Collaboration with Spreadsheets

Skip Ellis & Carlos Maltzahn
Department of Computer Science
University of Colorado at Boulder
Campus Box 430
Boulder, Colorado 80309-0430
USA
{skip,carlosm}@cs.colorado.edu

Abstract

This paper presents a number of design concepts and ideas for a group spreadsheet. Unlike single user spreadsheets, these group spreadsheets should have specific features to enhance group communication, and to maintain a shared context. Rather than presenting a single design, we discuss alternatives and generalizable issues because the specific design can be very dependent upon the environment. The paper covers both low level architectural issues and high level collaboration issues. We will account for the fact that spreadsheets are used in collaborative environments and we will investigate how spreadsheets can be further integrated into such environments.

Keywords: Group Spreadsheets, Architecture, Collaboration Patterns, Design

1 Introduction

Collaborative activities occur less frequently than individual activities. In fact, many organizations are designed to minimize communication requirements in order to cut coordination overhead. This suggests that groupware features should be integrated with single-user features without obstructing them. Nevertheless they should be accessible and known to the user [8]. Therefore it is interesting to look at successful single-user applications and investigate possible extensions in order to facilitate collaboration. In this paper we look at extensions, and redesigns of spreadsheets for groups (see also [7]).

We will account for the fact that spreadsheets are used in collaborative environments and we will investigate how spreadsheets can be further integrated into such environments. We choose spreadsheets not only because of their success but also because they have proven to be a powerful and robust metaphor which is used in such different domains as logic programming [14], spreadsheets on Smalltalk objects [12], interactive graphics [15], and constraint maintenance for user interface specifications [10]. Thus exploring groupware extensions to spreadsheets might be fruitful to many application areas.

In this paper we will give a framework for classifying group spreadsheet architectures, extensions, and features. Section 2 gives a brief explanation and motivation for spreadsheets. Section 3 provides an architectural landscape for group spreadsheets, and section 4 elaborates spreadsheet facilities enabling collaboration among end users, local developers, and programmers. The paper ends with conclusions, acknowledgments, and a bibliography.

2 Spreadsheet Definitions, Advantages, and Limitations

A *Spreadsheet* is an application that is used to create and use *worksheets*. A worksheet is a matrix of *cells* of which each can contain a *value* and a *formula* that computes this value generally based on the values of one or more other cells. The formula can either be a built-in function or a *macro* which is a user-defined algorithm. A special case is a cell that has no formula in which case the user can enter and manipulate its value. Any change of a value in a cell triggers automatically the re-evaluation of formulas that refer to that cell. A *region* is a set of adjacent cells, e.g., a column. A *spreadsheet* document is an instance of a worksheet, i.e. the matrix of cells applied to a certain set of data.

Figure 1 shows the use of a very simple worksheet for calculating the sum of two numbers. The end user

figure=Fig1.ps

Figure 1: Example of a Spreadsheet Document

has entered 2 and 3 in the cells B2 and B3 respectively. As soon as 2 and 3 are entered the number 5 appears in the cell C4 based on the evaluation of the formula which has been defined by the developer of this worksheet. If the end user would change one of the values this change would be automatically propagated such that the value in C4 would change accordingly.

As discussed in [1] spreadsheets have the following advantages: automatic propagation of change; the user does not have to deal with flow control, parameter passing, and recursion; and there is no seam between programming worksheets and their execution. Some limitations of spreadsheets are: worksheets are difficult to debug because cells usually have complex interrelationships and non- mnemonic names; for the same reason worksheets are difficult to redesign; there are generally no constructs to support modularity and abstraction.

3 Forms of Architecture for Spreadsheets

Computerized spreadsheets are a successful single user tool. In practical everyday usage, they are also shared, by printing them, by electronically conveying them from user to user, or by other ad hoc means. This ad hoc sharing misses many group enhancement opportunities such as concurrent activity, distributed interaction, shared context, and automated coordination. In this section we discuss architectures for spreadsheets which are specifically designed for groups of people.

