THE DESIGN OF AN OBJECT-ORIENTED COLLABORATIVE SPREADSHEET WITH VERSION CONTROL AND HISTORY MANAGEMENT*

David A. Fuller†, Sergio T. Mujica‡, José A. Pino§

† Departamento de Ciencia de la Computación, Pontificia Universidad Católica de Chile
‡ Departamento de Ingeniería Informática, Universidad de Santiago de Chile
§ Departamento de Ciencias de la Computación, Universidad de Chile

Keywords: Spreadsheet, object-oriented, interfaces, history management, version control.

Abstract

A novel spreadsheet to support collaborative work is introduced. Its features include an object-oriented group interface, shared-objects support, object version control and group process history management.

Introduction

Spreadsheet systems are one of the most successful software products for microcomputer platforms, and recently have become popular in other environments. The traditional spreadsheet has a good human-computer interface, useful in typical offices.

Panko [11] has reported that office workers spend as much as 30 to 70% of their time in meetings. However, there are not many office tools to support these activities, except for individual work in those meetings. In particular, spreadsheets lack collaborative features in general.

There are however, spreadsheets that have limited collaborative features. Excel [9] is a spreadsheet with the ability to share documents using the publish/subscribe facilities provided by the Apple Macintosh System 7 operating system. Also, Patel and Kalter [12] describe a spreadsheet for cooperative work in a multi-user platform. However, both provide facilities for working on the same document, but lack the tools for assisting real "face-to-face" or "remote" collaboration.

We are seeking features that provide real collaboration among users and are not restricted to tools to view or work on the same document. These features should allow a group of people to actively work on cases such as the following ones:

1. Engineers, business analysts and executives preparing a contract proposal.
2. A software engineer, a software user, and a financier negotiating the specifications of software.
3. The management of a company working on the budget for the next year.

All these people need to collaborate to produce final spreadsheets. They mutually interact using collaboration protocols which are determined by their position in the company, expertise, role played in the problem, standard work procedures, etc. For this purpose, the software tool should provide facilities to create private and shared documents, provide a common view of data to all users, real-time notification of modifications to data, access to parts of a spreadsheet, modification of data without destroying other data previously contributed by other users, storage of different versions, the ability to review past decisions, etc. These features should be non-intrusive for the users.

The idea of the intuitive interface of existing spreadsheets should be kept in a collaborative spreadsheet system. Furthermore, the convenience of using object-oriented interfaces for solution design is well-known; its advantages include reusability and incremental developments [4]. Thus, it would be desirable to have object-oriented facilities in a collaborative spreadsheet system. Such facilities should include functionality to share objects according to different group work strategies. An object-oriented approach for operation of the spreadsheets also seems convenient because managing only spreadsheet cells and formulas requires thinking at low levels of abstraction whereas an object-oriented approach provides abstraction capabilities enabling users to structure their work in more powerful ways.

The process of collaboration is dynamic, i.e. it requires management of data that is constantly under modification by the group members. It is a common practice in software engineering to use computer-based tools to manage source code versions, such as RCS [16] or SCCS [15]. Such feature could be applied to support group collaboration with spreadsheets. It is desirable to have means to temporarily review and modify the values of some objects. There is also a need to compose sets of versions which are connected according to user criterion. This connection is the basis on which to build a release of versions of objects.

Access to past decisions is important in single user systems to review them. Even more important is this feature in collaborative systems, where the interaction of people make decisions harder to take.

* This work was partially funded by the Chilean National Science and Technology Fund (FONDECYT), projects 0812-92, 0437-92 and 1162-92, and grants from Pontificia Universidad Católica de Chile DIUC 35PFI-91 and Synapsis S.A.

ACM-SAC '93/2/93/IN, USA
© 1993 ACM 0-89791-568-2/93/0002/0416...$1.50

This paper describes the design of S^3 (Super SpreadSheet), a collaborative spreadsheet system under development for the UNIX/Motif platform. It has a novel object-oriented group interface, it provides object version control and group process history management. Up to seven S^3 users can have their own graphic workstations, working in the same room, interacting verbally among themselves. The development of S^3 is an applied research project undertaken by groups from three universities, and it is not (yet) intended to become a commercial venture. The main objectives of the three groups is to study history management, tightly-shared objects, and user interfaces in collaborative systems.

Overview of the user-interface

Although S^3 is a tool which can be useful to sophisticated users, it can be used by novices as well. The basic procedure to ask for a service is to point to whatever is to be operated upon and then to the operator, which can be a button, an arrow or a menu item.

