Coordinating Networked Learning Activities with a General-Purpose Interface

John Brecht
SRI International
Charles Patton
SRI International
S. Raj Chaudhury
Christopher Newport University
Chris DiGiano
SRI International
Deborah Tatar
Virginia Tech
Jeremy Roschelle
SRI International
Krista Davis
SRI International

Abstract

Classrooms equipped with wirelessly networked tablets and handhelds can engage students in powerful collaborative learning activities that are otherwise impractical or impossible. However, the system must fulfill certain technological and pedagogical requirements such as tolerance for latecomers, supporting disconnected mode gracefully, robustness across dropped connections, promotion of both positive interdependence and individual accountability, and accommodation of differential rates of task completion. Two approaches to making a Tuple Space-based computer architecture for connectivity into an inviting environment for the generation and creation of novel coordinated activities were attempted. One approach made the technological "bones" of the system very clear but assumed user vision of the complex goals and settings of real education. The more satisfactory approach made clear how Tuple Spaces matches the complex goals and settings of real education, but backgrounded technical complexity. This approach provides users with a system, Group Scribbles, which may inspire a wide range of uses.

1. Introduction

Classrooms equipped with wirelessly networked tablets and handhelds can engage students in powerful collaborative learning activities that are otherwise impractical or impossible. . A networked classroom can support real-time formative assessment for teachers (Abrahamson, 1999) as well as activities such as interactive role playing, joint concept mapping and group critiquing. To date, research and development in mobile learning has focused mostly on single-purpose tools in support of particular activities. A more desirable solution would be a general-purpose system that can play a range of roles, so that teachers and students need only learn and invest in one primary kind of classroom connectivity.

The purpose of our research was to identify characteristics of computational platforms that not only enable the implementation of coordinated networked learning activities but inspire the design of activities with high pedagogical value. This goal led us to a program of research where we identified basic enabling requirements for platforms, created competing platforms that satisfied these requirements, and then evaluated the pedagogical value of the activities that emerged from these platforms.

1.1 Requirements

On the basis of a review of the CSCL literature, we identified the following technical requirements for a platform for implementing collaborative learning activities:

- It must have *latecomer tolerance,* so that devices that join a session after it has started can fully and gracefully catch up without having lost data.
- It must be *robust across dropped connections*, which will occur frequently (although the main concern is not guaranteed transmission with a return receipt, but rather smooth participation in the flow of an activity).

- It must also *support disconnected mode gracefully*, allowing students to work offline and later submit their work in bulk (while catching up as well).
- The coordination layer must have a simple *discovery paradigm* for figuring out what kinds of sessions are running and which ones a user might join.

We created two platforms that satisfied these basic requirements: an "API" platform based on the tuple spaces architecture, and a "GUI" platform, which provided a general-purpose GUI (Graphical User Interface) for activity design.

1.2 Evaluation Criteria

CSCL literature also led us to a list of high-value pedagogical qualities for comparing collaborative learning activities resulting from our platforms:

- *Positive interdependence and individual accountability.* Every student should be individually accountable for some portion of the task, and the overall goal requires all students' contributions (Johnson & Johnson, 1989).
- *Role specialization.* Students should be encouraged to focus deeply on one dimension of teamwork at a time (Kagan, 1992).
- *Even-odd tolerance.* In realistic classroom settings, applications also must address the possibility that "extra" students will be assigned to a group.
- *Support for differential rates of completion.* Some students work faster than others and can be disruptive if they have nothing to do but wait.

A common theme in all the above qualities is the notion of *distributed control* in collaborative learning activities. Interdependence and accountability suggest that students ought

to be making decisions that affect the progression of the activity. Role specialization emphasizes a distribution of responsibilities. Even-odd tolerance is often characterized by the ability of participants themselves to adjust to nonideal student counts, without centralized intervention, say, by the instructor. Finally, support for differential rates is often predicated on the ability of individuals to set the pace of their involvement in an activity.

In particular, we sought to measure the degree to which each platform *inspired* the above qualities. This analysis is distinct from the question of whether a platform simply *enabled* a programmer to create an activity with such qualities.

2. Test Cases

To evaluate the pedagogical value of artifacts derived from our two platforms, we needed test cases in the form of classes of activities to implement. Prior work (DiGiano, Yarnall, Patton, Roschelle, Tatar, et al., 2003) has promoted the value of identifying "whole activity" patterns in collaborative learning activities and using these patterns as objects to think with during design. We started by identifying 41 candidate whole activity patterns to serve as test cases. We then ranked the candidates based on criteria such as importance to science and mathematics education (the content focus of our efforts) and popularity among instructors and the CSCL research community. The top four activity patterns, described below, became the focus of our testing.