As an example of the functioning and utility of a group spreadsheet, consider the activity of putting together an advertising budget for a large distributed corporation. This is a collaborative endeavor and involves at least marketing, sales, and management. Suppose that each of these three subgroups is located in a different part of the country (or in different countries). A group spreadsheet could assist these subgroups to each construct, view, and update an appropriate view of the total budget spreadsheet. Corporate security may require that some sensitive information can only be viewed by management, so one task of the group spreadsheet is to provide access control mechanisms.

As asynchronous groupware, the spreadsheet would coordinate the update activity so that people can work at different times, or in parallel without introducing inconsistencies into the spreadsheet. The spreadsheet would also provide context so that everyone would be notified of changes since their last work session every time that they log into the system.

As synchronous groupware, the spreadsheet would also facilitate real-time interactions. At certain stages in the budget construction, subgroups may find that it is most productive to interact at the same time to do brainstorming or decision making as a group. It would be easy on the spreadsheet to do sensitivity analyses, and to try what-if scenarios. When one participant makes a change to one spreadsheet cell, all participants would immediately see the effects of the change and react to it. A group spreadsheet is not an isolated tool, but a system. Thus, it would allow participants to display and share other relevant documents, to see the group context, and to informally communicate as they are working.

3.1 Architectural Choices

When considering group spreadsheet architecture, we envision a generic group spreadsheet as consists of a set of hardware / software components dispersed among a set of participants. Frequently, participants have sub-systems to help them communicate and attain their goals. The primary hardware building blocks are computers and networks. The primary software building blocks are programs and data. Group spreadsheet architectures can be classified according to the connectedness of the sub-systems, and according to the level of centralization or decentralization of components. There is a spectrum of design possibilities from fully centralized to fully distributed architec-

tures. Data can be centralized, distributed, or hybrid. Control can be centralized, distributed, or hybrid. We assume that every component is somehow directly or indirectly connected to every other component. The computation of the user views as seen on participant screens (or other devices) can be done centrally, or done separately at each site, or hybrid schemes are possible.

We note that the computers may be large or small, shared or single user dedicated. Data and programs may also be large or small. User interfaces may be a variety of hardware and software ranging from dumb terminals to multi-media, multi- sensory all encompassing virtual realities. Communication may be LAN or WAN, minimal speed phone line or speedier fiber optic, broadcast or point-to-point. Given wide ranging options such as these, how do we select and match them to the requirements for group spreadsheets? This paper offers a categorization and motivation to answer this question.

Standard components of a spreadsheet that are candidates for distribution include spreadsheet values, worksheets, and formulas. These, together with other components such as the spreadsheet application code and the user interface, can be centralized, dispersed, or replicated. This three-way classification can be applied to any component. Centralized means that the component is kept all together on one machine. Dispersed means that the component is partially stored on more than one machine; e.g. value data of a two row spreadsheet could be dispersed so that row one data is stored on one computer, and row two data is stored on a different computer. If the same data is stored on more than one machine, then we say that it is a replicated architecture.

We can apply different choices to different spreadsheet components. For example, the basic spreadsheet manipulation program could be replicated at each participant's workstation; formulas could be centralized (stored on a server machine), and other information (worksheet values and structure) could be dispersed according to who owns which regions of the spreadsheet. A natural extension implied by the access control within groupware is to allow uneven, non- rectangular sets of spreadsheet cells. For example, I may have read access to all of row one, but access to only the first three cells of row two.

Interesting categories within the above three-way classification include the fully replicated systems in which all components are replicated at all sites, and the fully centralized systems in which all components are totally stored together at one site. Replicated systems have very high performance potential, exhibit tantalizing distributed computing challenges to implementors, and can be sculpted to elegantly fit the real-time groupware requirements of high currency and high concurrency. Centralized systems tend to be more secure and conservative; they can exhibit the advantages (and disadvantages) of the client/server architectures. The fully replicated architecture assumes a distributed interconnected computing environments in which the local computing facilities associated with each participant are capable of storing a copy of all the spreadsheet data, performing operations on this data locally, transmitting data (values, structure, and formulas) and operations as necessary, and presenting the data via varied user interfaces for varied participants. Given the decreasing price and size of computation power (processors) and the growing ubiquity of computer networks, this is a viable option, and may be a very popular option of the future. Significant technical challenges concerned with issues such as flexible distributed control and simultaneous update must be addressed in this architecture.

