcelo de Carvalho Moreira, Marcelo Schmidt Moreira, Andrés Ricardo Pardo Irazabal and Christiano de Oliveira Braga who cooperated in the making of this paper. A special thank-you goes to the anonymous referees who carefully revised this paper.

This effort is part of Project DRESSER *Design and Re-design of Software/Eletronic Meeting Room* (CNPq - Brazilian National Council for research and Development, Grant nº 521698/93-6), which is a follow-up to Project RASD *Research Agenda for Software Design* [11] (CNPq Grant nº 500277/91-3).

References

[1] Cowan, D. D., Ierusalimschy, R., Lucena, C. J. P., Stepien, T. M. Abstract Data Views, Structured Programming, 235-249, 1993

[2] Ellis, C. A., Gibbs, S. J., Rein, G. L. Groupware: Some Issues and Experiences, Comm. ACM, 34(1), 38-58, 1991

[3] Elrod S., Bruce, R., Gold, R., Goldberg, D., Halasz, F., Janssen, W., Lee, D., McCall, K., Pederson, E., Pier, K., Tang, J. LIVEBOARD: A Large Interactive Display Supporting Group Meetings, Presentations and Remote Collaboration, In Readings in Groupware and Computer-Supported Cooperative Work, ed. R. M. Baecker, Morgan Kaufmann Pubs, 709-717, 1993

[4] Fuks, H., Lucena, C. J. P., Duarte R. C. Towards the Requirements for Computer Supported Cooperative Design, Design Methods: Theories, Research and Practice; 27(4), 1865-1878, 1993

[5] Greenberg, S., Roseman, M., Webster, D. Issues and Experiences Designing and Implementing Two Group Drawing Tools, IEEE Proc. 25th Anual Hawaii Intl. Conf. System Sciences, vol 4, 139-150, 1992

[6] Greif, I., Seliger, R., Weihl, W. A Case Study of CES: A Distributed Collaborative Editing System Implemented in Argus, IEEE Trans. Software Engineering, 18(9), 827-839, 1992

[7] Haeberer, A., Veloso, P. Partial Relations for Program Derivation: Adequacy, Inevitability and Expressiveness, In Constructing Programs from Specifications, ed. B. Moeller, Elsevier Science Publishers, 319-371, 1991

[8] Hiltz, S. R., Turoff, M. Structuring Computer-Mediated Communication Systems to Avoid Information Overload, Comm. ACM, 28(7), 680-689, 1985

[9] Lano K. Developing Formal Semantics for Object-Oriented Specification Languages, Technical report, Lloyd's Register, Croydon, UK, 1993

[10] Leite, J. C. S. P., Prado, A. F. Design Recovery - A Multi-Paradigm Approach, Proc. First International Workshop on Software Reusability, Germany, 161-169, 1991

[11] Lucena, C. J. P., Leite, J. C. S. P., Shwabe, D., Fuks, H. A Research Agenda for Software Design, MCC 29/91, Dept. Informática, PUC-Rio, 1991

[12] Macaulay, L. Requirements Capture as a Cooperative Activity, IEEE Int. Symp. Requirements Engineering, IEEE CS Press, CA, 174-181, 1992

[13] Manson, C., Thurber, K. Remote Control, Byte, 235-249, July 1989.

[14] Ousterhout, J. An Introduction to Tcl and Tk, Addison-Wesley Publishing, 1994

[15] Roseman, M. Tcl/Tk as a Basis for Groupware, Proc. Tcl/Tk Workshop at Berkeley, 1993

[16] Roseman, M., Greenberg, S. GroupKit A Groupware Toolkit for Building Real-Time Conferencing Applications, Proc. Conf. Computer-Supported Cooperative Work 1992

[17] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Larensen, W. Object-Oriented Modeling and Design, Englewood-Cliffs, Prentice-Hall, 1991

[18] Stefik, M., Foster, G., Bobrow, D. G., Kahn, K., Lanning, S., Suchman, L. Beyond the Chalkboard: Computer Support for Collaboration and Problem Solving, In Computer-Supported Cooperative Work: A Book of Readings; ed I. Greif, Morgan Kaufmann Pub., 335-366, 1988

[19] Tang, J. C. Listing, Drawing and Gesturing in Design: a Study of the Use of Shared Workplaces by Design Teams, Xerox Parc, SSS-98-3, April 1989

[20] Ullman, J. D. Principles of Database and Knowledge-Base Systems, Computer Science Press, 1988

Appendix: Data Dictionary

This appendix introduces a data dictionary for all existing classes appearing in the diagrams of section three. The approach proposed by [9] is used to describe the attributes of a class.

Class: Action List

Description: Models an action list which is used for the Local History, Global History and Undo List.

Attributes:	actions	set_of Generic Action

Class: Administrator

Superclass: Member

Description: Models a user who takes the role of System Administrator. The administrator is in charge of creating projects, enrolling new teams, team members and applications.