1. *Question Posing*. Students propose questions for consideration relative to some topic, and all students are involved in reviewing, ranking, and clustering candidate questions. The result is a ranked set of important questions, with related items gathered in clusters.
2. *Multiple Representations*. A collection of artifacts representing learning phenomena is divided up among students. Each student (or small group of students) is charged with creating one of the preset representation types for the selected artifact.

3. *Simultaneous Annotation*. Students mark up the same content at the same time and then compare and contrast results.
4. *Image Mapping.* The instructor poses an inquiry. Students respond by placing tokens on an image. The instructor and students can then view and discuss the aggregation of tokens.

3. The Tuples API Platform

A common strategy for supporting the implementation of educational software is to add a pedagogically oriented layer to the generic application programming interface (API). This layer can add utilities for managing a collection of activities, persistence of student data, and graphical interface components specifically designed to support student inquiry. The layer can also be useful for ensuring a consistent user experience across a family of related products. We call this approach an "API" platform because educational activities are built on top of this layer by programmers who have learned to leverage the added layer to rapidly construct high-quality artifacts.

To evaluate how best to support the implementation of collaborative learning activities, one of our two platforms took this familiar API approach. As shown in Figure 1, the lowest layer of this platform is the Java programming language. On top of this is a "tuple space" library from IBM for coordinating distributed processes and the Eclipse Project's SWT GUI library (http://eclipse.org). On top of that we wrote abstract classes to simplify the most common tuple space operations and an activity management API, which provides, among other things, a simple interface for participants to select from a list of available activities. Activities implemented on this platform involve a server machine communicating with a network of Java-capable laptop or desktop student devices.

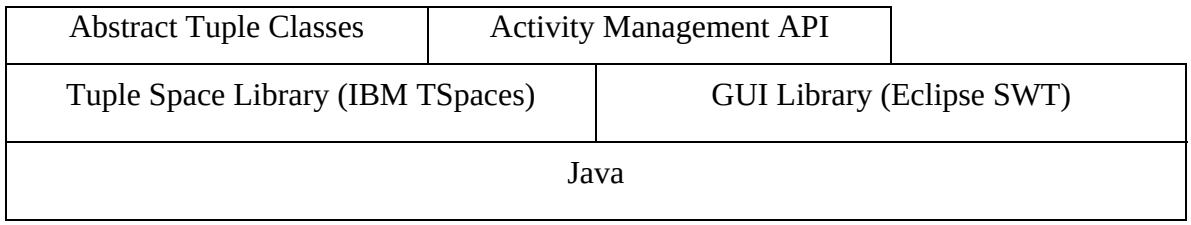

Figure 1. Layers of the API platform for implementing collaborative learning activities.

To appreciate the implementation issues that confront a programmer using this API platform to make a collaborative learning activity, it is important to understand the tuple spaces architecture. Tuple spaces were the conceptual base for the programming language Linda, a language for coordinating parallel processes. The tuple spaces architecture is an example of the "Blackboard Model" of computing, in which there is one central, public data store: the Blackboard. Processes in the system observe this data store independently, watching for data that they can act on, performing actions, and updating the Blackboard accordingly. Coordination in such a system is not determined by a central mediator dictating when the various processes should act; rather, the processes themselves make the determination, and the coordination is emergent. This highly asynchronous model satisfied our technical requirements for robustness across dropped connections and graceful support of disconnected mode. Latecomer tolerance may also be realized insofar as the Blackboard serves as a cache of activity.

The tuple spaces architecture extends the Blackboard metaphor with a specific data structure, the *tuple*; a shared memory structure, *spaces*; and an associated set of operations. *Tuples* are ordered sets of *fields*. *Fields* are data elements that have a value and, depending on implementation, may also have types and/or names. *Spaces* are simply "bags"—sets that can contain duplicate elements—that contain tuples. Three core operations can be performed on a space: *write*, *read*, and *take*. *Write* puts a tuple into a space, *read* reads the value of a tuple in a

space, and *take* removes a tuple from a space. Rather than using a query language to find tuples for read and write operations, the tuple spaces architecture relies on pattern matching for retrieval. A "template" tuple is provided as an argument in read and write operations, and tuples matching that template are returned. This highly flexible data structure and simple query mechanism are relatively easy to learn, compared with alternatives such as database tables and the Structured Query Language.