Usually, there is one site per user, and a site corresponds to one or more machines (the user's workstation and assessors); however some machines may run multiple sites. It is assumed that any site can communicate with any other site; they are typically connected via high speed networks. Within the category of fully replicated systems, there are still many possible choices of architecture. One must decide, for example, which modules are directly connected to which other modules, e.g. should the information repositories be directly interconnected?

3.2 Components

The spreadsheet application component contains and executes all application specific algorithms. It receives requests to perform application specific operations from its local user interface component, or over the network from a distant user interface. Unlike single user spreadsheets, it is necessary to have a group control component at each node of the network which listens to, and talks to other nodes for coordination and information access. Local requests are thus passed from the local user interface component to the local network control component to the application component. Modularity which separates the application component in this manner has the advantage that all application operations, local and nonlocal, have a single locus of delivery and execution at any site. Applications can be either collaboration aware in which case the application knows that it is

interacting with multiple participants, or collaboration unaware in which case the application does not know that it is servicing more than one client. Collaboration aware applications can use their knowledge of the participants and of the application domain to assist the user interface component to present tailored and enhanced information to the participants, to assist the network control component, and to do more sophisticated application computation. Collaboration unaware applications represent the vast majority of currently existing applications. This type of architecture is easiest to upgrade from single user to multiple user.

The information repository component is concerned with the storage structures and algorithms of the information objects being manipulated. This sub-system may range from an elaborate object oriented database [7] to a simple string manipulation program. The information may be standard database data (e.g. employee records,) or a set of documents, or multimedia modes of a hypermedia system, or CAD design objects. Since these structures are typically application dependent, this module is frequently directly connected to the application component. Architectural issues arise in this domain as to whether the set of information repository components should transparently handle all replication of data, and be interconnected as a separate subsystem, or whether replication should be a system visible property. This and other database issues arise as challenging architectural choices.

Another information repository issue that is affected by many architectural choices is group response time for real time synchronous spreadsheets. Quick response to actions of multiple users requires a multi-user architecture that insures fast response and consistency. In order to get fast response the communication overhead that is required to change a shared state has to be minimized. This is achieved by replicating an application for every user. However, it is much harder to maintain real-time consistency in replicated architectures than it is in non-replicated [9]. In the latter case all changes of a shared state are propagated through a central server which, however, becomes a bottleneck in communication, thus slowing down response.

A solution to this trade-off between response and consistency is to provide a server for each extension, e.g., a lock server, an informal channel server, a link server, and a white board server. The idea is to alleviate the bottleneck effect by distributing fairly independent services on different servers. Another measure is to let user front ends manage as much detail as possible such that only abstract signals are exchanged. An example for such an approach is the X Window system that keeps most of the screen management local.

The user interface component implements presentation and user input functions (see also [3]). It receives input (e.g. keystrokes) from its local participant which it must pass to the application component. It receives data from the application component which it must present in an appropriate manner to the local participant. Frequently the user interface component cannot simply spit out the next output, but needs to input the complete set of objects which form the relevant context, perform a transformation, and then regenerate a new output object set. If the participant manipulates the application by sophisticated input devices in interesting non-procedural ways, this must be mapped to standard high level input and output events that are expected and can be processed by application components. If the participant wants to view the items of data which are output by the application as graphs or as animations, then these transformations must be handled by the user interface component. The user interface component describes and monitors the participant's input/output environment, and is directly connected to the local network control component so that both local and non-local communication can occur without the application component necessarily being collaboration aware.

