Loops: Designing A Web-Based Environment for Persistent, Semi-Structured Conversation

Thomas Erickson, Christine Halverson, Wendy Kellogg, Mark Laff, Peter Malkin, Tracee Wolf
IBM T.J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598, USA
+1 914 784-7826
{snowfall|krys|wkellogg|mrl|malkin|tlwolf}@us.ibm.com

ABSTRACT
We describe the design of Loops, a second-generation CMC system aimed at small to medium-sized groups in a corporate environment. One goal of Loops was to preserve the lightweight conversation and awareness model developed in Babble. Creating this sort of environment on the web, a stateless and asynchronous medium, posed both design and implementation challenges. A second goal of Loops was to provide mechanisms for allowing users to impose structure on both static text and ongoing conversation. We discuss our approach—a server that uses TCP/IP-based XML communication to drive a client—and discuss the resulting system's architecture and interface.

Keywords
Design, CMC, Chat, Conversation, Structure, Semi-Structured Conversation, Social Proxies, Awareness

INTRODUCTION
Our chief goal is to design "socially translucent" systems — systems that convey social information and context by providing visual cues about the presence and activity of participants. We argue that such systems can, by taking advantage of the human ability to make inferences from traces of activity, provide an environment that supports a wide range of social processes (e.g. imitation; peer pressure) which permit groups to function effectively.

Up to this point, our work has been embodied in a first-generation system called "Babble." Babble is an online, conversation-centric system designed to support small to medium-size workgroups. In a series of publications we've described the design of the system [4], the "social translucence" rationale behind it [3], and studies of deployments and adoption of the system. In most of this work we have kept the focus on the socially translucent aspects of Babble, that is, the features which support participants' awareness of one another (and their awareness of that awareness).

This paper builds upon this previous research, but opens up two new areas of discussion. First, we describe the design and implementation of the second generation system, Loops. Although the goals of Loops are similar to Babble's, the implementation is radically different, involving a shift from client-server Smalltalk applications to a web-based system with a server driving a client written in Flash™ from Macromedia™[*]. Because of the stateless, asynchronous nature of the web, it as not easy to preserve the socially translucent aspects of Loops.

The second new area has to do with the issue of structure. In terms of design, one of the main ways in which Loops differs from Babble is that it incorporates interface elements intended to provide its users with lightweight means of structuring information. This emphasis on structure stems from observations of the very considerable efforts that users devoted to structuring information. Thus, as we discuss the rationale for the design of Loops, we will focus primarily on the evidence related to the creation and use of structure, since we have thoroughly discussed awareness and social translucence in other venues.

This paper begins by providing background on the design context in general, and the Babble system in particular. In the next section we lay out the factors that shaped the design of Loops; we devote particular attention to examining how users constructed structures within the weakly-structured Babble environment. In the third section we describe the architecture and user interface of Loops. Next we discuss work, still underway, that draws upon the Loops architecture and interface to provide scaffolds for semi-structured conversations. We close with reflections on the relationship between design and research.

BACKGROUND
Before turning to the factors which specifically drove the design of Loops, we'll say a bit about the context in which development occurred. Here we describe the position and role of the development group, the general situations for which we were designing, and the nature and use of the first generation system to which Loops was a response.

[*] Macromedia is a registered trademark, and Flash is a trademark of Macromedia Inc.

Design Research

In understanding the various forces that shaped the design of Loops, a good place to begin is with the context in which the design takes place. Our group is situated in ALC Research, and our charter is to carry out "adventurous research," that is, research that does not necessarily play into ALC's product development strategies.

Our work is best described by the rubric "design research." We conduct our research by embodying a variety of claims or conjectures in design prototypes, and then seeing how they play out that context. While some of this work may involve various forms of reflection and critique familiar to designers (see Schon [8] for a good description and rationale), or the genre of 'user studies' most familiar to HCI audiences, most of our work proceeds via the creation, implementation, and deployment of working systems to actual workgroups. This is a natural outcome of our interest in supporting *social* processes: group-based activity takes weeks to months to coalesce, and the only effective way we see of studying it is to observe it 'in the wild.'

This, in turn, means that even though we are engaged in "adventurous research," we need to construct systems which are sufficiently robust that they can be deployed to other workgroups. Ideally, as has been the case with Babble, the system will not only be robust, but will be sufficiently attractive to entice a number of groups into using the system. And this, in its turn, means that the ability to easily deploy and update the system is critical—something which became an important factor in the design of Loops.

The Babble System

Our first generation system, a Smalltalk-based client and server system, was called Babble. Here we provide a very brief description of it; interested readers should see [4] for more details of its design.