3.1 Experience with the API Platform

Project team members were able to create instantiations of all four of our test case whole activity patterns in-house:

- *Question Posing*. This involved writing a custom GUI for capturing questions, writing the questions to a tuple space, and ranking them. A specialization of the platform's abstract tuple class was created to store questions. We used tuple space read operations to collect all questions on a particular topic. This occupied a programmer's time for several weeks.
- *Multiple Representations*. The existing ChemSense drawing tool from SRI (http://chemsense.org) was repurposed with minimal effort to read and write diagrams from and to a specialization of the abstract tuple class. We also needed to write a specialized GUI widget for writing and taking names of molecules in a tuple space. Identifying which molecules had what representations required invoking the read operation on the space. Two programmers worked together over several weeks.
- *Simultaneous Annotation*. Students mark relevant lines in a document and share their marks. This involved creating a custom document viewer with a check box beside each line of text. A student's particular subset of selected lines were stored in a specialization of the abstract tuple class. Aggregating student selections required read operations on the

appropriate tuple space. Additional GUI code for browsing and viewing the aggregated selections was created. A bar graph to the right of each line indicates the number of students who highlighted the particular line, and the font size of more popular lines is also increased to indicate frequency of selection. One programmer worked over several weeks.

- *Image Mapping.* We created a custom GUI that could display an arbitrary image file and interpret clicks as token placements. We needed to create a specialization of the abstract tuple class to capture a student's token positions. Token placement involved simply writing this tuple to a particular tuple space. Aggregating tokens involved a tuple space read operation that matched on a particular question posed by the instructor. The GUI required further work to display aggregated tokens with or without attribution. One programmer worked over several weeks.

At least two high-value pedagogical qualities appear in three of these four activities (Table 1). Most support positive interdependence and individual accountability; role specialization was not explored.

Table 1.

Evaluation of Activities Generated by In-House Programmers Using the API and GUI Platforms

Activity pattern	Positive interdependence and individual accountability	Role specialization	Even-odd tolerance	Support for differential rates of completion
Question Posing	API & GUI: Students		API & GUI: Not	API & GUI: Faster

	accountable for creating their own questions; ranking depends on peer contributions		affected by student count	students can always add more questions while waiting for slower ones
Multiple Representations	API: Students accountable for selecting molecules GUI: Students accountable for selecting equations API & GUI: Task completion depends on aggregation of everyone's representations	API & GUI: Certain students can be responsible for particular representation types	API & GUI: If student count is not evenly divisible by representation count, it's OK if a few do an extra round	API & GUI: Faster students can render more representations while waiting for slower ones
Simultaneous Annotation	API: Students contribute to an annotation tapestry GUI: Students contribute to an overall class rating for items (based on number of stars)		API & GUI: Not affected by student count	API: Faster students can do a closer review in a follow-up pass while waiting for slower ones
Image Mapping	API: Students		API & GUI: Not	API & GUI: Faster

	contribute to an emergent pattern of token placements GUI: Students contribute to an emergent class consensus on *Homo sapiens* origin		affected by student count	students likely to influence token placement by slower students

In addition to implementing activities ourselves, we invited college-level student programmers to create collaborative learning activities using the programmer-centric platform. This was done in the context of a 3-week summer workshop at SRI and in one design course during the school year at Virginia Tech. Through direct feedback from students and comments from instructors, we learned that in the first year of instruction these students had considerable difficulty implementing new activities. Some of these difficulties were technical; however, more profoundly, the students had difficulty envisioning the desired activity and the relationships among the activity, the pedagogy, and the design of the system. By arriving at the metaphor of playground games for the activities, the professor was able to convey the relevant values to the students, who in short order successfully built seven working prototypes. As important as that development work was, it became clear that this approach to programming would not lead to spontaneous, self-motivated, untrammeled development (Lin et al., 2006).

3.2 Reflections on the API Platform

We found that applications, such as ChemSense, that were already implemented on other architectures with limited collaborative functionality could be quickly reimplemented by in-

house programmers and “collaboratized” in the tuple spaces framework. For these programmers, our platform provided a powerful and comparatively simple path to adding collaborative functionality to existing, well-understood, applications.

Results suggest that students needed more than simply an improved API. In particular, success was achieved only after significant in-class discussion, debate, and coaching around the nature, possibility, and desirability of highly distributed control.