In this short section, it was not possible to cover all of the architectural options. For example, there is a need within many groupware systems for components to migrate from site to site for load balancing and for efficient transaction processing. In a workflow system there may be static data concerning the organization, and transient data to move among sites as different participants utilize the data of different stages of the transaction.

In conclusion, we see that there exists a wide variety of architectural possibilities. Selection of appropriate architectural elements for a particular purpose and environment is highly dependent upon the system users and the roles they play. This is discussed in the next section.

4 Forms of collaboration with spreadsheets

In [11] Bonnie Nardi describes two studies of the usage of single-user applications, namely spreadsheets and X Windows. Both studies indicated that people collaborate in order to master these applications. Although the domains of these applications are very

different, these studies showed very similar patterns of collaboration which involved three roles:

- *End users* who have no or only very little programming experience, are not interested in learning any technical details about computers and are just focussed on their own domain specific interests.
- *Local developers* who work in the same domain as the end users but are more interested in computers and like to discover more advanced functionality of the application. They play the role of translators who mediate the need of end users to the programmer and convey new functionality to the end users. Bonnie Nardi describes this role sometimes as *productivity gardeners*.
- *Programmers* who are professionally trained, provide programs to end users and local developers and help them learn new things.

With these roles, three different kinds of collaboration can be identified that differ in their possibilities of support: The end user is interested in *sharing values* with other end users. These are worksheets applied to a given set of data. The computation of the data is only relevant on a very abstract level and fully expressed by the layout of the worksheet. The layout encodes domain specific knowledge which is communicated through spreadsheet documents. Typical patterns of collaboration are routing spreadsheet documents through the organization, discussing the outcome of spreadsheet computations, and sharing what-if scenarios.

The local developer is the prime creator of worksheets. S/he knows the domain and the needs of end users. In order to provide optimal worksheets to end users, local developers are interested in *sharing worksheets* with other local developers and end users. Typical tasks are the collaborative design of new, large spreadsheets, and the discovery of reusable worksheets designed by other local developers.

The programmer typically provides formulas, and macros. Typically, there are a lot of application domain specific functions and macros. The problem of collaboration at this level is very similar to software development projects. There are generally a vast variety of formulas and macros to keep track of. Considerable coding time is saved by reusing existing functions and combining them in new ways. However it is often the case that finding out whether a particular formula is useful for the implementation of a new formula is more time consuming than just programming the new

figure=Fig2.ps

Figure 2: Typical Collaboration Pattern For Complex Single-user Applications

formula from scratch. Thus, programmers are interested in an infrastructure that enables them to share information about formulas.

Technically, spreadsheets applications offer a seamless transition between worksheet design and worksheet application: One can enter data, add more formulas and change the layout within the same session. However, the observations of Bonnie Nardi seem to indicate that this feature is mostly used by local developers and programmers and that end users tend to avoid worksheet design as much as possible. This leads to distinctions we make in the following sections of this paper.

4.1 Sharing Values

In this section we will describe various design features that are intended to support end users sharing spreadsheet documents. In particular, we will present various approaches o value sharing within a single spreadsheet document and across multiple spreadsheet documents.

4.1.1 Same Spreadsheet Document

One of the most straight forward support for sharing values is to extend spreadsheets applications such that users can have shared access to the same spreadsheet document. Such an extension can either support same time interaction where users share a spreadsheet document during a single session, or different time interaction where users work on a shared worksheet during individual sessions. Both types of interactions need to facilitate coordinated access. However, same time interaction allows for somewhat different means of coordination as different time interactions. Some of the features described below can be used in both interaction forms. Especially features in different time interactions can also be used within a session of same time interaction.