Babble was designed to serve the needs of small to medium sized corporate groups. It was intended to provide a semi-private online conversation area where members of groups such as workgroups, committees, and special purpose task forces could have text-based synchronous or asynchronous conversations. Among the assumptions embodied in Babble were that the groups would be relatively small, that most participants would know one another or *of* one another, and that, because participants were identified and situated in an organizational context, there would be considerable pressure to behave responsibly. Babble's approach to supporting conversation involves providing visual cues about the social context of the interaction (see [2] and [7] for examples of other systems).

Figure 1 shows a screenshot of the Babble user interface. From the upper left its basic components are: the list of users who are currently logged on; a visualization of the presence and degree of activity of users called a social proxy; a hierarchical topic list of user-definable conversations; and, in the lower half of the window, the text of the current conversation. There are three things to note. First, although Babble resembles a chat system, the conversation in Babble may be synchronous or asynchronous: that is, remarks may be separated by seconds or by minutes, days or months. Second, Babble provides a number of cues about who is present and what is new. The social proxy in the upper middle pane shows not only who is in the current conversation (the dots within the circle), but how recently the participants have 'spoken', i.e. typed, or 'listened', i.e. clicked or scrolled (via the proximity of the marbles to the circle's center). The topic list in the upper right pane has 'mini-proxies' next to a conversation name if anyone is in it (e.g. the third topic), and displays the topic name in red if it contains material the user hasn't seen. The third point to note is that the topic list is user definable: that is, users can create new conversation topics, rename them, and organize them hierarchically in nested categories — we will examine examples of such organization when we turn to the issue of structure.

Babble Deployments

Over the past four years we have deployed Babble to about two dozen groups. Most, though not all, deployments have been within ALC to groups such as official (i.e. organizationally defined) work groups, ad hoc task forces, and special interest groups. Most deployments have been intended to provide environments for long term collaboration, although some have been deployed to support shorter term online events (ranging from three days to a month or so).

Our success in deploying Babble has been mixed. Initially, we had limited success in getting groups to use Babble for lengthy periods of time. Although almost all groups initially used Babble enthusiastically, usage often fell off after about six weeks (see [1]). Over time, based on our observations of successful deployments, we developed a screening process, advice for moderators, and suggestions for new users, that appear to have increased the rate of successful adoption. In the last six months, the primary diffusion of new Babbles has been viral: that is, a member of an existing Babble requests a new deployment to support some other group in which he or she participates and takes charge of launching and moderating it; in these cases the presence of one or more experienced 'Babblers' seems very beneficial to adoption.

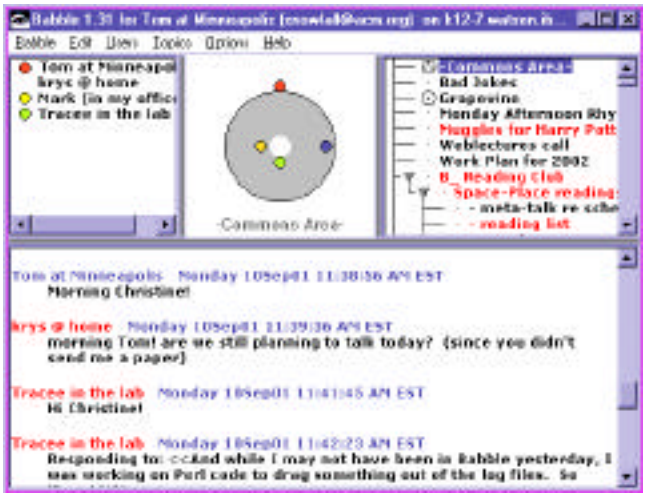

Figure 1. The Babble user interface.

FACTORS THAT SHAPED LOOPS

The design of Loops was shaped by three factors: deployment logistics; the desire to retain the most successful features of Babble; and the desire to support the creation structure within the online environment.

Deployment Logistics

As already noted, the ability to successfully create and deploy working systems to groups is critical to the success of our design research approach. This is particularly true if the system turns out to be popular, as has Babble.

Babble was not easy to deploy. The client, written in Smalltalk, was 2 to 3 megabytes in size. This meant that installation was non-trivial: users had to run an installation package that they usually downloaded from the network. The drawback is that this leaves the timing of the install up to individuals, and thus installation in most groups was staggered across several days. This made it more likely that those who installed Babble right away would log on and, finding no one to talk to, be uninclined to return.

Because Babble was a research prototype, it changed over time. This lead to the problem of trying to maintain compatibility across versions. Considerable efforts — in both programming and testing — were made to make sure that changes to the server or client did not break older versions of the client. However, incompatibilities did arise, sometimes by accident, and sometimes due to the desire to add new functionality that required fundamental changes. When this happened, all users need to reinstall more or less simultaneously, or lose access to the Babble environment when the server changed or incompatible clients came on line. The first case was difficult to manage for the reasons already mentioned; the second disruptive to group usage.

