Enhancing Creativity with Groupware Toolkits

Saul Greenberg

Department of Computer Science, University of Calgary, Alberta T2N 1N4 CANADA
saul@cpsc.ucalgary.ca; http://www.cpsc.ucalgary.ca/~saul/

Abstract. Effective groupware toolkits not only make it possible for average programmers to develop groupware, but also enhance their creativity. By removing low-level implementation burdens and supplying appropriate building blocks, toolkits give people a 'language' to think about groupware, which in turn allows them to concentrate on creative designs. This is important, for it means that programmers can rapidly generate and test new ideas, replicate and refine ideas presented by others, and create demonstrations for others to try. To illustrate the link between groupware toolkits and creativity, I describe example toolkits we have built and how others have leveraged them in their own work.

1 Introduction

The first true vision and implementation of real time groupware happened at the Fall Joint Computer Conference in 1968, where Douglas Engelbart demonstrated many important concepts including terminal-sharing, multiple pointers and turn-taking over shared displays, and audio / video conferencing [5]. This tour-de-force was far ahead of its time, and it was not until 15 years had passed that a few other researchers began replicating and extending Engelbart's ideas, most notably Sarin's [15] and Foster's [6] PhD theses. Shortly afterwards, the field of Computer Supported Cooperative Work (CSCW) formed (late 1980s). A veritable explosion of research followed, and CSCW is now considered a relatively mature discipline. In spite of this history and the many research advances made since 1968, groupware has *not* made many inroads into the everyday world, especially when compared to the advanced made in desktop publishing (which also started with Engelbart). Why is this the case?

I argue that a key technical problem is that average programmers do not have sufficient tools to design, prototype and iterate real time groupware. Current available tools are far too low level. For example, most commercial toolkits provide basic communication facility such as socket programming or remote procedure calls, but little else. This has several serious implications:

- most programmers eschew groupware development because it is too hard to do
- those who do decide to develop groupware place most of their creative efforts into low level implementation concerns
- resulting designs are often fairly minimal ones, with little attention paid to necessary design nuances (even ones well known in the CSCW literature) simply because they are too hard to implement.

Cite as:
Greenberg, S. (2003) Enhancing Creativity with Groupware Toolkits. Invited keynote talk. Proceedings of the CRIWG ' 2003 9th International Workshop on Groupware (Sept 28 - Oct 2, Autrans, France), LNCS vol. 2806, 1-9, Springer-Verlag.

The consequence of inadequate development tools is that—excepting members of a small specialist community—there has been relatively little evolutionary development and dissemination of groupware systems. The solution is that we as a community must develop groupware toolkits appropriate for the everyday programmer. By appropriate, I mean that a good groupware toolkit should:

- be embedded within a familiar platform and language in common use so that people can leverage their existing knowledge and skills
- remove low level implementation burdens common to all groupware platforms (e.g., communications, data sharing, concurrency control, session management)
- minimize housekeeping and other non-essential tasks
- encapsulate successful design concepts known by the research community into programmable objects so they can be included with little implementation effort
- present itself through a concise API that encourages how people should think about groupware
- make simple things achievable in a few lines of code, and complex things possible.

I believe that effective groupware toolkits not only make it possible for others to develop groupware, but also enhance their creativity. If we remove low-level implementation burdens and supply appropriate building blocks, we provide people a 'language' to think about groupware, which in turn allows them to concentrate on creative designs.

While some may question this premise as over-simplistic, we must recognize that toolkits in other domains have a proven record of enhancing creativity in the general programming community. For example, GUI toolkits, such as Visual Basic, supply a large set of interface components (widgets) and an interface builder for laying them out on the display. Because GUI toolkits encourage programmers to think in terms of widgets, programmers have created a plethora of applications that 'glue' these components together in interesting ways [14]. Another example is Macromedia's Flash, which encourages both programmers and non-programmers to think in terms of scripted animations. Because Flash makes it easy to do, we now see a proliferation of many quite amazing animations on the Web.