4. The Group Scribbles GUI Platform

In contrast to the first platform, which was intended for programmers as implementers of collaborative activities, our second platform, “Group Scribbles GUI,” is intended to be used by instructors or non-technical curriculum developers. As illustrated in Figure 2, the GUI platform shares the same foundation of Java and tuples as the API platform. However, the GUI platform adds the Group Scribbles layer, a general-purpose graphical interface for the implementation and execution of collaborative learning activities. It also makes the the lower-level SWT GUI library is inaccessible to activity implementers. Group Scribbles-based activities involve a server machine communicating with a network of student devices, ideally Tablet PCs with a stylus interface (although we also support Pocket PC handhelds and mouse-based laptops or desktops).

Group Scribbles offers implementers, instructors and students what we believe is a powerful metaphor for thinking about and realizing collaborative learning activities. This metaphor is based on common physical artifacts from the classroom or office: adhesive notes, bulletin boards, whiteboards, stickers, pens, and markers. The fundamental unit of expression in Group Scribbles is the Scribble Sheet, a small square of virtual paper just large enough to express a single thought or concept, whether via a quick sketch or a few words jotted down. Scribble

Sheets can be posted to Public Boards, where many sheets can be arranged to express ensemble ideas, such as groupings, chronologies, or hierarchies.

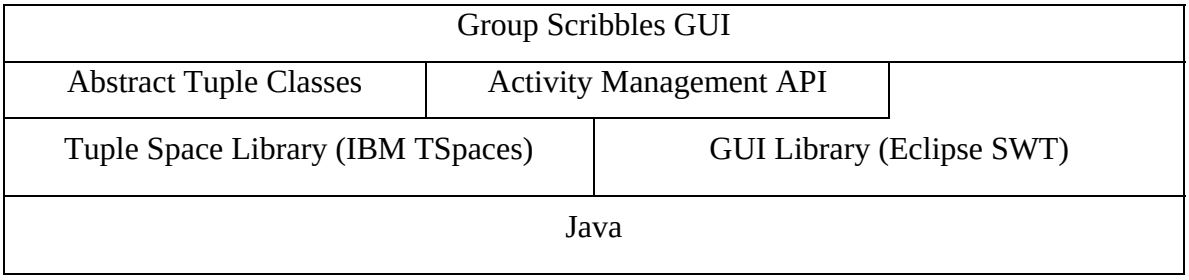

Figure 2. Layers of the GUI platform for implementing collaborative learning activities.

To understand how one implements a collaborative learning activity in Group Scribbles, it is important to understand what it is like to participate. As shown in Figure 3, each participant has a Private Board on which to create and arrange Scribble Sheets. A given classroom instance of Group Scribbles will have one or more named Public Boards accessible to all users. A typical client view is subdivided to show the user's Private Board in one region and a Public Board in another.