Upon entrance to S^3, a menu will appear at the top of the screen and a "control window" will also be shown. The menu contains the following entries: database, spreadsheet, edit, objects, history, format, options and communication (Figure 1).

The "database" entry is a menu for services concerning the whole database of spreadsheets that can be handled with S^3; it also has a "quit" option to exit the system. The "spreadsheet" menu applies to the spreadsheet the user is working on; it contains the services related to the complete spreadsheet, such as providing the way to begin work with a new spreadsheet. The "edit" menu includes options to insert or delete rows or columns and copy and paste. The "objects" menu offers the services to create objects, define protocols, etc. (discussed in section "The object system" below). The "history" menu has entries to define transactions and manage versions, releases and relations (see section "History management"). The "format" menu has the following options to handle data: number, alignment, font, border, style, row height, column width, justify. The "options" menu has choices to customize the user interface: color palette, preferences, calculate now. Finally, the "communication" menu lists services to help work with the other people and is discussed below and in its respective section.

The "spreadsheet" menu has the following options: new, open, close, make release, photocopy. Pointing to "open" will create a window showing a portion of an existing spreadsheet. The desired spreadsheet is chosen from a dialog box. Pointing to "new" creates a window showing a new (empty) spreadsheet. Figure 1 depicts a sample screen of a S^3 user; the "Purchases" spreadsheet is being worked on whereas "Sales" has just been opened. Notice in the "control window" that "Sales" is the current spreadsheet.

A beginner can immediately start writing on the cells of the current spreadsheet. The system then automatically sets default values to the corresponding object attributes (further discussion in section "The object system").

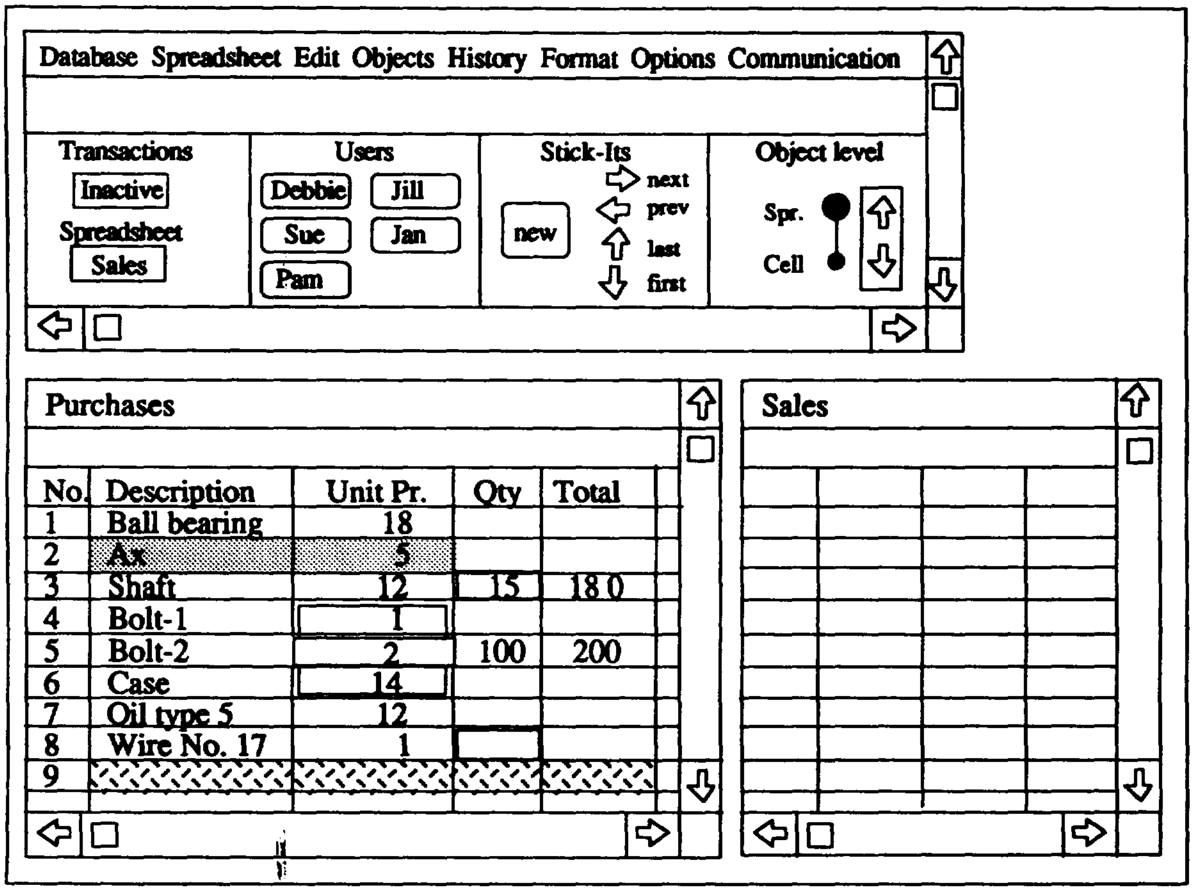
Other users running S^3 are depicted in the "control window" as buttons, each showing the user name and her unique assigned color. Selecting a user button activates a telepointer [14], which will be visible if that user is working over the spreadsheet portions shown on the open windows. If the mentioned user is working on another part of the same spreadsheet or on another spreadsheet, a new window is shown on the screen illustrating the activity of the chosen user; no automatic deletion is provided for such window, i.e., if the telepointing feature is deactivated, the windows opened beforehand remain in the screen unless the user manually closes them. The "telepointer" option of the "Communication" menu is an alternative way of activating/deactivating a telepointer.