To illustrate the link between groupware toolkits and creativity, I will provide in this paper several examples of groupware toolkits we have built and how students—both graduates and undergraduates—have leveraged these tools in their own work. Before doing so, I want to explain a recurring pattern that emerged over the years in our group's investigation of the human and technical factors of groupware, and how recognizing this pattern has led to our appreciating the value of good tools.

1. ***Human factors perspective.*** Our initial goals in our groupware projects are typically oriented towards human factors. Essentially, we wanted to understand how people interact together when using a particular style of yet-to-be developed groupware application. We would then generalize this understanding to inform other groupware designs.
2. ***Initial prototype.*** Next, we would set about building the first version of the groupware application. This typically involved huge effort as measured by lines of code, time, learning, failed attempts, debugging, and so on. In spite of this effort, the result was often a fragile and rudimentary system.

3. ***Prototype testing.*** We would then have people try out this prototype. Because are first design attempts, we often saw major usability problems that required fixing.
4. ***Iterative prototypes.*** To fix these usability problems, we would then iteratively redesign the prototype. Yet this often proved impractical to do. The prototype code was often too complex to change, or the system itself was too fragile. Redesigning from scratch, while possible, was onerous due to the time involved.
5. ***Retrenchment: building a groupware toolkit.*** We would then realize that—in the long run—iterative prototyping would be far easier if we took the time to build a robust toolkit. Thus we would set ourselves a new technically-oriented goal, where we would delve into the challenges of understanding and building this toolkit and its accompanying run-time architecture. This often meant that we had to defer work on our main human factors goal.
6. ***The payoff: rapid prototyping.*** After building the toolkit, we would release it to our internally community. There would then be an explosion of activity. Those with core interests in the human factors challenges would rapidly develop and test a variety of groupware interaction techniques and applications. Those with interests lying elsewhere would often create innovative groupware applications as a side project just to satisfy their own curiousity.
7. ***Improvement and dissemination.*** Because we would develop the toolkit and applications side by side, we would bring good application ideas back into the toolkit as building blocks that could be trivially included in other applications. Examples included common architectural features, widgets, interface components, and interaction techniques.

In the remainder of this paper, we will briefly summarize our experiences with several toolkits that we developed for three groupware domains: real time distributed groupware, single display groupware (SDG), and physical user interfaces.

Toolkits for Real Time Distributed Groupware

My first foray into groupware echoed the above pattern. In 1989, I and my students decided to build a simple drawing application for distributed participants designed around John Tang's human factors observations of how small design teams draw together [17]. The result was GroupSketch [11], a program written over several months by student Ralph Bohnet. While simple in functionality, the actual program was quite complex. Its more than 5000 lines of code had to deal with many things: setting up the basic communication architecture and protocol for data exchange, creating a session manager that would let people join an existing conference, managing an event stream that handled simultaneous local and remote user actions, creating labeled telepointers for each participant, and creating the actual drawing surface and actions. All this had to work efficiently so that the participants would see no noticeable lag, and this required quite a bit of time and experimentation to get right. Shortly after, student Roseman built GroupDraw [11], an object-based drawing system. As with GroupSketch, the majority of the GroupDraw programming effort went into developing the underlying architecture and worrying about performance issues *vs* designing the actual group-drawing interface.