A third problem arose when trying to deploy to tightly administered environments. For example, one deployment of Babble was to a University class, which meant that Babble needed to be available in the public University computer labs. However, the lab administrators were wary of installing 'experimental' software on their lab machines, fearing that it might crash, corrupt disks, or have infelicitous interactions with other software. This particular problem was solved by creating a version of Babble that ran off a CD-ROM and kept its user-specific files on a diskette — however, this was not really a viable long term solution, and it exacerbated the update problem by requiring the burning of new CDs. While deploying to a public lab is a special case, note that similar concerns arise in most mission critical environments, such as help desks, sales support operations, etc., as well as any environment where hardware and software support is centrally controlled.

In considering the next generation design, each of these problems pushed us in the direction of developing a web-based client that could be run within a standard browser. Such a solution addresses the deployment problems, which are inherent in the design research approach we're pursuing, as well as making it easier to support multiple hardware and OS platforms, and people using multiple machines. However, we were rather reluctant to pursue this approach, because of the issues we discuss in the next section.

Retaining and Enhancing What Worked

Our experience with and studies of Babble left us convinced that it got quite a few things right. The principle features of Babble that we wished to retain were:

- The lightweight, blended synchrony, just-start-typing conversation model.
- The social proxy and other features that created an awareness of participants' presence and activity.
- The sense of history and inhabitation that resulted from the persistence of conversation and other signs of activity over time.

One of our concerns was that our emphasis on synchrony, awareness, presence, and inhabitation might prove difficult in an environment that was fundamentally asynchronous and lacked any concept of state. That is, whereas the Babble clients maintain a continuous connection to the server, web browsers do not: they connect with the server only to send or receive information. It was not immediately clear to us how to maintain the feeling of presence (i.e. that users are continuously connected to the environment), and we were concerned that we might end up trying to maintain an illusion with no infrastructural means of support.

In addition to these infrastructure concerns, we were also interested in creating an engaging user experience. We had two goals. First, we wanted freedom to use a range of subtle visual and auditory effects in designing successors to the Babble social proxies. Second, we wanted to allow our interaction designer to directly work in the medium, rather than having design prototypes reinterpreted by a programmer. As noted by Houde and Sellman [6], most development environments do a good job of supporting design or programming, but not both. Although we explored alternatives, these user experience goals eventually lead us to build the Loops client in Macromedia's Flash 5, an environment well-known for its ability to support the creation of interactive animations that can play in browsers.

Supporting Structure

As already described, we have considerable experience in deploying Babble. While we have discussed many aspects of our studies elsewhere, we have said little about how users have actually turned Babble to their own ends. In this section, we examine some of the ways in which Babble users have structured information within Babble to support their needs. We consider this structuring quite significant because Babble provides only rudimentary support for it.

To begin with, we need to say a bit more about how structure is created in Babble. Basically, Babble allows any user to create or rename the topics and categories shown in the hierarchical topics list (Figure 3, upper right). Topics are simply names for single conversations (analogous to documents in a GUI-sense), and categories are a means of grouping conversations (analogous to folders) which can contain topics or sub-categories. The only way to create structure in Babble is by creating named hierarchies of categories and topics.

For the purpose of this paper, we will look at data drawn from the five most active current Babble deployments (see Table 1 for a summary). It is important to note that the

explicit analysis we present here did not drive the design; it is really to persuade the reader. Having spent years watching the users of dozens of Babbles structure and re-structure their environments, the conclusions drawn from this data were already evident to us.

Perhaps the most obvious feature of these Babble deployments are the number of categories and topics and categories in evidence. The number of user-created topics and categories per Babble ranges from 62 to 170, with the average being 129 (these counts exclude automatic archives of conversations generated by the system, and topics and categories that users deleted). When one considers that there are a relatively small number of regular, active users who contribute to conversations (typically 10 to 20, but around 30 for B1), this is quite a lot. Why is this happening?

What we see when we look at the lists of categories and topics is that quite a bit of the structure which has been generated is for presenting and organizing static information. That is, conversation topics are often not used for conversation. Instead, what we see is that users are trying to create meaningful structures which, while they often contain niches for conversation, are larger in scope.

Table 1 summarizes a few of the most common structure types observed across Babbles. These include personal places (places 'owned' by a particular user), structures designed to support events (e.g. an upcoming conference) or projects, and places for announcements. Note that the counts shown in Table 1 are quite conservative; they reflect only existing structure (not structure created and later deleted by users), and only structures that are named so that an outsider can recognize their purpose.