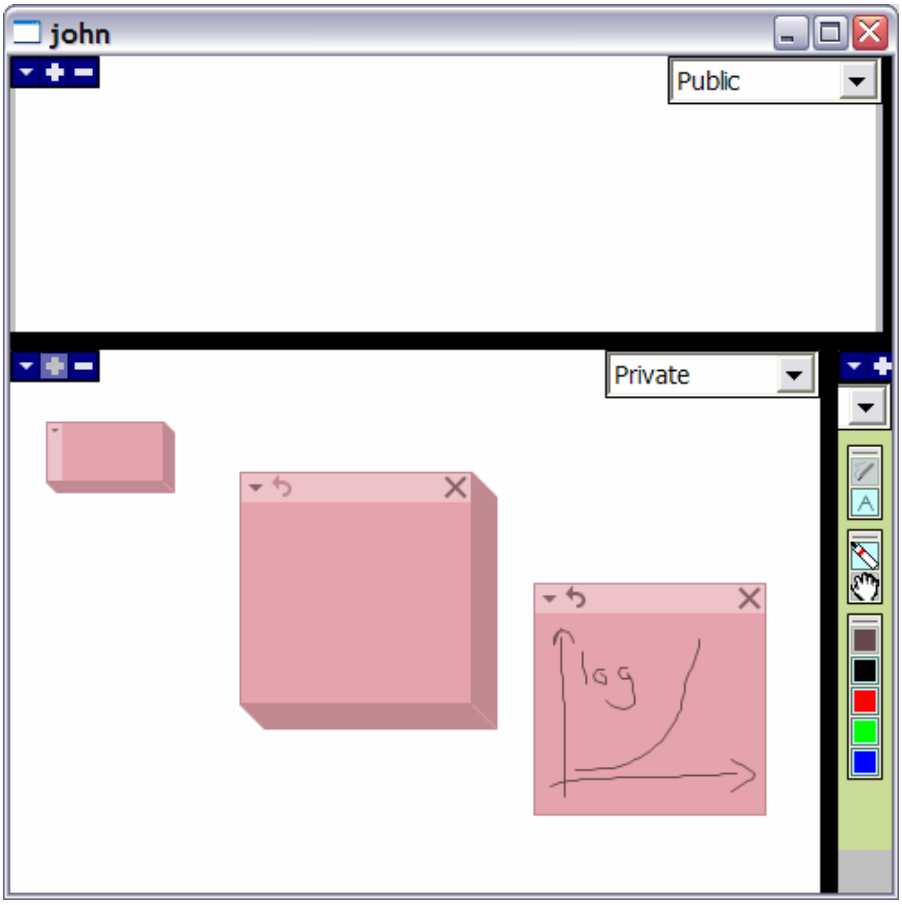

Figure 3. The Group Scribbles GUI with a Private Board below and a Public Board above.

On the Private Board, a user finds a Scribble Pad, an endless source of fresh Scribble Sheets. Users can pull sheets off the pad and write (or type) on them to generate new content. Figure 3 depicts a fully zoomed-in view of the Private Board. Users can zoom out several levels to help arrange and maintain their Scribbles. Also available on the Private Board are Scribble Labels—essentially smaller Scribble Sheets useful for annotation. When users are ready to publish a Scribble Sheet, they simply drag it onto the Public Board. The new sheet then appears on the Public Board of all participants; the frequency of screen updates depends on user configuration.