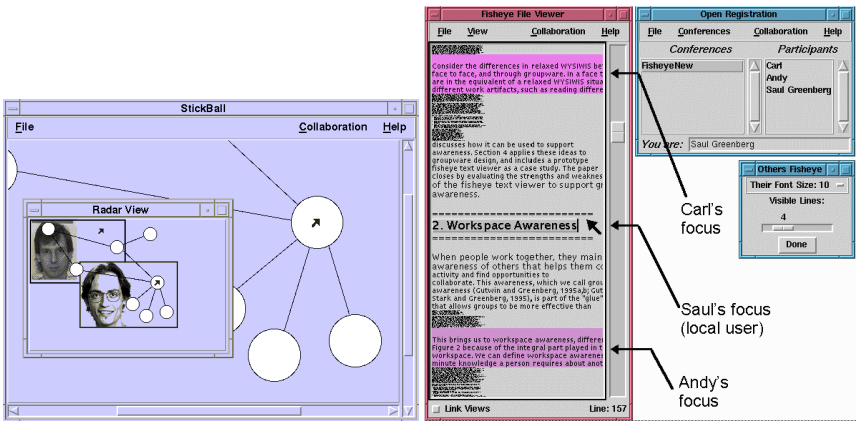

Fig. 1. a) the portrait radar view, **b)** a fisheye editor; both show where others are working

Both systems worked well enough to give us insights into what we wanted to do next, but were just too large and too finicky to extend. Consequently, we turned our efforts into developing GroupKit, a toolkit for building distributed real time groupware applications [10]. Our experiences with GroupSketch and GroupDraw helped us identify elements common to real time distributed groupware applications, and our GroupKit design would provide these elements to the programmers.

- A run-time architecture automatically managed processes, their interconnections, and communications; thus programmers did not have to do any process or communications management. This came for free.
- Session managers let end-users create, join, leave and manage meetings. A selection of session managers came as pre-packaged interfaces (one is visible on the top right of Fig 1b), and the programmer could use these 'out of the box'. However, the programmer could craft their own session manager if they wished.
- A small set of groupware programming abstractions let a programmer manage all distributed interactions. Through an RPC-like mechanism, the programmer could broadcast interaction events to selected participants. Alternatively, the programmer could manage interaction via a shared data model: programmers would then think about groupware as a distributed model-view-controller system. Local user actions would change data in the shared model, and remote processes would detect these and use the altered data to generate the view.
- Finally, groupware widgets were included that let programmers add generic groupware constructs of value to conference participants. Our first widgets were telepointers, which a programmer could add with a single line of code. Later widgets included awareness widgets such multi-user scrollbars and radar views.

GroupKit considerably simplified groupware development e.g., using GroupKit we reimplemented GroupSketch in a few hours in less than a page of code. What was more important is that we could now explore various design ideas through rapid prototyping. For example, our group created a flurry of systems illustrating different methods for supporting awareness within a visual workspace, sometimes turning around several different design ideas in a single day. Figure 1 shows two example

systems illustrating how people within a group could maintain awareness of others' actions [13,9]. Because we could now try out our ideas, we could quickly determine which ones were worth pursuing and which were not.

Fig. 2. Presence History (M. Boyle)

While GroupKit was very useful for prototyping real-time distributed graphical user interfaces, it did not handle multimedia. Consequently, we built a new toolkit called the Collabrary [3] that would let us rapidly prototype multimedia groupware. It provides extremely easy access and manipulation of multimedia information. For example, discovering a video camera and acquiring an image takes two line of Collabrary code. It also provides a straightforward API that lets people distribute this information between groupware program instances through a shared data model. Similar to GroupKit, students began creating multimedia groupware because it was easy to do. One example is the Presence History system included in Figure 2, built by Michael Boyle. The remote video is displayed, and a graph at its bottom shows a history of other people's presence and activity in the scene, determined by image analysis. The entire program is 38 lines of code. Another much more complex example is Michael Rounding's Notification Collage [12], where colleagues post multimedia elements onto a real-time collaborative surface that all members can see.

Toolkits for Single Display Groupware

Researchers in Computer Supported Cooperative Work (CSCW) are now paying considerable attention to the design of single display groupware (SDG) i.e., applications that support the work of co-located groups over a physically shared display [16]. Our own work in SDG began with an investigation of transparent menus as an interaction technique that would minimize how people working close together would interfere with each other [19]. Fortunately, Bederson and Hourcade [1] had developed the MID toolkit that let one access multiple mice from Microsoft Windows'98. While it was still difficult to develop SDG, their toolkit made our own development a reasonable prospect.

