

ProChat: Dynamic Formal Collaboration Protocols in a Chat Tool for Handheld Collaboration

Randal K. Whitehead, David Stotts

Department of Computer Science
University of North Carolina
Chapel Hill, NC 27599-3175
+1 919 962 1981
{whitehea, stotts}@cs.unc.edu

ABSTRACT

ProChat is a simple chat tool allowing two users to exchange text messages between handheld computers over an infrared link. Unlike most chat tools, however, the rules governing group interaction, i.e., who holds the floor, how many people can hold the floor, how the floor is passed, etc., are encoded separately from the application. These rules, or *collaboration protocols*, can be developed outside the application, analyzed for errors, shared among users and changed dynamically. In this paper we describe the architecture of ProChat, and how the underlying protocols are used to modify the user interface to control the collaboration. We conclude with some open questions raised by the work done so far.

Keywords

handheld computing, computer supported cooperative work (CSCW), collaboration protocols, mobile computing

INTRODUCTION

In synchronous collaborations there exist some set of rules governing what each participant can do. These rules, or protocols, may be encoded within the collaborative application, may be part of a broader social protocol governing group interaction, or may be otherwise part of the physical environment.

ProChat explores the specification of such protocols by applying previous work in externally specified collaboration protocols to synchronous collaboration using handheld computers. We discuss a simple collaborative application, a chat tool, running on a popular brand of personal digital assistant (PDA), the Palm III. (Because the concepts described here are not specific to the Palm, we will refer to the devices as PDAs for the remainder of this paper.) ProChat supports a number of existing chat protocols (open-floor, floor passing, moderated), but new ones can be created, specified, and added dynamically. These rules are expressed in a formal, analyzable notation, and their dynamic nature allows the user to tailor the behavior of the system to their needs. Newcomers to a ProChat session can have the protocol in use uploaded to them if they do not already have it in their application database. In essence, newcomers to a ProChat session can be “taught the rules”.

Motivation for the Work

Collaborative software systems of the future will be far more complex than the simple systems we have today. As information transmission technology improves, we will be able to support multi-user applications that more nearly equal the complexity of interactions people naturally have face-to-face. Our research is developing the formal structure necessary to reliably produce and maintain collaborative systems that are far more complex than today's systems. This is a focussed investigation in formal methods for collaborative software systems, and is an outgrowth of earlier investigations in Trellis multi-user hypermedia.

We are developing a specification method for defining intricate rules of collaborative interaction in a group computing application, and we are producing analysis

algorithms for verifying the correctness of the rules. We call such group rules *formal collaboration protocols*. Collaboration protocols in current CSCW applications are implicitly defined by code segments scattered throughout the application. A better method from a software engineering viewpoint is to explicitly define the collaboration semantics and implement it specifically as an identifiable module in the system. The advantages of this are:

- ease of reading and understanding the rules;
- formal notation allows analysis for errors;
- ease of developing collaborative systems with modular components;
- dynamic replacement of rules as collaboration progresses is possible.

The technical issues being examined include the specification notation(s) to use and the mechanisms for enacting and enforcing the rules across distributed networks. Collaboration protocols must capture in a formal notation these aspects of group interactions:

- Who are the group members?
- Where are the group members?
- What are the goals of the group activity?
- What are the capabilities of each member?
- What resources are required by each member?
- What information will each member contribute to the activity?
- What concurrent threads can exist in overall group activity?
- When must the threads be synchronized?

This list is just a starting point. We expect to discover more characteristics as work progresses.

In addition to capturing informational aspects of group interactions, a full specification must be in a formal notation so that analysis and reasoning algorithms may be developed and applied in support of system development and execution. Collaboration protocols must be unambiguous and formal. Informal notations convey information to humans but do not support analysis and automated reasoning. Collaboration protocols must however be flexible, not unusably rigid. People are not automata; this is what makes CSCW so fascinating as an area of study. They often need guidelines more than "rules". They need freedom to relax or sidestep rules when circumstances are not adequately covered by those rules. As people discover better methods and processes, the rules

need to allow modifications and updates. We want to develop a protocol specification method that will be both formal and flexible. Rules need to be malleable while still conveying clear semantic information to the system using them. People need to feel *in control of* a system, not *controlled by* a system.

RELATED WORK

ProChat builds on previous work in two fields; handheld computer supported collaborative work (HCSCW) and formal collaboration protocols.

Handheld CSCW

For the purposes of this work we are interested in HCSCW systems in which there is a synchronous collaborative interaction between two or more users. In these cases there exists some set of rules that control how the participants interact, whether these rules are encoded within the application or are external social protocols governing other group interactions. Protocols found in HCSCW applications generally fall into three categories: social protocols, open-floor protocols, or asynchronous protocols. We give examples of each in the following system descriptions.