Same Time Interaction. During same time interaction coordinated access can be facilitated by (1) *sharing* and *synchronizing* views of different users on a particular part of the spreadsheet document, and (2) by *informal channels* such as a voice connection. Sharing views can be conveyed by highlighting the region that a particular user is currently watching. Synchronizing views allows a user to "switch" to another user's view. The synchronized view can be either loosely aligned in which case the user only gets a snapshot of another user's view of the spreadsheet document, or closely aligned where the user can also watch the other user's view changes, e.g. scrolling. The advantage of synchronizing views is that users can quickly gain common ground for discussions that develop over informal channels: if discussion participants all have the same view on a shared spreadsheet document then much less time is needed to describe spreadsheet document views by natural language. Also, user's can quickly switch between different areas of the spreadsheet document under discussion. The advantage of sharing views by highlighting regions is that users see those views relative to a context that is solely determined by them individually. Switching between views causes massive context changes and can lead to temporary loss of orientation.

Spreadsheet documents are objects with complex interconnections, i.e., a change of a value can trigger something in such a distant location that one view cannot include the changed cell and the triggered cell. In order to share information in a more compact way, one idea is to make dynamically updated graphics available to participants. This extends the notion of views by means of data representation.

Informal channels can be supported by voice channels or chat boxes. With multicasting capabilities one can start subdiscussions without interrupting ongoing discussions. It is important that such an informal channel management does not completely cut off participants from other on-going discussion threads but make seamless transitions between different discussion threads possible. A good metaphor is smalltalk as it occurs at parties where conversations spontaneously split and merge because of spatial and semantic proximity.

Different Time Interaction. During different time interaction coordinated access can be facilitated by *annotations* which can be recorded voice, informal written notes, or email that refers to certain location of a spreadsheet document or can include fragments of it. A more formally controlled approach is the use of *layers* which can be seen as a hierarchy of shared and individual end user workspaces. Each end user has an individual layer in which s/he can freely edit and compute data. Data can be imported from and merged with a layer common to a group which in turn lies on a global layer.

Routing lists that are attached to worksheets are useful for ensuring correct shipping of spreadsheet documents to the right people. A more sophisticated mechanism is the concept of *ordered regions* where data has to be first completely entered and processed in one region before a successive region can be accessed. Thus, the concept of ordered regions imple-

ments a workflow system application defined by a worksheet.

Coordination of different time interaction is greatly facilitated by notification mechanism. Notification can be placed by assigning *input requests* to users w.r.t. certain cells or regions. Input requests are represented as special formulas assigned to cells that notify the corresponding user that input is required. *Notification functions* are formulas that send notifications depending on cell values and thresholds.

A very versatile infrastructure for routing spreadsheet documents and notifying end users is provided by email. A well integrated email system would be able to send computationally active fragments of spreadsheet documents or whole spreadsheet documents and have various annotation mechanisms to comment on these fragments.

4.1.2 Different Spreadsheet Documents

Different spreadsheet documents can be associated by *links* which couple cells in different spreadsheet documents. Changes in one spreadsheet document can thus propagate along these links to other spreadsheet documents. Links either define very tight coupling, i.e. changes are visible almost immediately across all interlinked spreadsheet documents; or propagation can occur only periodically, say once a day. Links can span wide geographical distances and are especially useful for collaborations that involve remote sites.

4.1.3 White Board

Another idea of sharing spreadsheet documents is the metaphor of *white boards*. White boards have proven to be very valuable in group meetings as a shared drawing and writing space. Being able to paste fragments of spreadsheets on a whiteboard and mixing it with typed or handwritten text and free hand drawing is very valuable for discussing domain specific computations and their results. Ideally, a spreadsheet white board is not only a shared display but also offers the computational capabilities for performing shared simulations. Although white boards are usually used in same time interactions they can also serve as bulletin boards in different time interactions.

Such white boards are an example of support of a "double level language", one level being the implicit communication through artifacts, and the other level being the explicit communication through speech and ad hoc notes [13]. It is assumed that any collective activity requires effective communication at both levels.

4.1.4 Locking vs. Orientation

Traditionally access to any shared resource is coordinated via exclusive *locking*. In spreadsheet documents locks are usually placed on regions. Locking can make regions invisible (in this case a region is the union of a set of columns and a set of rows); obscured in which case the existence of cells is visible but not their content; or readable in which case the content is visible but cannot be changed except by the owner of the lock.