The most prevalent form of structure is what we will refer to as an 'office'. Offices are topics, or, most often, hierarchies of categories and topics, that 'belong to' and are named after a user. The third row of table 1 shows the number of distinct offices in each Babble (the top number), the total number of topics and categories devoted to offices (the second number), and the percentage of structure in that deployment that is devoted to offices. As Table 1 shows, offices comprise from one to three quarters of the structure (i.e. number of category names and topic names) in these Babble deployments. A common form for an office is:

Pat's Place
    About Me
    Talk with Me

the first item being a category, and the next two items being topics contained within it. The first topic is typically intended to contain a profile of the person, and the second as a place for conversation.

The event and project structures shown in row 4 are similar offices, in that the distinguish between topics intended for conversation and topics for information, although they typically contain much more structure. For example, one Babble uses project names as categories, and underneath the project name uses topics with names like "Current status", "Meet the project members", and "Tell us what you think!" In general, this approach, of using some topics to contain static information, and designating particular topics as specially for conversation, occurs across all Babble deployments for a variety of different types of structures.

	B1	B2	B3	B4	B5
Months of use	10	6	6	6	5
Total structure in terms of # of topics + categories	141	147	170	62	126
# of distinct 'offices'	28	47	20	15	12
Amount of structure (t+c)	90	109	46	38	33
% of total structure	64%	74%	27%	62%	26%
# distinct events/projects	3	1	2	1	3
Amount of structure (t+c)	14	9	19	1	53
# announcement topics	1	0	2	2	3
Amount of structure (t+c)	1	0	2	2	3

Table 1. Summary of the use of structure — topics (t) and categories (c) —in five active Babble deployments.

Another structural feature found in most Babble deployments is the attempt to explicitly or implicitly create one or more topics or categories for announcements (row 5 of Table 1). Most often these are named "Announcements," since that results in them appearing at or near the top of the alphabetically sorted topic list; other examples are "Heads Up!," "News," and special purpose announcements like "Kittens Free to a Good Home!" (However, most announcement structures do not seem successful, judging by their degree of use; one of the moderators reports that people wouldn't come to the announcements topic quickly enough, and so he shifted to posting announcements in a topic where most participants in that Babble 'hung out.')

To summarize, throughout the various deployments it is clear that users are doing more than creating places to talk. First, they are trying to display static information in a readily accessible way. Thus, they resort to using topic names to signal whether the topic is supposed to be a place for conversation (e.g. "Talk to me!" "Questions," "Chit-chat," "Discuss <Project Name> Here"), or whether it is primarily informational ("About Me," "<Project Name> Status," "About"). Second, users are trying to make certain types of information visible. The most obvious example of this is the "Announcements" topic, which, by virtue of its name, appears at the top of the alphabetically sorted list. More generally, all five Babble deployments exhibit attempts to structure topics within particular categories by using numbers or punctuation characters as prefixes to control their sorting order (most often trying to put informational topics first in the list as with "-Where to Start"). Both the desire to display structured static information, and the desire to control the visibility of information, are taken up in the design of Loops.

LOOPS: THE WORKING SYSTEM

Thus far we've described our overall goals, the design context, and the factors which shaped the design of the new system. In this section we describe the resulting system. Currentlly, all the major functionality is implemented, and it is being used by our group; we still have minor features to add, and performance problems to resolve before it is ready for distribution outside the group.

Conceptually, Loops consists of a set of user-definable rooms, each of which can contain a conversation, tools, documents, static text, and people. The basic user

experience is that people connect to Loops server, and move from room to room, reading conversations that have changed in their absence, contributing new comments, and encountering other users as they do so. As with Babble, the ultimate goal is that Loops feel like an inhabited place, in which users may 'hang out' during the day as they work on their computers, or into which they may occasionally venture to see what has happened in their absence.

The Architecture

Loops uses a client-server architecture, with the client and server communicating via TCP/IP, with content coded in XML. A server written in Java provides a persistent store; the client is implemented in Macromedia's Flash 5 and runs in any standard web browser that supports the Flash plug-in. The server maintains a list of 'currently connected' clients, the contents of the Loop's conversations, and a log of user activities used to drive the Loop's social proxy and other elements of the interface that show traces or results of user activity. This architecture allows us to track user location and activity within the conversation space (e.g., which conversation a user is 'in', how long since the user visited or posted to a conversation, etc.), thus supporting the user experience described in the next section.

Figure 2 illustrates how the architecture works. The story begins with the user of client 1 entering a remark. The client wraps the comment in XML (along with meta information regarding the conversation it is being posted to) and sends it to the server (e.g. the "Post_Item(text)" request). Upon receipt, the server processes the request: it verifies that the client has appropriate permissions, creates a command by adding meta-information such as the client_ID and time, and stores the command in its activity log. Then, the server broadcasts the result to all connected clients (e.g. the "I_Posted(client1,text,time)" command), including the originating client. When the clients receive the command from the server they act appropriately (in this case, displaying the new comment in the conversation window, and updating the social proxy to reflect client 1's activity).