The various applications associated with the PEBBLES project [11] do not have explicit floor control mechanisms built in, relying instead on social protocols to prevent conflicting user actions. This allows great flexibility of user actions and leverages more information than could be practically encoded in a formal protocol, but may break down if the oral communication central to many social interactions is not possible.

An existing HCSCW chat tool, IrChat [12], also has no explicit floor control, but because there are no conflicting user actions it uses an open-floor protocol allowing all users to send messages simultaneously. This is simple to use, but does not support situations where a more structured interaction is needed.

Some HCSCW applications, such as NotePals [10], support users working synchronously, but assume an asynchronous merge of individual work. This is equivalent to a strong floor passing protocol where each synchronizing user "holds the floor" until their sync is complete.

SharedNotes [8] is an interesting exception to these categories that bears closer examination. In the SharedNotes system, a user creates personal notes in advance of a meeting on their PDA. They have the option of marking some of the notes to be “publicized.” Prior to the meeting the user syncs their PDA with a public PC in the meeting room. This loads the notes they had marked to publicize onto the public display. Once the meeting begins, all of the participants can use their PDAs to add annotations to any of the public notes or edit their own annotations. Any public text, though, must be edited on the public display. Thus there is a protocol “encoded” in the physical environment. That is, what text a user can edit during a meeting is based on what physical device they are using. They can edit annotations they created on their PDA, but public text can only be edited in the public domain (i.e., on the public display). There is a protocol controlling the group interaction, but its basis in the *physical* environment makes it difficult, if not impossible, to modify.

In ProChat, we provide protocols to control group interaction, but make them flexible by formally defining them separate from the code of the application.

Formal Collaboration Protocols

As previously stated, for most collaborative applications the rules governing the interaction of the users are fixed and embedded within the code of the application. This fixed behavior amounts to a “one size fits all” approach to collaboration. However, there is no one set of interaction rules that applies to all users in all situations. Even a system that allows for flexible collaboration rules imposes inflexibility on the end user if the application must be recompiled to change the rules, as is the case with Suite [4, 5, 13].

Flexible collaboration is possible if the interaction rules are specified separately from the application and can be loaded dynamically by the system. The rules then become *formal collaboration protocols* [6], which can be specified unambiguously, analyzed for logical errors, and dynamically shared among users. The use of collaboration protocols in hypermedia documents has been explored previously in the Trellis [14, 15, 7] and CobWeb [17] projects.

Trellis was a formal model of hyperdocuments, as well as several hypermedia system prototypes based on this model [14]. A Trellis document was built on a place/transition net (Petri net, or P/T net) formalism. This specified formally not only the document linked structure (the graph of the automaton) but the browse-time behavior (the browsing

semantics) as well. In the systems based on Trellis, the collaborative browsing semantics were embedded within each document. Later versions of Trellis were based on colored nets, allowing individual users to be distinguished from one another. This advance allowed the encoding of collaborative browsing behavior directly in a hyperdocument, and allowed Trellis documents to be used as “executable” specifications for collaborative systems [6].

CobWeb [17] built on the results of Trellis by separating the protocol formalism out from the structure of hyperdocuments and applying them in more general contexts. The protocols are still expressed formally as colored nets, but are implemented as Java classes in early prototypes. These classes are stored centrally on the CobWeb collaboration server and downloaded by each connecting user. Later CobWeb prototypes use a net specification notation rather than Java classes, and this net specification language is the one employed in ProChat as well.

ProChat applies the concepts of collaboration protocols to a simple collaborative application (a “chat” tool) running on a handheld computer (a PDA) and communicating with other devices via a wireless (infrared) link. There is no server so there is no central repository of protocol specifications. Instead a user can carry with them a collection of their favorite protocols and can choose any of those when starting a session. When connecting to an existing session, the protocol in use within that session will be chosen automatically, perhaps including installation of the protocol on the new user’s device. If a user wishes to join a collaborative session, but does not have the application installed, the entire application can be sent over the infrared link by “beaming,” a common feature of Palm devices. This kind of ad hoc collaboration is well suited to the use of small portable devices.

Analysis of Collaboration Protocols

One of the main reasons cited in the motivation section for using formal notations for collaboration protocols is to allow automated analysis of the structures for logical errors. The approach we take is to use a modified form of model checking. Model checking was originally developed by Emerson and Clarke for verification of hardware circuits [3]. We have successfully adapted it to the analysis of the logical structure of collaboration protocols by converting the colored nets into appropriate models [16]. Temporal logic (an extension of the predicate calculus) is used to express the logical properties a protocol should have. The model (a finite state machine annotated with atomic true

predicates) is then traversed to determine the truth of the temporal formulae. In our work, the colored nets provide the model structure. A full presentation of temporal logic and model checking is beyond the scope of this paper.