The rightmost subwindow of the "control window" is intended to give a hint on the object hierarchy (discussed below) of the current spreadsheet. There is a vertical path, the bottom node representing a simple cell of the spreadsheet, the node above it representing an object which is an aggregation of simple cells, etc. The top node represents the whole spreadsheet. The default object level is the cell. The vertical arrows that appear in this subwindow are used to climb or descend on this hierarchy. A mouse click on a cell selects the respective simple object; this action is shown on the "Object level" subwindow by highlighting the cell and enlarging the bottom node on the path. Climbing the hierarchy is, of course, reflected by highlighting the corresponding object and enlarging its level node.

One of the distinctive features of the S^3 user interface is the "Stick-It" paradigm. This feature is contributed as an intuitive device to simplify history management, since it provides the capability to overwrite data without losing it, with unlimited levels. A "Stick-It" is the image of a piece of paper depicted in the control window as a metaphor of a Post-It[1] note. When the user makes a mouse click over an object and then clicks over the "new" icon of the "Stick-Its" subwindow, whatever data is contained in the object is made invisible, covered by the virtual piece of paper. Now the user can write over the cells covered by the "Stick-It". Of course, new "Stick-It"s can be placed over the previous ones. These "Stick-It"s may be of different sizes and shapes, depending on the sizes and shapes of the objects covered by them. "Stick-It"s can be examined on a top-down fashion using the arrows provided in the "control window". Thus, the user can uncover the current "Stick-It", place again the previously uncovered one, see the first (original) data and restore the last "Stick-It" over a chosen object (which can be a simple cell, see section "The object system"). The arrows provided in the "Stick-Its" subwindow are the tools to handle these tasks.

A "Stick-It" placed over an object of the spreadsheet can be recognized because it draws a second line around the object. A dotted region indicates that one or more "Stick-It"s previously covering it have been removed for examination using the controls on the "Stick-Its" subwindow. Figure 1 shows a few "Stick-It"s in the "Purchases" spreadsheet. There is one covering the "Qty" column for items No. 3 and 8, the first being written over. Another "Stick-It" covers the "Unit Pr." column for three values, but it is itself partially covered by one having the value 2 for item No. 5. Finally, there are one or more "Stick-It"s covering the second and third values of item No. 2, but now have been temporarily removed.

[1] Post-It is a trade mark of 3M Corporation.



No.	Description	Unit Pr.	Qty	Total
1	Ball bearing	18		
2	Ax	5		
3	Shaft	12	15	18 0
4	Bolt-1	1		
5	Bolt-2	2	100	200
6	Case	14		
7	Oil type 5	12		
8	Wire No. 17	1		
9				

Figure 1: S^3 user interface.

The "Stick-It" traversal is done on an object basis. Thus, the user must first choose an object (which may be a cell or an aggregation of these) and then use the arrows in the "Stick-Its" subwindow to examine them. Notice that if the object is a cell, all "Stick-It"s covering it are accessible, but if the object is an aggregation, only the "Stick-It"s covering the complete object are accessible through the arrows.

After working with a spreadsheet for a while, the users may be satisfied with its currently visible values and do not wish to keep the "Stick-It"s. They can do this by choosing the "photocopy" option from the "spreadsheet" menu.

Another menu is called "database". This menu offers options to handle the whole database of spreadsheets which are being worked upon. Its options are: make release, commit, rollback, checkpoint, restart, quit.