Attributes:	principal	boolean

Class: Application

Description:	Models a cooperative application.	
Attributes:	name	string
	enrolled by	Administrator

Class: Application List

Description:	Models a list which contains all the cooperative applications that use the Cooperative Manager to access documents.	
Attributes:	applications	set_of Application

Class: Backup

Description:	Models a backup copy of the document.	
Attributes:	version	Document Version

Class: Change

Superclass:	Generic Action	
Description:	Models a change done over a minimal unit.	
Atributes:	minimal unit before	Minimal Unit
	minimal unit after	Minimal Unit

Class: Comments

Description:	Models a comment made about the whole document.	
Attributes:	text	string
	owner	Member

Class: Access Control Hierarchy

Description:	Models the team's access control hierarchy, and defines for each user's minimal unit, which users have access to it.	
Attributes:	global access rights	set_of Global Access Right

Class: Cooperative Document

Description:	Models a cooperative document, i.e. a document that could be edited by many people at the same time.	
Attributes:	application	Application
	public reading	boolean

	roles that access documents	set_of Role
	versions	set_of Document Version
	private areas	set_of Private Area
	public comments	boolean
	creator	Member
	backups	set_of Backup
	local history	Local History

Class: Cooperative Manager

Description:	Models the cooperative manager whose function is to manage all system activities like team formation, members enrollment, document access, etc.	
Attributes:	members	set_of Member
	teams	set_of Team
	administrators	set_of Administrator
	application list	Application List

Class: Creation

Superclass:	Generic Action	
Description:	Models the creation of a minimal unit.	
Attributes:	minimal unit	Minimal Unit

Class: Deletion

Superclass:	Generic Action	
Description:	Models the deletion of a minimal unit.	
Attributes:	minimal unit	Minimal Unit

Class: Document Restrict Area

Description:	Models a reserved area in the document just for application specific needs, for example minimal unit formats.	
Attributes:	data	seq_of byte

Class: Document Version

Description:	Models a specific version of the cooperative document.

Attributes:	version number	string
	minimal units	set_of Minimal Unit
	suggestions	set_of Suggestion
	undo lists	set_of Undo List
	comments	set_of Comment
	document restrict area	Document Restrict Area

Class: Generic Action

Description: Models a generic action taken over a minimal unit, which could be creation, change, deletion, etc.

Attributes:	date	date
	hour	hour
	member	Member

Class: Global Access Rights

Description: A set of Global Access Rights is the way to define the access control hierarchy of a team. Basically this Class Models sentences of the following kind:

```
member B MAY CHANGE units of member A
member C MAY SEE units of member D
```

Attributes:	member1	Member
	member2	Member
	type	Access_Type

Class: Global History

Description: Models the project's history, and contains all the actions related to the project like document creation, deletion, etc.

Attributes:	action list	set_of Generic Actions

Class: Local Access Rights

Description: Defines which users, apart from those enabled by the access structure, may access a specific minimal unit. Basically this class models sentences of the following kind:

```
member B MAY CHANGE minimal unit X of member A
member C MAY SEE minimal unit X of member D
```

Attributes:	member	Member
	type	Access_Type

Class: Local History

Description:	Models the document's history, and contains all actions related to the document like minimal unit creation, deletion, etc.	
Attributes:	action list	set_of Generic Actions

Class: Member

Description:	Models a Cooperative System.	
Attributes:	name	string
	password	string
	enrolled by	Administrator

Class: Minimal Unit

Description:	Models a Minimal Unit which is the minimum part of the document to which ownership can be assigned—it varies depending on the kind of document. The semantics of a minimal unit is only known by the application—it is unknown by the Cooperative Manager.	
Attributes:	id	ID
	data	seq_of byte
	owner	Member
	local access rights	set_of Local Access Right
	locked by	Member

Class: Private Area

Description:	Models a private user area proper for sketching.	
Attributes:	owner	Member
	minimal units	set_of Minimal Unit

Class: Project

Description:	Models a project comprising cooperative documents.	
Attributes:	name	string
	global history	Global History
	documents	set_of Cooperative Document
	created by	Administrator

Class: Role

Description:	Models the members role for document editing. The possible roles are:	
	Author:	may create Minimal Units;
	Commentator:	may make Suggestions and Comments;
	Reader:	may read and comment a document.
Attributes:	member	Member
	type	Role_Type

Class: Suggestion

Description:	Models a suggestion, which is the proper way for a user who is not enabled by the access structure to change the document, at least to propose changes in the document. The change proposed by the suggestion will take effect if and only if all the users who are owners of the minimal units affected accept, otherwise the suggestion is discarded. The "text" attribute shows the motivation for the change.	
Attributes:	units to be altered	set_of Minimal Unit
	proposed units	set_of Minimal Unit
	owner	Member
	text	*string*

Class: Team

Description:	Models a team consisting of many members involved in various projects	
Attributes:	name	string
	head	Member
	members	set_of Member

	access control hierarchy	Access Control Hierarchy
	members who can creates documents	set_of Member
	projects	set_of Project
	created by	Administrator

Class: Undo List

Description:	The undo list contains all the actions performed by the team members on a cooperative document.	
Attributes:	owner	Member
	action list	Action List