This story is complicated by two factors. First, not all users may be connected simultaneously. As shown in Figure 2 (grey area), when Client 3 connects at a later time, it issues a "Get_Activity" request to the server. The server responds by sending it the activity and content for the conversation that Client 3 is viewing ("I_Posted (client1,text,time")), which, in this example, is the same conversation to which Client 1 originally posted. The second complicating factor has to do with the (common) case in which all connected users are not viewing the same conversation. Thus, suppose that Client 4 (not shown) was present when Client 1's comment was posted; but was viewing a different conversation. In that case, Client 4 would receive the same I_Posted command that the server broadcast to Clients 1 and 2, but would use it differently: Client 2 would update elements of its social proxy (e.g. those that show the time of Client 1's most recent activity), but would do nothing with the comment itself, since it was displaying a different conversation; instead, if and when it switched into Client 1's conversation, it would use Get_Activity to obtain the contents.

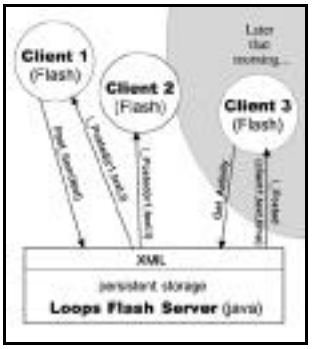

Figure 2. The Architecture

The User Interface

We'll begin with an overview of the general interface elements of Loops. The basic features shown on the screen layout (Figure 3) are, clockwise from the lower left:

- a conversation pane
- the social proxy (above), which provides cues about the awareness and presence of others
- a bulletin board for editable static text (to the right);
- a room pallet (lower right) with a categorized list of conversation rooms and tools for manipulating them;
- a user pallet (lower middle) containing a list of active users and tools for manipulating user information

In this example, four people are connected, three of whom are together in the "Commons Area", the room being shown. The particular elements of the "Commons Area" room are the conversation pane (showing a segment of not-quite-synchronous talk about producing slides for a meeting), the social proxy (showing that three people are in the room), and the bulletin board (containing reminders about an upcoming group meeting and the ECSCW conference). To the right of the conversation pane, the user pallet shows that four people are connected to Loops as a whole, and, to the right of that, the room pallet shows the hierarchical structure of this conversation space (primarily personal offices and a debugging topic), and provides indications of where people are located (in this example we can see that someone is in "Christine's Office/Talk with me" by the small icon to the left of the room name).

Awareness and Conversation

Because the awareness and conversation models were transposed from Babble, we'll cover them briefly.

The chief awareness interface element is the social proxy, the circle in the upper left corner of Figure 3. The circle represents the conversation being viewed, and the colored dots (referred to as marbles) represent people. A marble inside the circle means that a user is in the current conversation; when users are active (meaning that either they type or click) their marbles move to the inner ring of the circle (as in Figure 3), and then, over the course of 15 minutes, they drift to the edge of the circle. Marbles shown outside the circle represent users who are connected to Loops, but are viewing different conversations.

Loops implements the lightweight type-and-post, conversation-in-one-window model that Babble supported. A user adds to a conversation by clicking on the speech bubble icon in the lower right corner of the conversation pane and typing into a moveable dialog box. This arrangement, which differs from the standard chat/instant messaging model, enables people to refer to the existing conversation, or even to move to and copy from other