The "make release" option of the "database" menu has the effect of constructing a release of all spreadsheets. The concept of release is based upon the concept of version that we describe in full in section "History management". Data stored in the database can have multiple versions at the same time, one of which is said to be the current version, while any other can be retrieved on demand. A release is a collection of versions stored in the database. "Make release" saves a collection of all current versions of data, together with pertaining documentation. The release does not store the "Stick-It"s; it stores only visible information and thus its effect is that of a copy machine over the spreadsheet as well as providing a milestone to document it for later reference. The "checkpoint", on the other hand, is simply a copy of the state of the whole system at the time it is issued. The "restart" option resets the state of the system to the time of a selected checkpoint instant.

The leftmost subwindow of the "control window" has an indicator of transaction status (see section "Concurrency control" for a detailed explanation of transactions). This indicator is "active" whenever a transaction is open and is "inactive" after a "commit" operation is performed. Notice that the indicator is set to "active" either by an explicit "open transaction" action or implicitly by starting to write over a new spreadsheet.

The "edit" entry of the main menu has the following options: delete, copy, cut and paste. The delete option erases the data contained in the previously selected object. Object selection is done by clicking over cells and going up in the object hierarchy using the arrows tool in the "object level" subwindow of the control window, if necessary. Also, clicking over a column or row heading selects the complete column or row. Notice that the deletion of a spreadsheet may lead to method inconsistencies, which must be resolved by the user; this is similar to the formula modifications that are needed after deleting rows or columns in a traditional spreadsheet.

The "copy" option of the "edit" menu is equivalent to a "clone" operation and initializing the object to the value of the copied object. The "cut" option completely removes an object; it is replaced by empty cells.

The object system

The root of all object-oriented systems is found in Simula [5]. There are several object models which are currently accepted, such as the SmallTalk model [6]. This model is based on classes, meta-classes, class inheritance and polymorphic operations. We might call it a *classical* model, which is essentially static, i.e. the program text defines the structures of objects and inheritance rules. Thus, classes inherit properties when they are defined rather than at run-time. Other models such as actors [1] leave inheritance resolution to run-time procedures through mechanisms such as delegation [8]. Another model is the one used by ThingLab [3] in which there are no classes, but a user can create prototype objects and then new objects are created based on such prototype objects.

A spreadsheet application needs to be as simple as possible to casual users. We have defined two basic objects, spreadsheets and cells. We use cells to compose other, more complex objects. Each object is a prototype from which we can create new objects that have the same structure and behavior as the prototype. The only way of creating new prototype objects (this is dynamic creation of classes) is by aggregation [13] of other, previously existing objects.

Inheritance of behavior is accomplished at run-time by delegation of messages. Inheritance of object attributes is also done at run-time by copying a prototype object. Aggregation implies inheritance of collaboration protocols as discussed below.

One essential ingredient of collaborative systems is providing the ability to *tightly share data*[2] among a group of people collaborating in the pursue of common goals. Some of the basic tight sharing capabilities available in S^3 are:

1. The existence of all objects is knowable by all users, regardless of whether objects are private or shared. The identity of who is either reading or writing each object should also be knowable by all users.
2. Two or more users can access the same object simultaneously. It is possible, however, to place some restrictions on such simultaneous access to objects by several users. For example, two users write while some of the others can only read.
3. Changes made to an object that is shared by two or more users are visible to those users who share the object in real-time (i.e. as they are made).
4. A user who is modifying an object can choose to restrict the visibility of changes made to the object until such changes are complete, provided that the user has exclusive write access to the object.
5. There are capabilities to define shared and private objects as well. A shared object can be simultaneously used by several users. A private object can be used only by its creator.
6. It is possible to create and trace versions of individual objects and to create sets of versions of objects which are consistent with each other according to users' criteria. We call such sets *releases*.

Capabilities 1-4 are also available in systems for different purposes such as coSARA [10]. The following subsections deal with techniques to realize these capabilities in the collaborative spreadsheet.

Objects can be aggregations of other objects, which in turn can be cells or other aggregations [13]. Aggregations can be represented by a tree structure in which each node represents an object, an aggregation is the parent of its components and leaves of an aggregation tree correspond to cells. Thus each object belongs to only one aggregation.

Initially, each open spreadsheet is an independent tree whose leaves are the spreadsheet cells. When a set of cells is aggregated to make a new object, it becomes a child of the spreadsheet and the aggregated cells become children of the new object. For example, we could aggregate cells to make an object "matrix" and an object "vector". We could then make an aggregation of "matrix" and "vector" to form a new object "system of linear equations" (Figure 2 (a)). In the same vein it is possible to disaggregate objects, by removing the root of some subtree. Figure 2 (b) shows the situation after we have disaggregated the object "system of linear equations" leaving as a result objects matrix and vector.