Even in the case where locks leave the content of cells readable locking excludes certain types of collaboration. In particular people cannot closely interact on a shared region. Being forced to pass locks back and forth artificially sequentializes the work. As reported in [5] the experience with the GROVE editor indicated that groups easily and naturally develop very sophisticated *social protocols* in order to coordinate their access to the shared artifact. Most importantly, people developed social protocols that would have been incompatible with locking.

An approach that is based on these findings is achieving coordinated access by issuing warnings and extended means of *orientation* instead of locking. By means of orientation we think of services that make people aware of each other's activities within the shared resource. An examples for such a service are *clouds* [4] which indicate regions of change to all group members without making the actual result of the change visible. This enables users to selectively focus on one change at a time and to compare it with the original state. In order to switch between the original state and the resulting state users can click on the corresponding cloud. Clouds are aging, i.e. their color changes over time. This gives users an overall picture of how up-to-date their view is and where the current regions of change are located. People can develop their protocols based on the contextual information such a service provides and will not be forced to comply with a purely technically motivated mechanism like locking.

4.2 Sharing Worksheets

This section discusses designing and sharing of worksheets among local developers. We will distinguish between the shared design of worksheets and their shared reuse.

4.2.1 Shared Design

One of the disadvantages of spreadsheets mentioned in section 2.0 ("Spreadsheet Definitions, Advantages, and Limitations") is the lack of modular concepts. In

shared design this lack is exacerbated by the additional need of design coordination. A suitable concept of abstraction for worksheet design seems to be the concept of *regions*. So far we have only referred to them as collections of adjacent cells. However, this concept can also be used to create an abstract layout without specifying individual cells. Principle relationships can then be defined between these regions such that the interface of a region is well defined.

In traditional spreadsheet applications cells are referred to as pairs of letters and numbers. This makes formulas hard to understand since these references do not mean anything unless one has an intimate knowledge of the spreadsheet layout. With the advent of regions as a modularization concept one needs to name them. Since these names have to be consistent over the whole shared worksheet it is appropriate to have a *name server* for regions. This also solves the problem of unreadable formulas since regions can consist of only one cell so that cells can be named with more mnemonic names. Name servers can also be used across worksheets in order to design links as mentioned in subsection 4.1.2 ("Different Spreadsheet Documents").

While developers design a region in a shared worksheet they need to have some insulation from other people's operations on the same worksheet. Especially if a developer is in an early testing phase and the region has a lot of relationships to other cells, the changes that are propagated through those relationships can be very confusing to the tester. Therefore, the developer needs to be able to temporarily and selectively *sever* relationships between regions and incrementally reconnect them as integration of the individual regions progresses.

Another important aspect of shared design is the storage and management of design history. In the following we will distinguish between short term history services, long term history services and services that are based on heuristics which are condensed from multiple design histories.

Short Term History: In subsection 4.1.1 ("Same Spreadsheet Document") we outlined a layered scheme of coordinated access to a shared spreadsheet document. This scheme allows for a nice integration of *command history* in which each layer keeps its own history of events. Command histories are even more useful if they not only allow command reuse but also *command undo*. This means that commands can be retracted from the history. Depending on the sophistication of such an undo service it allows the user to either undo the last command, or undo the commands in reverse command sequence, or undo any command in the history. The undo service would then compute the consequences of that retraction. In non-trivial cases this computation might require user intervention. In the case of an undo on a shared layer it might even require the intervention of multiple users.

Long Term History: It has proven to be very valuable to record the rationale of a design. Especially forms used in offices, e.g. worksheets, bear a lot of domain specific wisdom in them. Changing or abolishing office forms is therefore risky if the design rationales of these forms are not known. Design rationales should be formally connected with the design artifacts, i.e. cells and regions of particular worksheets. They also should offer a semantic structure such that related design issues are also formally related (see for instance [2]).