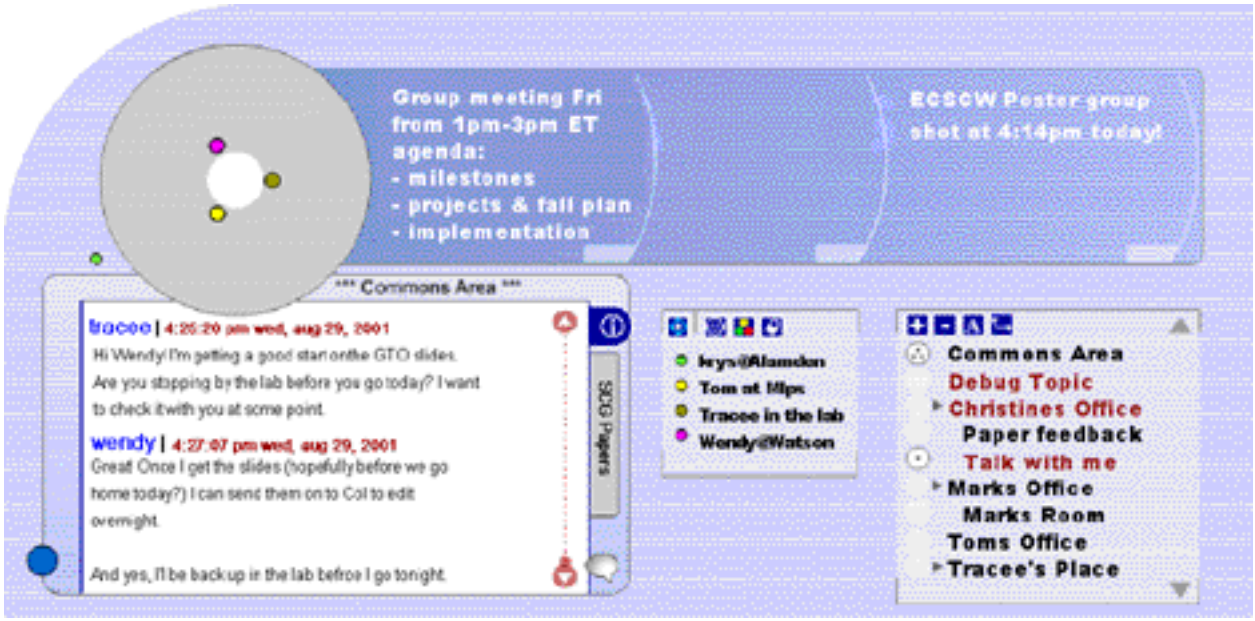

Figure 4. The Loops User Interface

conversations, thus making it easier for participants to compose reflective and synthetic responses. Once the user posts the comment, it immediately appears in the conversation. For users who are in other rooms (or who log on later), the name of the room in the rooms pallet turns red to indicate that there is new content.

Static Structure: Bulletin Boards and Tabs

As noted earlier, there was considerable evidence that Babble users devoted a lot of effort to making information accessible. In response to this, Loops provides two ways of structuring static information: bulletin boards and tabs.

Bulletin boards (Figure 3, top middle to right) provide a means for posting text in a highly visible place. Each room has its own bulletin board, and its text may be edited by anyone who has write permission for the room. When new or changed text is posted to a bulletin board, the new text is signaled to those in the room by the background color of the bulletin board fading out and then fading back in with the new text displayed. We anticipate that bulletin boards will be used for purposes ranging from announcements and reminders to scene setting (e.g. "You see a messy office."), based on observations of and comments from Babble users.

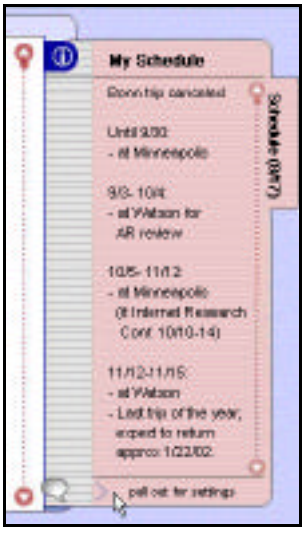

Figure 4. An opened tab.

The other means for making information readily available is the tab. Each room has its own tab, which peeks out from behind the conversation pane. Clicking on the tab causes it to slide out (Figure 4), revealing the (editable) information on it; a second click provides access to controls for setting the background color, editing the tab title and heading, and adding and removing additional tabs. We expect that tabs will be used for activities such as sharing lists of URLs, schedules, and keeping to do lists.

One role that we expect both bulletin boards and tabs will play is to visually distinguish rooms from one another. In Babble, rooms looked almost identical; in Loops, the text in the bulletin board, and the number, colors and names of tabs will make rooms feel different from one another.

SEMI-STRUCTURED CONVERSATION

The previous section describes the working version of Loops that we are using as of the time of writing. Essentially, this version of Loops supports general purpose online conversations, albeit with mechanisms for customizing certain static elements of rooms. In this section we describe current development work, which uses the architecture and interaction mechanisms to provide 'scaffolding' semi-structured conversations.

Our interest in supporting semi-structured conversations comes from our observations of various uses of Babble. For example, in the Babble deployments previously described, there are many examples of efforts to enact particular types of conversations such as interviews, meetings, and brainstorming sessions. In fact, in response to requests, we deployed three Babble's to support short term events (e.g. a month long online brainstorming session), rather than the longer duration deployments for which we designed it.