Objects respond to the reception of messages by executing a sequence of actions that has been previously defined as a method corresponding to the message name. Our approach uses prototypes as a way to achieve ease of use. This model gives the user the feature of not having to define object structures before using the spreadsheet.

We define rules of inheritance for behavior, attributes and collaboration protocols. Inheritance of behavior occurs dynamically at run-time by delegation of messages to other objects. Inheritance of attributes occurs when a clone of an object is made. Furthermore, aggregation realizes inheritance of collaboration protocols.

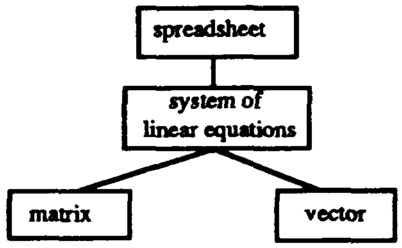


Figure 2 (a): Disaggregation of objects.

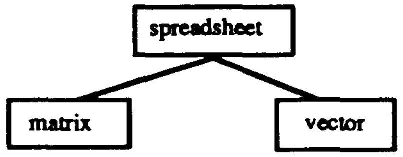


Figure 2 (b): Disaggregation of objects.

[2] Also known as collaborative sharing of data and strong sharing of data.

Concurrency control based on collaboration protocols

A collaboration protocol is defined as a set of rules stating when and how two or more users can access the same object simultaneously. In this work we associate a protocol to each object. Objects are given a default protocol at creation time to reduce overhead to the user. The default protocol is that of an open floor, in which any user can read and write the object simultaneously.

We define a protocol as a list of the form (protocol-type list-of-users) where protocol-type is either ALL or ONE-OF, and list-of users is a list of user names. (ALL list-of-users) means that all users whose name appears in list-of-users can read and write the object simultaneously. (ONE-OF list-of-users) means that only one of the users whose names appear in list-of-users can write the object while the others can only read it, unless the writer has chosen to restrict visibility of changes until they are complete. In both cases when list-of-users is nil, it is assumed that all possible users are in the list.

This set of protocols allows to make a private object for a user named owner assigning it either (ALL owner) or (ONE-OF owner). It also allows to easily express the default open-floor protocol as (ALL). Furthermore, components of an aggregation inherit the aggregate's protocol.

The users are informed about their writing rights with a color cue in the interface. A yellow colored object shown on the screen means this user is not currently permitted to modify this object. Figure 1 shows a yellow object on the "Purchases" spreadsheet in row No.9 (represented here as a marked area).

Within the scope of this work we have chosen to define a simple set of collaboration protocols. This set can be extended to cover more complex cases of collaboration. We have not attempted to model either collaborative tasks or complex group interactions. Other systems include other classes of protocols. The Coordinator has an embedded model of group interaction called *conversation for action* [17]; coSARA [10] includes a method to model the behavior of a group of users who share a set of objects which are displayed on the screen of their workstations using a computer-based tool.

Collaborative work is organized in transactions, using non-strict 2-phase locking [2]. The spreadsheet will always run one shared transaction, that runs distributed in the machines of all collaborating group members. A transaction is started when the system starts. Then a transaction commit operation marks the beginning of a new transaction (see Figure 1).

Locks are requested implicitly when a user attempts to modify an object for the first time since the beginning of the current transaction. This feature allows the user interface to be more alike ordinary spreadsheets since locks do not need to be requested explicitly in order to modify any object. Locking an aggregate implies locks on all its tree of components.

While operating within a transaction, a user acquires locks on objects at any time and in any arbitrary sequence. All locks are released atomically when the transaction is committed to prevent cascading roll-backs. The non-strict 2-phase locking scheme, may lead to deadlocks. We rely on the fact these are long interactive transactions, controlled by human users, and permitting resolution of deadlocks by social negotiation among users. The object management system provides all necessary information to carry on such negotiations. Deadlock detection algorithms are also used to warn users that deadlocks exist and that they should be resolved by negotiation.

For example, consider the following scenario[3]: user 1 needs to have exclusive access to objects A and B to perform some task. User 1 successfully obtains a lock on object A, but locking object B fails and user 1 is informed by the object management system that user 2 already has a lock on object B. User 1 then asks user 2 for how long she plans to use object B. User 2 answers that she will use object B only for a few minutes as soon as she gets a lock on object A. At this time user 1 realizes that a deadlock exists and informs user 2 that she already has a lock on A. User 1 then agrees to release her lock on A and wait until user 2 is done.

The interaction illustrated by this example is possible because the object management system provides information such that user 1 could identify user 2 as the current owner of a lock on object B. The interaction could be carried out easily because user 1 and user 2 were working face-to-face and therefore it was easy to talk.