The problem was that MID did not work with later versions of Windows. Consequently, we decided to re-implement and significantly extend some of the ideas in MID in our own SDGToolkit [18]. SDGToolkit automatically captures and manages multiple mice and keyboards (as does MID), and it also presents them to the programmer as uniquely identified input events relative to either the whole screen or a particular window. Unlike MID, it transparently provides multiple cursors, one for each mouse. To handle orientation issues for tabletop displays (e.g., people seated across from one another), programmers can specify a participant's seating angle, which automatically rotates the cursor and translates input coordinates so the mouse behaves correctly. Finally, SDGToolkit provides an SDG-aware widget class layer

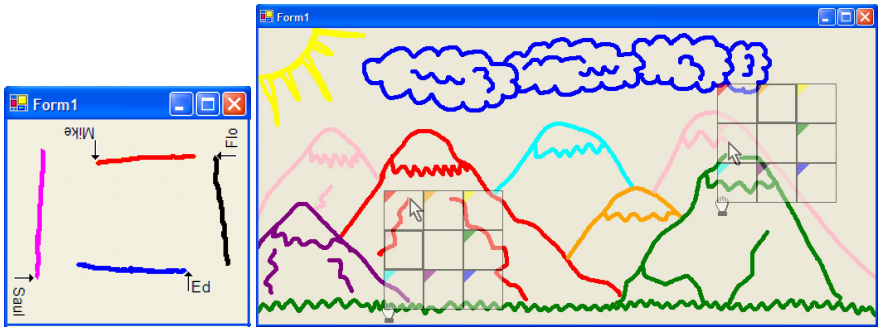

Fig. 3. a) SDG TableTop Drawing; **b)** SDG MagicLenses

that significantly eases how programmers create novel graphical components that recognize and respond to multiple inputs.

With SDGToolkit, simple things are simple. For example, Figure 3a illustrates a simple drawing application designed for a square tabletop with four seated people, one per side. Cursors and text labels are oriented appropriately, and the person's mouse behaves correctly given their orientation. It is written in 20 lines of code.

Another example illustrates how students Nicole Stavness and Edward Tse reimplemented Xerox PARC's ToolGlass interaction technique as an SDG widget. Each user has two mice. Using the first mouse in with their non-dominant hand, each moves his/her toolglass around. With their other hand and mouse, they click through the lens to choose a color. Their programming effort to manage and identify multiple input devices and to package it up as an SDG widget was relatively small; instead most efforts went into the creative aspects of the ToolGlass graphics.

More recently, we received a DiamondTouch surface from MERL, which detects multiple simultaneous touches by multiple people and that reports them to a programmer through a basic SDK. We created the DiamondTouch toolkit that wraps this SDK and adds extra capabilities to it, considerably simplifying how people program multi-user / multi-touch applications [4]. Similar to our SDGToolkit, the toolkit identifies multiple inputs on a per-user basis. It generates events that reports when people tap or double-tap the surface, the bounding box surrounding one person's multiple touches, and a set of vectors reporting the signal strength of a person's touches.

To illustrate, Figure 4 shows SquiggleDraw, a paint program written in approximately 10 minutes in about 15 lines of code. SquiggleDraw has two interesting properties.

- A person adjusts line thickness on the fly. One draws by changing the bounding region of the drawing with two fingers. One draws thin lines by holding their thumb and forefinger close together, and progressively thicker lines by spreading their fingers apart.
- Up to four people can draw simultaneously, with each person's lines appearing in a different color.