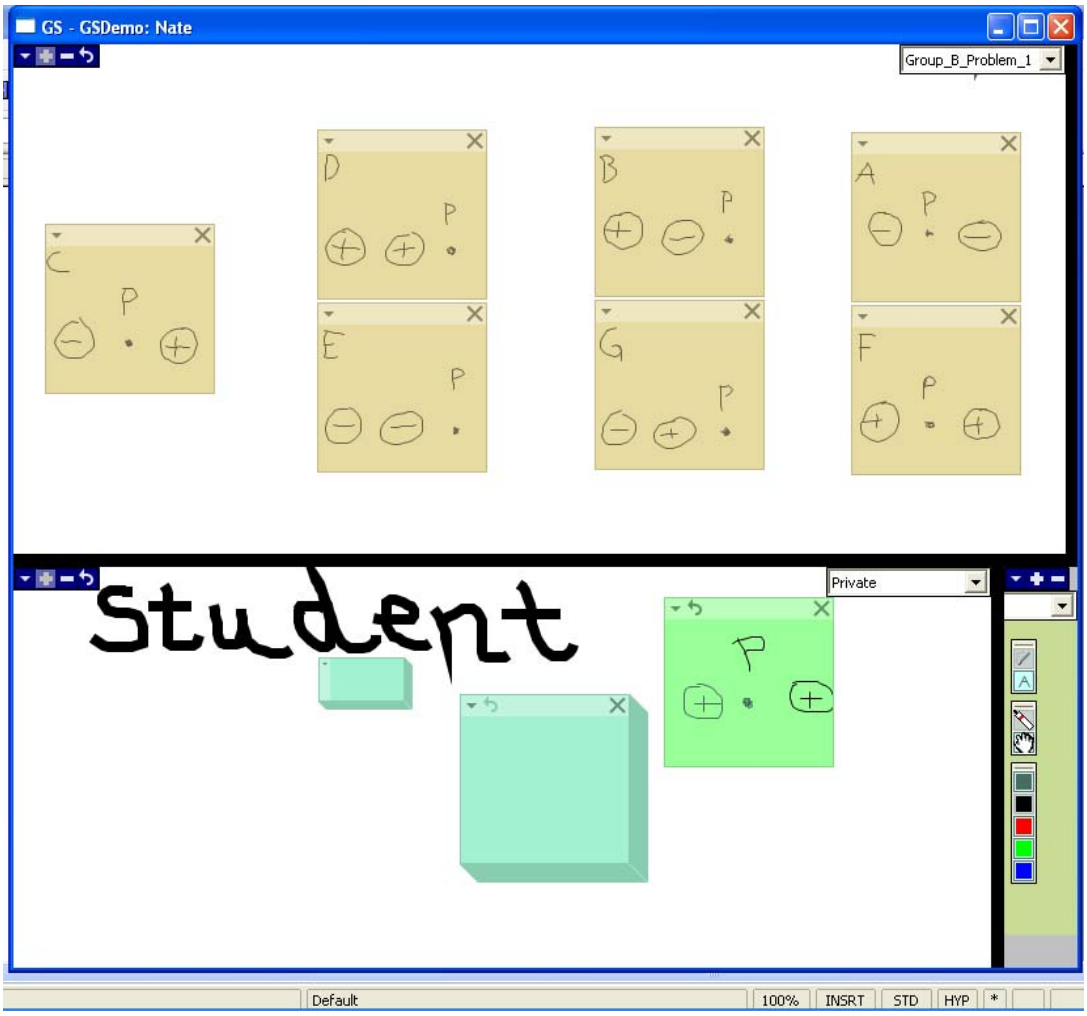

Figure 4. Ranking Task activity with an introductory physics class.

On a Public Board, any user is free to reposition any Scribble Sheet so that, while individual thoughts are expressed within individual sheets, collective ideas are expressed over entire boards. In this way, Scribbles can be sorted, grouped, or otherwise arranged to express interdependent meaning. Scribble Sheets can also be taken from the Public Board (and later returned) and brought onto a user's Private Board (e.g., for activities calling for exchange or to take a token representing a turn in a sequence).

The object of the activity depicted in Figure 4 (Chaudhury et. al., 2002; O'Kuma et. al., 2000) was for students to determine the strength of the electrostatic force at the point P due to seven different arrangements of charges. Students had to solve each scenario, recognize equivalence of certain pairs of scenarios and arrange the sheets from left to right based on the strength of the net force.

Because we built the GUI platform on top of the API platform, it shares all the same technical qualifications of the latter: robustness across dropped connections and graceful support of disconnected mode. Exactly how we implemented Group Scribbles on the API platform is beyond scope of this paper, but it is worth noting that there is a close correspondence between Group Scribbles GUI objects and tuples. For example, when a Scribble Sheet is dropped onto the Public Board, a tuple describing the sheet is written to the tuple spaces server. Other clients with the same Public Board in view will receive notification of the newly written Scribble Sheet tuple. When they read this tuple, the new sheet will be rendered in their view of the Public Board.

The process of implementing a new Group Scribbles activity is not much different from the end-user experience of participating in an activity. It involves placing sheets with seed content on a Public Board and perhaps creating a background image that can contextualize the activity. The board configuration is automatically saved indefinitely under an activity name on the tuple server, so that at any time (a day, a week, or a year later) the instructor can ask students to open the "pre-baked" Group Scribbles GUI through the system's activity manager. The idea that the instructor asks students to perform such a task on their Group Scribbles client is characteristic of activities built on the GUI platform: students are often responsible for pulling content in a distributed fashion because there are no built-in push mechanisms in Group Scribbles (beyond

Public Board sharing). We will show below how this reliance on "social mediation" actually has pedagogical value.

4.1 Experience with the GUI Platform

As detailed below, we were able to create instantiations of all four of our test case whole activity patterns in-house:

- *Question Posing*. The instructor relies on social mediation to get students to scribble a question on a sheet and submit it to the Public Board for review. Students rank questions by jointly arranging sheets on the Public Board, say, from left to right. By default, Group Scribbles with its physical metaphor prevents more than one student from moving a sheet simultaneously. No "pre-baked" content is necessary but an optional background image for the Public Board can suggest bins for organizing questions into groups.
- *Multiple Representations*. We found that a matrix background image (Figure 5) is all that is needed to organize a simple activity in which students render multiple representations of a set of functions, such as an equation, a graph, and a tabular depiction of each. To start the activity, the instructor populates the matrix with a set of tokens. Any student can take a token and replace it with a rendering corresponding to that cell—e.g., a graph of a periodic function.
- *Simultaneous Annotation*. We currently have limited experience with this whole activity pattern. Given the size restrictions of sheets, the classic example of text markup is not easily implemented. However, we have successfully conducted activities in which students use Sheet Labels to attach star ratings to content on the Public Board.
- *Image Mapping.* Again, we found that a background image is all that is needed to implement a simple mapping activity. For example, the instructor loads an image of the

globe onto the background of the Public Board and then uses social mediation to get students to place a Sheet Label on the region associated with the emergence of *Homo sapiens* around 200,000 B.C.