Transaction commitment is distributed, since several users may be writing the same set of objects. Therefore it is necessary that they reach agreement to commit the current transaction. This process is done using a 2 phase commit, *at user level*, which is run with computer assistance.

Version and release management

We have developed an object management system with version support, using "Stick-It"s (previously discussed): intermediate values of an object created during a transaction are recorded temporarily in a Stick-It. In this metaphor we can place one "Stick-It" over another many times. We can thereafter examine and remove "Stick-It"s according to time of creation and by physical visibility.

When the transaction is committed, new versions of the objects that have been modified within the transaction are created. Each user that owns a write lock on an object is offered a chance to document the new version. When several users own a lock, the system forces them to choose one person to do the job by means of a simple user-interface device. A list of owners of the shared lock is presented to all of them and they must sign-off until only one user is left in the list.

When a transaction commits, all "Stick-It"s holding intermediate values are discarded by making a photocopy as explained in Section 2. At some point, while committing a transaction the group of users may decide that all objects have values that are consistent with each other by some set of criteria. Then the users may choose to create a release, which relates all current versions of objects.

The object management user interface

The user interface to object management has been briefly explained before. Now we provide technical details of how the operations that are offered by this interface are realized in the

[3] Example adapted from [10].

S^3 object management system. The following entries are available in the "Objects" menu:

aggregate: This operation allows the creation of a new aggregate object. A set of objects, which can be either aggregates or cells, are selected with the help of the "Object-level" controls. The aggregate entry is then selected from the menu to create a new aggregate object.

disaggregate: One object is selected, and then the disaggregate entry is selected from the menu. The aggregate object is disassembled into its components as was explained earlier. One exception is that an spreadsheet cannot be disaggregated.

clone: One object is selected and then the clone entry is selected from the menu. A new object is created using the selected object as a prototype object. The new object is stored internally and the user must place it on some spreadsheet by pointing to a cell that will be the origin of the object (in S^3 the origin is always the upper left cell of the object).

message: When one object is selected and the message entry is chosen from the menu, a dialog box pops up to either add, delete or modify a method of the selected object. The dialog box presents a list of those messages that are handled by methods of the selected object and buttons to add, delete, edit and cancel.

protocol: This menu entry allows a user to modify the current collaboration protocol of a selected object. When this menu entry is chosen, a dialog box pops up showing the current protocol in a subwindow with basic editing capabilities.

print: The print entry sends a message print to the currently selected object.

describe: This entry produces the following information about the currently selected object in a pop up window: status, a list of current locks, description of its structure, messages handled, its collaboration protocol, current version, last release and date of last modification.

History management

It is important to record and analyze the history of group interactions of object-oriented collaborative systems in three different levels of interaction: 1) among objects, 2) between objects and users, 3) among users. These three levels need to be analyzed using different techniques. However, common to all three of them is the large amount of information generated, being difficult to review it with standard software tools.

In S^3 we are not interested in the first level of history management, applicable when auditing processes to every state of the system is required. The other two levels of history management are important to S^3. Level 2 refers to the interaction of the users via S^3 commands with the objects in S^3. Some software systems such as the UNIX C-Shell or the Emacs editor with the "electric history mode" feature provide simple command editors with the ability to edit and re-execute a past command. Also, the most popular software tools running on Apple Macintosh and MS-DOS platforms permit the undoing and/or redoing of the last given command.

Although this feature provides some facilities in S^3, the approach proposed in [7] is also of interest. Here a UNIX interface is studied introducing automatic learning techniques to increase the users' functionality of the interface. The interface is provided with the capability of recognizing patterns in the sequence of users' commands using heuristic reasoning. In our case, the system can define the sequence as a new command in the S^3 interface, or execute them directly after asking the involved users for permission. For example, this interface can discover that the users are performing a "commit" command approximately every 20 minutes, printing the values of some of the objets, etc. It is clear that this new ability can be used to modify system decisions, such as modifying the collaboration protocol based on the users behavior. Of course, this feature can be deactivated if the users do not want the system to learn.

The other type of interaction which is of interest to S^3, is the one among users. Specifically, we are interested in recording the decisions made by the users about objects, with the purpose of revising them in the future. For example, a user may want to recover a specific version of an object, review a decision made about the modification of certain method, or know about a previous release. Users have to annotate each object with their decisions during a commit. Also, it is necessary to annotate the commit command, since it is an element of decision in time. It is clear that the user who requested the commit will have to annotate it.