Because of the availability of both the SDGToolkit and the DiamondTouch Toolkit, many other students in our laboratory are now working on single display groupware. Some are concentrating on quite serious research projects, and are rapidly

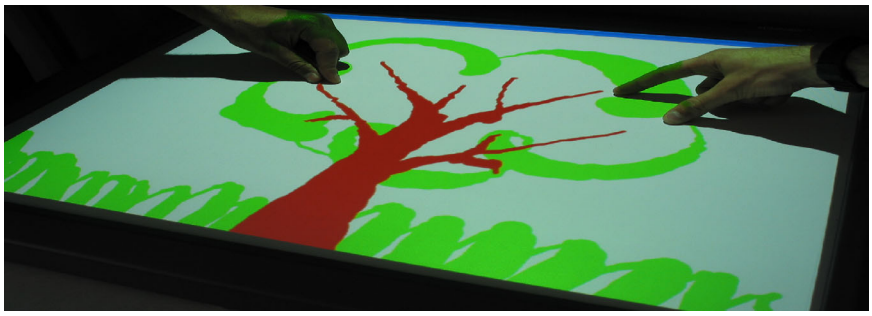

Fig. 4. Two people using SquiggleDraw.

implementing ideas just to test them out. Others are 'dabbling' for their own curiosity, but are producing fairly interesting systems. For example, student Tony Tang combined both the SDGToolkit and the Collabrary to create a tabletop application that handles both co-located and distance-separated participants.

Toolkits for Physical User Interfaces

In the last few years, researchers have developed groupware designs that include physical user interfaces augmented by computing power. These typically involve ambient displays for showing awareness information, or collaborative physical devices that are controlled by multiple (perhaps distributed) people. While this is an exciting new area, everyday programmers face considerable hurdles if they wish to create even simple physical user interfaces. Most lack the necessary hardware training. Those willing to learn find themselves spending most of their time building and debugging circuit boards, firmware and low-level wire protocols rather than on their physical user interface designs. The problem is that we have not provided programmers with adequate building blocks for rapidly prototyping physical user interfaces. This leaves them in a position similar to early GUI researchers who had to build their widgets from scratch, or to early graphics researchers who had to build their 3D environments by brute force. Given this onerous situation, it is no wonder that most research on physical user interfaces come from top researchers at major university and industrial research laboratories.

As with our other areas, our solution was to develop a toolkit for rapid development of physical widgets, or *phidgets* [8]. Our approach was to provide programmers with pre-packaged hardware devices that can be 'dropped into' software applications. This familiar programming paradigm is directly analogous to how graphical user interface (GUI) widgets are programmed.

I gave phidgets to undergraduate students with no hardware expertise to see what they could do with them. These typically took the form of a short two week assignment. The results were remarkable. While some students replicated examples of physical user interfaces reported by other researchers, most produced their own innovative designs [8].

Fig. 4. a) Messenger Frame, **b)** MC Status, **c)** Appointment Assistant, and **d)** FoosWars

A few example of groupware systems they built are illustrated in Figure 4. The first two are physical notification devices attached to MSN Instant Messenger. In Messenger Frame by Mike Hornby-Smith (4a), a contact's photo is lit up and a sound cue generated as that contact appears online or changes their activity status. One sends a message directly to that contact by touching his photo. In MC Status by Christian Leith (4b), contacts are represented by figurines. Offline figurines face the wall, and online figurines face forward. Touching the area in front of the figurine initiates a message. Appointment Assistant by Zaid Alibhai (4c) is an ambient appointment reminder display that interacts with a user's on-line calendar to remind them of upcoming appointments. As an appointment approaches, the figure on the top of the display moves along the scale and LEDs light up to further indicate the time remaining before the next appointment. FoosWars by Mike Larke and Mike Clark (4d) is a soccer table for distributed participants. One person plays on the physical table while the other plays over the web. The remote player has a live aerial view of the table captured via a web camera located above the table, and directly manipulates his or her players through use of physical sliders. Other example phidget projects are found at www.cpsc.ucalgary.ca/grouplab/phidgets/gallery.

Closing thoughts.

Groupware development parallels Gaines' BRETAM phenomenological model of developments in science technology [7]. The model states that technology-oriented research usually begins with an insightful and creative breakthrough, followed by many (often painful) replications and variations of the idea. Empiricism then occurs when people draw lessons from their experiences and formalize them as useful generalizations. This continues to theory, automation and maturity.