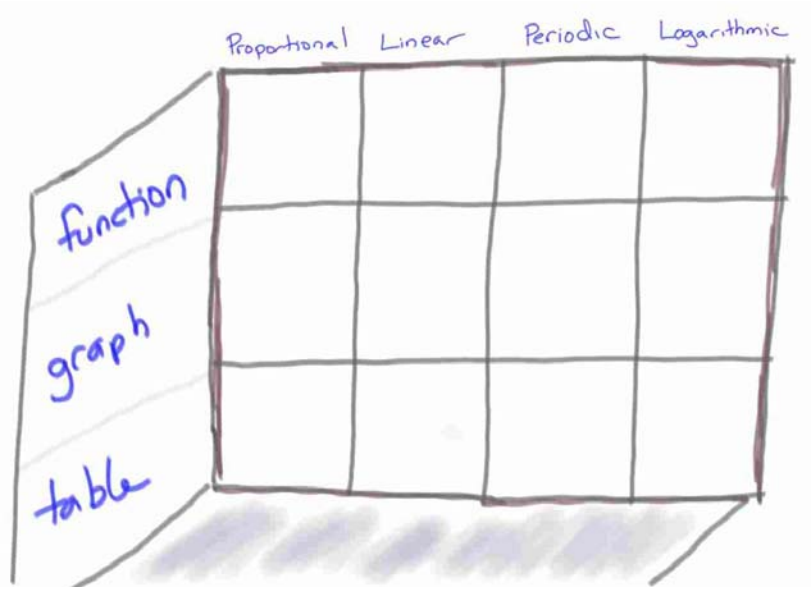

Figure 5. Matrix background image.

To gain more objective feedback on the GUI platform, we tested Group Scribbles in an informal higher education setting—with undergraduate research assistants enrolled in a NASA summer program. Activities were designed for short, 10- to 15-minute interactions, primarily to test the various hardware platforms being investigated for wider classroom use. Question Posing and Image Mapping activities were implemented with a view to whole-class aggregation and discussion. Although no formal assessment of learning based on these sessions was conducted, the students were able to learn to use the basic features of Group Scribbles within a few minutes and successfully participate in instructor-led activities. An emergent feature of the use of the system was back-channel “chats” via Scribble Sheets: even though sheets were being placed on

the Public Board for everyone to see, the content was clearly targeted at specific individuals. In most cases, these individuals had existing social relationships, similar to classic note-passing in secondary classrooms. Formal classroom testing with students enrolled in credit-bearing courses is currently being conducted at a partner university. Preliminary feedback reinforces the view that GS is an easy to learn system and Figure 4. is a direct outcome of a successful learning activity conducted by one of the authors in an introductory physics class for non-science majors.

4.2 Reflections on the GUI Platform

At least two of the desired pedagogical qualities appeared in two of the four activities. As with the API platform, the most prevalent quality was "positive interdependence and individual accountability"; the least prevalent was role specialization.

However, none of the experiences in implementing collaborative learning activities with the GUI platform required programming. With no programmatic control over the design of activity, one might expect that there would be limited opportunity to ensure interactions with high pedagogical value. Yet, our evaluation of the activities showed the presence of almost all the desirable qualities of the much more customized activities built on the API platform. We attribute this success to design features of the Group Scribbles GUI that naturally affords high-quality collaborative activities:

- *(Re)arrangable*. Scribble Sheets can be positioned and repositioned to convey meaning. This feature allows students to jointly negotiate the relationship between artifacts, such as the ranking of questions in the Question Posing activity by simple drag operations. The result is an environment where any student's response can be subject to challenge, thus emphasizing interdependence.

- *Unique*. As simulated physical artifacts, Scribble Sheets cannot be in more than one place at a time. This attribute naturally supports activities where certain artifacts are manipulated sequentially, such as a particular equation in the function representations activity that gets rendered in one form and then another. Ultimately, this feature simplifies the coordination of artifacts when students are playing specialized roles or trying to avoid duplicate work.
- *Metainformatic*. As described in the simultaneous annotation and image mapping activities, Scribble Sheets and Labels can mark up images or other Scribble Sheets. This feature also contributes to a sense of positive interdependence.
- *Modeless*. Unlike with products of the API platform, our experience is that Group Scribbles (perhaps because of its lack of programmability!) encourages activities in which students are free to choose from a wide variety of operations rather than being constrained to a particular subactivity. The ability for students to be working simultaneously on different aspects of an activity or for one student to use surplus time to revisit an aspect helps support differential rates of completion and even-odd tolerance.

With our GUI platform, the activity implementer sacrifices the possibility of a user experience that is highly tailored to the task at hand. For example, it is not currently feasible to add scaffolding for students to generate valid ball and stick diagrams in chemistry. On the other hand, we have been pleasantly surprised by the wide variety of activities we can support with reasonable fidelity, which we attribute to two other Group Scribbles qualities:

- *Representationally neutral*. Students can use digital ink on Scribble Sheets to easily capture diagrams, drawings, mathematical and scientific notation, and text. Different sizes afford different kinds of activities.