The annotations of decisions concerning objects are modeled by nodes in a hypergraph. Each node is associated with what we call a "history item", defined according to its type. Thus, if a node represents the version of an object, it will include text, object name, version number, name of the last user who modified it with date and time, and all the relations generated automatically by the system as explained below. The responsibility of annotation of these kind of nodes is assigned to the owner of the object. The annotations of a commit of transaction the annotations of a release contain alike information.

The decisions made over an object can have relations with other objects, which are represented in the hypergraph by arcs connecting nodes. These relations should be known to facilitate the process of navigation through the decisions. For example, a user may want to navigate across the decisions made in time by different users about an object, or perhaps across the decisions made by all objects modified by a specific user. Thus, during object annotation the users must specify their relation to other nodes in the hypergraph and the desired way of examining the history of such version.

The relations between objects are defined for "similar" objects. The user can define that two objects have this relation, or else, the system can automatically add new vertices to the hypergraph when it learns that two objects have a similarity relation, according to simple heuristics. For example, if two nodes model the history items of version k and version k+1 of an object, the system will include a similarity relation to the hypergraph. Other relations are discovered for objects of the same release, cloned objects, and objects that send or receive messages from other objects. The system also defines a similarity relation between two nodes A and B if the user frequently has to visit an intermediate node C (which is related to both A and B) to go from A to B.

Figure 3 presents part of a possible hypergraph, showing nodes for object k for commits at three different times: t_n, t_{n+1} and t_{n+2}. The nodes for the same time contain the annotated decisions of all objects for a commit, in other words, the history of decisions of the version of objects modified in a transaction.

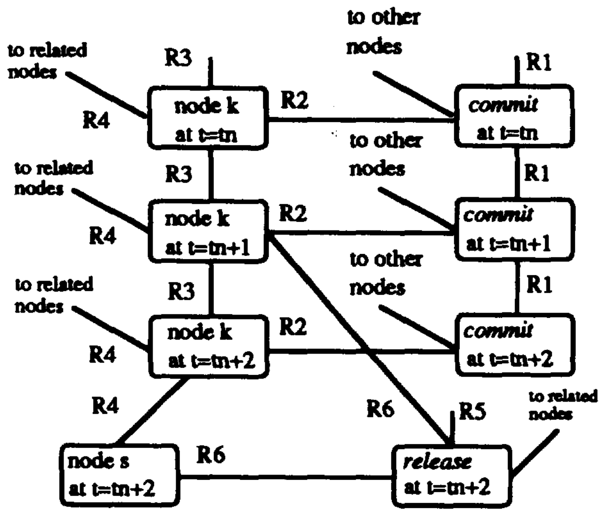


Figure 3: Hypergraph for navigation.

In figure 3, relation R1 is connecting history items of commits in successive instants of time. R2 connects commit nodes with other nodes involved during a commit. R3 is used to relate nodes of versions of objects. R4 connects similar history items. R5 connects the nodes from consecutive releases. Finally, R6 connects nodes from a release.

A user can only review history items for shared and owned objects, but cannot see any information about non-authorized objects. If she wants to analyze the history of decisions of an object, she will select the object and choose from the "History" menu one of the following four options. The "transactions" option will present a window showing the part of the hypergraph corresponding to the history of the object according to the history of transactions that influenced the object. In order to obtain the contents of the history items, she will have to click on the specific node.

The other options have similar procedures. The "version" option will show the annotations of the versions that affect the object. The "release" option will show the history of the releases that influence the object. The "relations" option will show the history of "similar" items to the selected object. Finally, "activate/deactivate learning" will activate/deactivate the automatic construction of similarity relations.

This system allows the selection of more than one object for history management purposes. In that case, the "version" option gets deactivated. However, the other three options show the common history of the selected objects.

When a user starts a commit operation, the system will present her a dialog box to annotate the decisions involved. At the same time, the system will present dialog boxes to all users who have modified objects during the transaction period.

Communication tools

The interface has a menu entry called "Communication" with tools intended to enable users to be informed about the work being done by their coworkers and report them any type of information.

In order to follow the mouse activities of other users, the user can select icons from the user interface representing users to be monitored, and then choose the "telepointer" option from the menu "Communication", as explained in section "Overview of the user interface". Of course, this feature can be used only to monitor mouse activities within the spreadsheets limits.

A user who wants to communicate with other connected users has to select the icons representing those users and click the option "message to" from the menu "Communication". A dialog box is opened in the senders interface, where she can type the text to be sent. The box has two buttons, "send" and "cancel" which allow the user either to send the message when she is ready or to abort it.