We have developed (in CobWeb) several sample floor-control protocols, including a moderated meeting protocol. In this protocol, a moderator determines who can join a collaboration. He has the ability to take the floor away from a speaker if he so desires; the speaker is put on hold, with the intention that the moderator will return the floor to him rather than handing it off to a different speaker. Our analysis tool discovered an error in the original formulation of this protocol. It was possible to place a speaker on hold and then leave him stranded there. The error was subtle and not obvious from the structure of the net.

PROCHAT ARCHITECTURE

ProChat is a tool that allows two users to pass simple text messages between handheld computers. It is implemented in C and runs on the Palm III and later models. In its simplest state, it resembles an ordinary chat interface commonly found on web pages and elsewhere.

At startup, the user chooses an alias to identify messages they send (see figure 1). The default chat interface (see figure 2) includes a scrolling text field for a transcript of the chat session, a text field showing the user's alias for the session, a text field for the composition of a new message, a *Clear* button that erases the current message, and a *Send* button that sends the current message.



Figure 1: Setting ProChat Alias

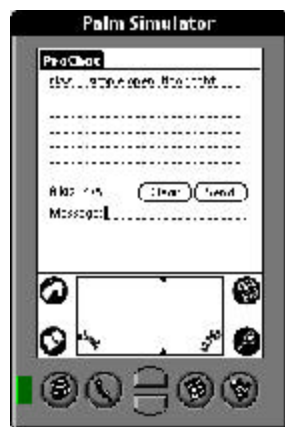


Figure 2: Default Chat Interface

Because the goal of this project is to explore collaboration protocols and not infrared networking, the chat session is

limited to two users. This limitation, however, is only in the networking. The collaboration protocols and their implementation would support a larger number of users. In a later section we consider some additional issues that would be confronted if the system were scaled to a larger number of participants.

Interpreting Nets as Collaboration Protocols

In each of the systems using collaboration protocols, the protocol is expressed formally as a colored Petri net. In such a system, the users are represented by tokens in the net. The places in the net correspond to various user states. The transitions act as pushbuttons, modifying the state of the net and moving tokens (users) from one place (state) to another. The enablement of a transition based on the rules of the net control what actions can be taken based on a user's state.

The remaining discussion of ProChat will focus on how the net underlying a protocol guides group interaction. For the purposes of this discussion we will italicize place and transition names and will underline place names. This will avoid confusion when describing how places and *transitions* in the net control the application.

Interface Defined by Collaboration Protocol

In addition to the default interface features, there are six buttons whose labels, visibility, and enablement are controlled programmatically based on the state of the current protocol. The effect of pressing any of these buttons is to change the state of the underlying Petri net by firing the associated transition. The enablement of the *Send* button is also controlled by the protocol, but it always has the effect of sending the current message.

Augmenting the net with additional rules links the interface and underlying Petri net specification. As in the CobWeb system, *visibility rules* define which transitions are visible from which places, and *enablement rules* define which transitions are enabled from which places (in addition to the enablement defined by the state of the net). *Layout rules* are new in ProChat and associate transition names with the programmable buttons in the interface, giving the protocol developer greater control over the user interface. A user is represented in the net by a token whose identifier is the user's chat alias. The programmable buttons are modified based on the visibility, enablement, and layout of transitions connected to that token's current place.

The default protocol (figure 3) is a simple one, with a single place, *Speaker*, and a single transition, *Send*, which is further defined by these rules:

VISIBLE	
"Send"	"*"
ENABLED	
"Send"	"Speaker"
LAYOUT	
"Send"	6

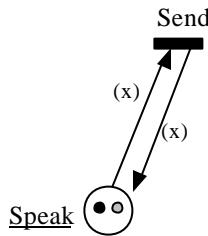


Figure 3: Default Chat Protocol

All users are represented by tokens in *Speaker*, so *Send* is enabled and visible for all users, giving the equivalent of an open-floor chat.