- *Background imagery*. Our experience is that a simple background image, such as of the globe or a Cartesian coordinate system, can provide critical pedagogical context without requiring programming.

Finally, it is worth noting that instructors and students may actually benefit from the fact that Group Scribbles-based activities have a common—albeit somewhat generic—look and feel. Anecdotally, we hear complaints from instructors who must juggle (and subject their students to juggling) many different classroom tools. The flexibility of the Group Scribbles GUI for collaborative learning activities challenges the need for such disparate applications and hints at a future when instructors and students may need to invest in only a small number of core tools that they can grow with over time.

5. Conclusions

Apple's introduction of HyperCard in the 1980s ushered in a wave of inventive and useful educational applications, all created by teachers, students, and other nonprogrammers. Until now, technology-enhanced classroom collaboration and coordination activities have been, like educational applications in pre-HyperCard days, the sole domain of dedicated programming teams, greatly limiting the scope of experimentation and creative effort. Extending the analogy, it may be useful to consider what could be learned from the HyperCard experience that might be applicable to the programmer cognition issue around distributed control: were similar issues neatly sidestepped by HyperCard through appealing directly to end users as programmers? We argue that HyperCard not only provided programming tools to teachers but sidestepped a programmer cognition issue relevant at the time: the conceptual shift to user-event-driven programming.

Though the graphical user interface, and, indeed, hypertext were introduced by SRI researchers in 1968 (Engelbart, 1968), until the time of the introduction of HyperCard, the dominant conception of programming was highly shaped by the terminal interface (DOS command line, telnet, etc.) and an interaction style dominated by computer control in both outputs and inputs ("input statements" are a dead giveaway of this mind-set). In that style, the computer program determines what information is demanded (word used advisedly) of the user and when input is allowed. Programs were normally structured as a branching tree of requests for specific inputs and displays of resulting outputs. The very notion seemed ridiculous that one could create novel and useful applications with only

- buttons to push;
- text fields to type into or click on;
- screens ("cards") containing buttons, graphics, and text fields; and
- the capacity to set up automatic "links" from one card to another (Neuburg, 1994).

But the aspect uniting these elements, and perhaps the most significant payload, was a programming paradigm that situated control primarily in the hands of the user.

Though it is clearly too early to predict with confidence, there is reason to believe that Group Scribbles, with its small set of generic features and emphasis on graphical (and hence machine uninterpretable) content, all united by a distributed control paradigm, could pave the way for a broader understanding of the benefits and challenges of programming in this model.

References

Abrahamson, A. L. (1999). *Teaching with classroom communication system: What it involves and why it works.* Retrieved February 20, 2004, from http://www.bedu.com/Publications/PueblaFinal2.html

DiGiano, C., Yarnall, L., Patton, C., Roschelle, J., Tatar, D., & Manley, M. (2003). Conceptual tools for planning for the wireless classroom. *Journal of Computer Assisted Learning, 19*(3), 284-297.

Chaudhury, S.R., Rodriguez, W.J., Cooper-Pabis, B.J., and Lee, K., "Using Ranking Tasks to Introduce Scientific Visualization in the Lecture", presentation at the AAPT National Meeting, Philadelphia, 2002

Engelbart, D., English, W; "A Research Center for Augmenting Human Intellect," AFIPS Conference Proceedings of the 1968 Fall Joint Computer Conference, San Francisco, CA, December 1968, Vol. 33, pp. 395-410 (AUGMENT,3954,). Johnson, D. W., & Johnson, R. T. (1989). *Cooperation and competition: Theory and research*. Edina, MN: Interaction Book Company.

Kagan, S. (1992). *Cooperative learning.* San Juan Capistrano, CA: Resources for Teachers.

Lin, S., Tatar, D., Harrison, S., Roschelle, J., & Patton, C. (2006, August). *Learning when less is more: "Bootstrapping" undergraduate programmers as coordination designers*. Presented in Exploratory Discussions at the Participatory Design Conference 2006, Trento, Italy.

Neuburg, M. (1994). HyperCard 2.2: The great becomes greater. *TidBITS,* 213. Retrieved from http://db.tidbits.com/article/04075

O'Kuma, T., Maloney, D. and Hieggelke, C., Ranking Task Exercises in Physics, Prentice Hall, 2000