There is, of course, a long history of work in HCI that has to do with supporting various types of structure in conversation. This includes research on workflow, structured hypertext, design rationale, and, most recently, threaded chat [7]. Our approach differs from most of these,

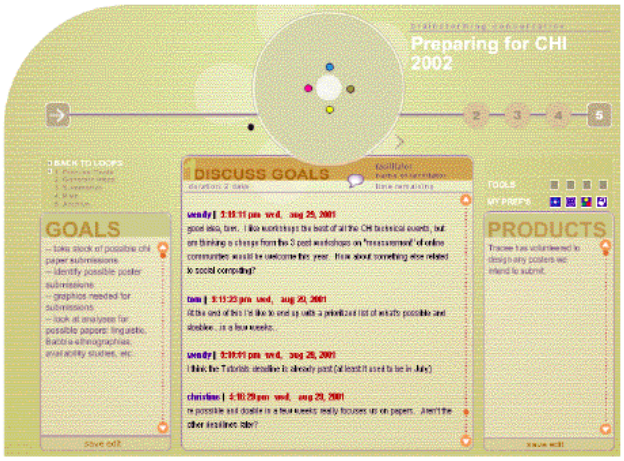

Figure 5. Design for a semi-structured 'Brainstorming' conversation in Loops.

in that it tries to support semi-structured interaction primarily by creating a mutual awareness among participants of who is doing what, thus permitting self-organization.

The Rationale

The basic idea is to provide support for a structured process without forcing individuals to work in lockstep with one another or to proceed through a particular process without deviation. The strategy that we follow is to create a visual analog of the conversational structure within which people may work. However, rather than trying to build constraints on their actions or interactions into the system, instead, we make the activities of the participants visible to one another. That is, we hope to use social pressures to allow people to interact coherently with one another, rather than trying to enforce a particular interaction pattern via hard-wired technical constraints. Or, to use an analogy, when a person is giving a presentation to a co-located meeting, we do not gag the other participants to keep them from speaking, nor do we lock the door to prevent them from leaving; instead, we rely on social norms and the fact that individuals' behaviors are visible and audible to all participants. Because everyone has the option to interrupt or walk out, it means something when people refrain; and because social norms and pressures are in play, it also means something when people interrupt or walk out.

The Working Prototype

The interface shown in Figure 5 is a partially implemented working prototype[1] designed to support such an approach. Because this is exploratory work, we begin by designing to support a particular type of conversation that we know to be of interest to potential users. Should the approach work, we would then look at ways of generalizing it, either by providing a set of templates for conversation types, or perhaps by designing a scripting language for semi-structured interactions (as has been done in [7]).

Figure 5 shows a Loops semi-structured conversation room that supports a five stage brainstorming process. This room differs from a normal Loops room in two obvious ways: first, the social proxy has grown larger, containing elements that represent each of the process' stages; second, the conversation pane has grown, and is flanked on either side by 'capture panes', that is, editable text areas which are used to capture key items that persist across stages. This expansion has eliminated the general purpose features of rooms — the bulletin board, tabs, and the user and room palettes — in favor of items which support brainstorming.

To describe the way the prototype works, we'll start off by summarizing the brainstorming process which it supports (obviously, there are many different forms of brainstorming), and then discuss the prototype as it is used to move through the process. The process supported in the prototype is straightforward: discuss the goals; generate ideas in a non-critical mode; codify and elaborate ideas; critique and rank ideas; archive the results. The prototype works by providing a 'sub-room' for each of the five stages, and static text areas to either side of conversation pane to capture products of the discussion to be carried on to future stages.

The idea is that participants gather in the first room to discuss the goals; the result of this discussion is a list of agreed upon goals which are posted in the "Goals" area to the left so that they can be referred to in future stages. Once this is done, the group shifts to the second stage. However, we do not enforce this model in the system; participants can, in fact, move back and forth among stages regardless of where the group's focus currently is.

Instead, by making the location and activities of participants visible across the entire process, we aim to allow social pressures to encourage coordinated action. We do this in the following way: When a person moves from one stage to another, their marble animates and moves to the appropriate circle in the proxy (i.e. circle 2) above. From the movers point of view, the stage 1 subroom shrinks, and the newly entered stage 2 room expands so that it can show the fine structure of interaction within it; from the viewpoints of the other participants who are still in the stage 1 sub-room, the mover's marble moves into sub-room 2, which remains in its minimal state. Thus, if people are acting in a coordinated fashion, all participants ought to see a general movement of marbles between the stage 1 and stage 2 subrooms. On the other hand, it may be that some of the group moves, and two or three members remain behind; since the proxy makes this situation visible, it can serve as a sign that perhaps more work (or at least negotiation) is needed. In summary, this allows people to see where others are and make inferences about what they are doing. By making collective activity visible (as well as deviations from it), we hope to enable a group to act coherently (either by working together, or by

[1] If this seems an odd phrase, note that our prototyping and implementation environment, Flash, are the same. Thus, the details of the interaction design are worked out; what remains is the work of 'wiring up' the server so that it can drive the Flash visualizations. The screenshot shown in Figure 5 is a running version shown in the process of being used; however only some elements of participants behavior is reflected in the proxy.

recognizing when it may be necessary or useful to relax the expectation of simultaneous collective activity).