A user can obtain information about the status of other connected users or the objects in the spreadsheets. For the first case, she selects the icons representing those users in the control window and then choose "info about" from the "Communication" menu. The system will inform her about the name, idle time, and objects in which she owns a lock. For the case of information about objects in the spreadsheets, the user will select the objects and then choose "info about" from the "Communication" menu. The system will answer showing the commitment state, locks list, description of the objects (such as type, number of components, to what messages they respond to, protocol, current version, latest release including these objects, date of creation and date of latest commit).

A user can broadcast a message to all other connected users by choosing the "broadcast" option from the "Communication" menu. A dialog box is opened to the originating user, where she can write the message. The procedure is the same as in "message to".

Conclusions

In this paper, a novel collaborative spreadsheet to support "face-to-face" work is presented. Among its features, it has an object-oriented group interface, a shared object support system, object version control and group process history management. The user interface is provided with helpful mechanisms such as the "Stick-It"s and photocopies, two paradigms taken from day to day office work.

We use a client-server architecture in which data management is centralized in a server process for simplicity. Client processes handle the user interface and high level spreadsheet operations such as selection and printing. This system is currently under construction, a prototype of it will enable us to test its collaboration features.

Authors' addresses

David A. Fuller, Departamento de Ciencia de la Computación, Pontificia Universidad Católica de Chile, Casilla 306, Santiago 22, Chile. E-mail: dfuller@ing.puc.cl.

Sergio T. Mujica, Departamento de Ingeniería Informática, Universidad de Santiago de Chile, Casilla 10233, Santiago, Chile. E-mail: mujica@lascar.puc.cl.

José A. Pino, Departamento de Ciencias de la Computación, Universidad de Chile, Casilla 2777, Santiago, Chile. E-mail: jpino@dcc.uchile.cl.

References

1. G.A. Agha, ACTORS: "A Model of Concurrent Computation in Distributed systems", The MIT Press, Cambridge, MA (1987).
2. P. A. Bernstein, V. Hadzilacos, N. Goodman: "Concurrency Control and Recovery in Database Systems", Addisson-Wesley, Reading MA (1987).
3. A. Borning: "The Programming Language Aspects of ThingLab, a Constraint-oriented Simulation Laboratory", *ACM Trans. on Programming Languages and Systems*" 3, 4, 353-387 (1981).
4. T. Budd: "An Introduction to Object-Oriented Programming", Addison-Wesley, Reading, MA (1991).
5. O. Dahl, K. Nygaard, "SIMULA-An Algol Based Simulation Language", *CACM 9*, 671-678 (1966).
6. A. Goldberg, D. Robson: "Smalltalk-80: the Language and its Implementation", Addisson-Wesley, Reading, MA (1984).
7. H. Hoppe, R. Plotzner: "Inductive Knowledge Acquisition for a UNIX Coach", in Mental Models and Human Computer Interaction II, ed. by Ackerman and Tauber, North-Holland, Amsterdam (1990).
8. H. Lieberman, "Concurrent Object-Oriented Programming in Act I", in Object-Oriented Concurrent Programming, ed. A. Yonezawa and M. Tokoro, 9-36, The MIT Press, Cambridge, MA (1987).
9. Microsoft Corp.: "Microsoft Excel User's Guide", Redmond, WA (1992).
10. S. Mujica: "A Computer-Based Environment for collaborative design", Ph.D. Dissertation, University of California, Los Angeles, CA (1991).
11. R.R. Panko: "Office work". *Office Technology and People 2*, 205-238 (1984).
12. D. Patel, S. D. Kalter: "A Toolkit for Synchronous Distributed Groupware Applications", Proc. of Groupware'92 (D. D. Coleman, ed.), Aug. 2-5, 225-227, Morgan Kaufmann, San Jose, CA (1992).
13. J.M Smith, D.C.P. Smith, "Database Abstractions: Aggregations and Generalization", *ACM Transactions on Database Systems*, vol. 2, no. 2, 105-133 (1977).
14. M. Stefik, D. Bobrow, G. Foster, S. Lanning & D. Tatar: "WYSIWIS Revised: Early Experiences with Multiuser Interfaces", *ACM Transactions on Office Information Systems 5*, 147-167 (1987).
15. Sun Microsystems: "SCCS-Source Code Control System". Programming Utilities and Libraries Manual, 93-114. Revision A (March 1990).
16. W. Tichy: "Design, implementation and evaluation of a Revision Control System", Proc. of the Sixth Int. Conference on Software Engineering, Tokio, Japan (Sept. 1982).
17. T. Winograd, F. Flores: "Understanding Computers and Cognition", Addison-Wesley Publishing Company, Reading, MA (1986).