Within this context, we can now see how groupware toolkits affords creative research. Toolkits make it easier for researchers to create new breakthroughs through

rapid prototyping of many new ideas. They let others replicate and evolve ideas reported in the literature. They also let researchers more easily move into empiricism by making it easy to create different versions of testable systems.

References

1. Bederson, B. & Hourcade, J.: Architecture and implementation of a Java package for Multiple Input Devices (MID). HCIL Technical Report No. 9908. (1999)
2. Bier, B. & Freeman, S.: MMM: A user interface architecture for shared editors on a single screen. Proc ACM UIST (1991) 79-86.
3. Boyle, M. and Greenberg, S.: GroupLab Collabrary: A Toolkit for Multimedia Groupware. In J. Patterson (Ed.) ACM CSCW 2002 Workshop on Network Services for Groupware, November (2002)
4. Diaz-Marino, R.A., Tse, E, and Greenberg, S.: Programming for Multiple Touches and Multiple Users: A Toolkit for the DiamondTouch Hardware. Department of Computer Science, University of Calgary, Calgary, Alberta CANADA. June. Includes video (2003)
5. Engelbart, D. and English, W. Research Center for Augmenting Human Intellect, Proc. Fall Joint Computing Conference, AFIPS Press (1968) 395-410
6. Foster, G.: Collaborative Systems and Multi-user Interfaces. PhD Thesis (UCB/CSD 87/326), Computer Science Division (EECS), Univ of California at Berkeley USA (1986)
7. Gaines, B.: Modeling and forecasting the information sciences. Information Sciences 57/58, (1999) 13-22
8. Greenberg, S. and Fitchett, C.: Phidgets: Easy Development of Physical Interfaces through Physical Widgets. Proc ACM UIST (2001) 209-218
9. Greenberg, S., Gutwin, C. and Cockburn, A.: Using Distortion-Oriented Displays to Support Workspace Awareness. In A. Sasse, R. Cunningham, R. Winder (Eds), People and Computers XI (Proc HCI'96) Springer-Verlag (1996) 299-314
10. Greenberg, S. and Roseman, M.: Groupware Toolkits for Synchronous Work. In M. Beaudouin-Lafon, editor, Computer-Supported Cooperative Work (Trends in Software 7), John Wiley & Sons Ltd (1999) 135-168
11. Greenberg, S., Roseman, M., Webster, D. and Bohnet, R.: Human and technical factors of distributed group drawing tools. Interacting with Computers, 4(1) (1992) 364-392
12. Greenberg, S. and Rounding, M.: The Notification Collage: Posting Information to Public and Personal Displays. Proc ACM CHI (2001) 515-521
13. Gutwin, C., Greenberg, S. and Roseman, M.: Workspace Awareness in Real-Time Distributed Groupware: Framework, Widgets and Evaluation. In A. Sasse, R. Cunningham, R. Winder (Eds), People and Computers XI (Proc HCI'96) Springer-Verlag (1996) 281-298
14. Myers, B.: State of the Art in User Interface Software Tools. In R. Baecker, J. Grudin, W. Buxton and S. Greenberg (Eds) Readings in Human Computer Interaction: Towards the Year 2000. Morgan Kaufmann (1995) 323-343
15. Sarin, S. Interactive On-Line Conferences, PhD thesis, MIT/LCS/TR330, MIT USA (1984)
16. Stewart, J., Bederson, B. and Druin, A.: Single display groupware: a Model for Co-present Collaboration, Proc ACM CHI (1999) 286-293
17. Tang, J.C.: Findings from observational studies of collaborative work. International Journal of Man Machine Studies, 34(2), Academic Press (1991) 143-160.
18. Tse, E. and Greenberg, S.: Rapidly Prototyping Single Display Groupware through the SDGToolkit. Report 2003-721-24 Computer Science, University of Calgary, Canada (2003)
19. Zanella, A. and Greenberg, S.: (2001) Reducing Interference in Single Display Groupware through Transparency. Proc ECSCW, Kluwer (2001).