So far, we've seen that making the sub-room(s) inhabited by participants visible, serves as a signal about what stage(s) of the process is being focused on. In addition, the proxy will also reflect where new information has appeared (via the background of a sub-room changing color), so that we can distinguish between (for example), participants returning to a previous stage to refer to something, and people returning to a previous stage to add more information, the latter being likely to attract more attention. This second mechanism is important because a Loops brainstorming room need not be used only in synchronous mode. That is, it might be useful for a group (either because of scheduling difficulties, or because the group is scattered across time zones around the world, as is not uncommon in ALC task forces and *ad hoc* groups), to conduct an asynchronous brainstorming process. One might devote one day to each stage, so that participants checking in on Wednesday (i.e. stage 3, 'Idea Codification and Elaboration'), might notice that there was activity in stage 2 that had occurred since their last visit on Tuesday, or even that there was stage 1 activity where someone might have suggested expanding the set of goals.

We have a variety of other features working at the interaction design level. These include tools to support activity in various stages (for example, an 'idea counter' for the idea generation phase, and a voting mechanism for the ranking stage), and other indicators to enhance awareness of activity elsewhere. However we'll defer discussion of those features since our primary aim was to show how the exploratory work follows from our goals and leverages the architectural and UI work that we've already implemented.

CLOSING REMARKS

While we have framed this paper in practical terms — as an attempt to support coherent, online interaction among work groups — it can also be seen quite differently. From another direction, our program of design research may be seen as exploring, *in situ*, the implications of various programs of anthropological and sociological research so beautifully pulled together in Lucy Suchman's *Plans and Situated Actions*[9]. Suchman argues against the notion that behavior is planned, i.e. that abstract cognitive plans serve as the origin of and controlling structure for behavior. Instead, she argues that behavior emerges from local interactions between the actor and the details of the particular situation, and that the role of plans is to serve as a resource for behavior, just as maps help travelers determine, but do not control, their paths.

In the case of Babble and Loops, the social proxies we've designed, and the conversational 'scaffolding' we described in the previous section, are attempts to make abstract representations concrete, so that they may better serve as resources for collective behavior (other research programs, for example those of Donath and her colleagues [2], and Sack [00] may also be seen in this light). By making abstract structures visible, by allowing people to inhabit them and move within in them, and by enabling people to see and discuss one another's positions within the structures, our goal is to provide a texture of visible and audible cues to which inhabitants may react, and around which they may orient their behavior. Because we have yet to see how Loops fares in deployment, and thus in what ways the inhabitants of Loops make use of the inhabited visualizations that we've described here, we have focused on describing this work in terms of its instrumental and practical goals. Nevertheless, we want to emphasize that our work not only draws upon research, but has the potential to address questions of fundamental interest to researchers. We think it apropos to close with the final sentence from *Plans and Situated Actions*: "Just as the project of building intelligent artifacts has been enlisted in the service of a theory of mind, the attempt to build interactive artifacts, taken seriously, could contribute much to an account of situated human action and shared understanding. [9, p 189]."

REFERENCES

1. Bradner, E., Kellogg, W, & Erickson, T. The Adoption and Use of Babble: A Field Study of Chat in the Workplace. *The Proceedings of the European Computer Supported Cooperative Work Conference.* 1999.
2. Donath, J., Karahalios, K., & Viegas, F. (1999). Visualizing conversation. *Proceedings of HICSS-32*, Maui, HI, January 5-8, 1999.
3. Erickson, T. and Kellogg, W. Social Translucence: An Approach to Designing Systems that Mesh with Social Processes. *Transactions on Computer-Human Interaction.* Vol. 7, No. 1, pp 59-83. New York: ACM Press, 2000.
4. Erickson, T. Smith, D. N., Kellogg, W. A., Laff, M. R., Richards, J. T., and Bradner, E. Socially Translucent Systems: Social Proxies, Persistent Conversation, and the Design of 'Babble.' *Human Factors in Computing Systems: The Proceedings of CHI '99*. ACM Press, 1999.
5. Farnham, S., Chesley, H.R., McGhee, D. E., Kawal, R. and Landau, J. Structured online interactions: improving the decision-making of small discussion groups; *Proceedings of the ACM 2000 Conference on Computer supported cooperative work*, 2000, Pages 299 - 308
6. Houde , S. and Sellman, R. In search of design principles for programming environments. *Human Factors in Computing Systems: The Proceedings of CHI '94*, pp. 424-430. New York: ACM Press, 1994.
7. Sack W., Discourse Diagrams: Interface Design for Very Large Scale Conversations. *Proceedings of HICSS - 33rd Annual Hawaii International conference on Systems Sciences*, January 4-7, 2000
8. Schön, D. A. Educating the Reflective Practitioner. San Francisco: Jossey-Bass, 1987.
9. Suchman, L. Plans and Situated Actions: The problem of human machine communication. Cambridge: Cambridge University Press, 1987.