Heuristics: Studies in [6] show that recording of design rationales is not enough. Some mechanisms are needed to deliver the right part of a design rationale to the developer at the right time. This in turn motivates the developer to maintain design rationales. *Critiquing systems* can serve as such a form of delivery. A *critic* can watch a developer's design action and issue some form of feedback. This feedback can be either reflected by messages or by the behavior of the used design tool. Ideally, critiquing systems can automatically infer their behavior from recorded design rationales.

4.2.2 Shared Reuse

Local developers working on the collaborative design of worksheets need similar support as developers in a software development project. A very powerful service in a software development project is version control. Developers can check out versions of worksheets, change them and check in a new version. These versions can be presented in a catalog according to multiple criteria that might be useful to identify worksheets that are reusable in a particular development project. For example, in the newest version of Lotus 1-2-3 such a version catalog is structured around worksheet authors. These authors are local to an organization and therefore their names are laden with context information about their organizational history and the projects in which they are involved. In shared development efforts authors can also be groups of people. A version catalog is also an important service for end users who want to find out whether a worksheet already exists for a given task.

The structure of such a catalog is essential and sometimes hard to infer automatically. In such cases a

more interactive scheme might be appropriate where developers classify their worksheet versions according to a given standard. The designer of a version catalog presentation needs to choose a structuring solution from a spectrum between completely automatic structuring and pure manual structuring depending on what information is available on the content of versions.

4.3 Sharing Formulas

Programmers are employed to provide computational functionality - in our case, formulas and macros for worksheets. They are the ultimate experts for spreadsheet functionality within an organization but compared to local developers they generally know little about the organization's domain. However, programmers as well as local developers both face the problem that the amount of domain specific formulas and macros accumulated over time is overwhelming. The knowledge about what's there, how good it is and how can it be used is essential for local developers (and for programmers for reuse purposes). However, only local developers can tell how well a formula or a macro fits into their domain specific solutions. Programmers need that feedback in order to improve their design.

Thus programmers and local developers need to have some way to communicate this information. Since the amount of information is large and of long-term value a persistent and highly structured form of medium is more appropriate than just the exchange of messages as in telephone, email, or letters. Even if that information is recorded and kept in folders, it is difficult to access because it usually does not offer enough structure for effective and efficient access. An example of a highly structured medium is a hierarchical space of categories populated with formulas and macros. All communications are either associated with a category or a specific formula. Examples of such communications are pointers to documentation, experience reports, bug reports, pointers to worksheets which are using a particular formula or macro, and pointers to other (better) formula/macro alternatives. A medium designed in such a way would facilitate reuse and provide feedback for programmers. Furthermore, in this case feedback of local developers is not only communicated to programmers but also to local developers among themselves which provides important information for worksheet design.

A similar medium can be used to facilitate communication between local developers and end users regarding worksheets. This can be nicely integrated with the version catalog as mentioned in subsection 4.2.2 ("Shared Reuse").

5 Conclusions

This paper has addressed architectural issues and collaboration issues for group spreadsheets. The architectures include centralized, dispersed, and replicated components. We presented a number of different ways of extending spreadsheets applications in order to support collaboration. We pointed out three basic kinds of extensions supporting different roles in an organization: Extensions for *sharing values* which support end users as they are the primary creators of values; extensions for *sharing worksheets* which support local developers to design worksheets and reusing them; and extensions for *sharing formulas* and macros which support programmers and their communications with local developers. This classification of extensions is motivated by results of empirical studies reported in [11] which identified the roles of end users, local developers, and programmers. Furthermore it pointed out the importance of local developers which serve as translators of domain specific problems of end users into problems of worksheet design and specifications for formulas and macros which are meaningful to programmers. Bonnie Nardi refers to these roles as *gardeners* who are gardening productivity.

Since gardeners play such an important role in productivity of end users it is important to support them with systems (i.e. extensions of spreadsheets applications) that facilitate self control of their services and enable the end user to take optimal advantage of these services. In particular, as we described in the previous sections, gardeners need to have support in design (including various forms of design histories and critics), reuse (of formulas, macros, and worksheets), and in communicating with end users.

In this paper we only looked at only one example, i.e., spreadsheets, of extending single user application in order to support collaboration. However, the various aspects of integrating spreadsheets with collaborative environments is likely to be similar for other single-user applications.

6 Acknowledgments

This paper owes a great debt of gratitude to the University of Colorado graduate students in the Groupware Seminar, CSCI 7000, Spring of 1993 and 1994. In these classes, students were given an assignment to design a group spreadsheet. We appreciate

the enthusiasm and creativity of these students, and acknowledge that many of their good ideas appear in this paper. For completeness, all students are listed here:

Samy Ababatein, David Addoms, Amy Baltzer, Ruth Bittner, Terry Boger, Catherine Brand, Daniel Brodsky, James Connolly, Jonathan Cook, Anne Dahl, Simo El-Khadiri, Jeff Frankenstein, Christopher Gantz, Adam Griff, Jim Guyotn, Grant Jacoby, Brian Jones, Brian Keyser, Anita Levinson, Chris Lewis, Stefanie Lindstaedt, James MacLeod, Komanana Manu, John McAlister, Amy Morgenstern, Nicole Lawrence, Sung-Hee Nam, Pamela Romano, Frederico Serrano, Kurt Simmons, Boerre Steen, Christina Vaughn, David Vollmar, Vance Wilson.

References

[1] C.H.Lewis and G.Olson. Can psychology lower the barriers to programming? In *Second Workshop on Empirical Studies of Programmers*, Norwood, N.J., 1987. Ablex.

[2] J. Conklin and M. Begeman. gIBIS: A to9ol for exploratory policy discussion. In *CSCW'88*, pages 140–152, Portland, Oregon, September 1988.

[3] P. Dewan and R. Choudhary. Primitives for programming multi-user interfaces. In *UIST'91*, pages 41–48. ACM, 1991.

[4] C.A. Ellis, S.J. Gibbs, and G.L. Rein. Design and use of a group editor. In G. Cockton, editor, *Engineering for Human-Computer Interaction*. North-Holland, Amsterdam, 1990.

[5] C.A. Ellis, S.J. Gibbs, and G.L. Rein. Groupware: Some issues and experiences. *Comm. ACM*, 34(1):39–58, January 1991.

[6] G. Fischer, A. Lemke, T. Mastaglio, and A. Morch. The role of critiquing in cooperative problem solving. *ACM Trans. on Informational Sciences Journal*, 9(2):123–151, 1991.

[7] I. Greif. Designing group-enabled applications: A spreadsheet example. In D. Coleman, editor, *Groupware'92*, pages 515–525. Morgan Kaufmann Publishers, 1992.

[8] J. Grudin. Groupware and social dynamics: Eight challenges for developers. *Comm. ACM*, 37(1):92–105, January 1994.

[9] J.C. Lauwers, T.A. Joseph, K.A. Lantz, and A.L. Romanow. Replicated architectures for shared window systems: A critique. In *Conference on Office Information Systems*, pages 249–260, Cambridge, MA, April 1990. ACM.

[10] B. A. Myers. Graphical techniques in a spreadsheet for specifying user interfaces. In *CHI'91*, pages 243–249, New Orleans, Louisiana, April 1991.

[11] B. Nardi. *A Small Matter of Programming*. The MIT Press, Cambridge, MA, 1993.

[12] K.W. Piersol. Object oriented spreadsheets: The analytic spreadsheet package. In *OOPSLA'86*, page 385, New York, 1986. ACM.

[13] M. Robinson. Design for unanticipated use.... In *ECSCW'93*, pages 187–202, Milan, Italy, September 1993.

[14] M.H. van Emden, M. Ohki, and A. Takeuchi. Spreadsheets with incremental queries as a user interface for logic programming. *New Generation Computing*, 4:287, 1986.

[15] N. Wilde and C. Lewis. Spreadsheet-based interactive graphics: from prototype to tool. In *CHI'90*, pages 153–159, Seattle, WA, April 1990.