Things become more interesting, though, when a more complex protocol is used, such as in figure 4. The floor-passing protocol defined by the Petri net shown in figure 4 is augmented by these rules:

VISIBLE	
"Release"	"Speaker"
"GetFloor"	"Listener"
"Send"	"*"
ENABLED	
"Release"	"Speaker"
"GetFloor"	"Listener"
"Send"	"Speaker"
LAYOUT	
"Release"	0
"GetFloor"	1
"Send"	6

The transitions *Release*, *GetFloor*, and *Send*, are visible to all users, but are enabled only for users in certain states. For example, *Send* is only enabled to speakers and *Mutex* ensures only a single token in *Speaker*, thus enforcing floor control. As the user requests or gives up control of the floor, the interface is modified based on the updated state of the net. For example, a listener sees the interface as in figure 5, while the speaker see the interface in figure 6. When a listener presses the *GetFloor* button and becomes the speaker, the *Send* and *Release* buttons become enabled.

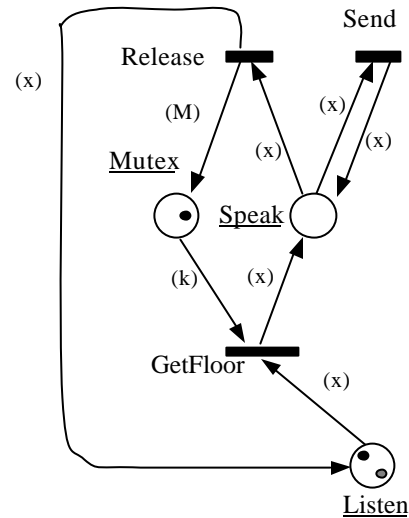


Figure 4: Simple Floor-Passing Protocol



Figure 5: Listener Interface



Figure 6: Speaker Interface

Representation of protocols

The protocols used by ProChat are stored in a database as a packed record including the number of places, transitions, arcs, etc. and a single text field with the place, transition, token, and variable names concatenated separated by null bytes. This packed record is also used for transferring

protocol specifications between devices, and is unpacked into specifications for places, transitions, arcs, etc. before instantiation.

Infrared Communication

Communication between two users is by means of the built-in IrDA standard [9] infrared (IR) communication commonly available on PDAs. IR was chosen so that no assumption would be made about the availability of communication hardware or software. There are, however, severe limits to IR communication as implemented for these devices, primarily the narrow beam width and short range, which make this feature impractical for an extended chat session. However, the important features of ProChat are independent of the communication medium. Other media, such as RF may prove to be more practical.

The data sent over the IR link fall into three categories:

- Chat messages
- Event Messages
- Control Messages

Chat messages are simply the text messages sent from one user to another and include both the message and the alias of the sender.

Event messages send interface events (button pushes) affecting the state of the net. Each event indicates the transition associated with the pressed button and the token representing the user who pressed that button. Because the current implementation only supports two users, there is no global ordering applied to event messages. If a larger number of users were supported then such an ordering would likely be required.

Control messages include IR session startup and shutdown, and the sharing and setting of collaboration protocols. The sharing and setting of protocols is a particularly important feature of ProChat.

When a user joins an existing chat session they accept the protocol already in use. The setting of the protocol is done via negotiation as described in figure 7. A connecting user is first told what protocol is in use. If a corresponding protocol specification is not found then its transfer is automatically requested. Once the protocol has been transferred it is instantiated and the current state transferred. All of this is done transparently and the user need not even be aware of the existence of the protocol

specification – only that the chat session takes on features that are compatible with other participants.

It is also possible for a user to explicitly send a protocol to another user, in which case it is stored on other device, but not instantiated. A user may also choose to change protocols during a chat. In that case the same protocol

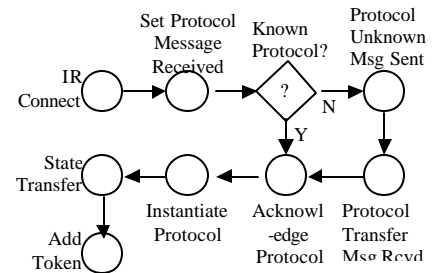


Figure 7: Protocol Negotiation

negotiation and instantiation done at startup is done on both sides.

Limitations and Future Work

The greatest limitation of the current work is the connection between devices. We chose IR as the communication medium because it was built into the current generation of handheld devices and thus requires no additional external networking hardware or software. Unfortunately, the basic IR support available limits us to a connection between two participants, and even that requires that the participants be within one meter of each other and that the devices be closely aligned, making this solution impractical for an extended chat session.

More importantly, this limitation has prevented us from exploring some important issues that arise as the system scales to three or more participants. In general these issues arise from the lack of a centralized server in the current architecture. If there are more than two participants, to which existing participant does the new participant connect initially? For IrDA support, there is an IR “discovery” phase during which a user selects device among all of the IR equipped devices within range. Would a similar discovery be used to select an entry point into the chat? Once connected, how would messages be routed to other participants? In an IR environment, Beigl’s multi-hop routing [1] might be used to answer these questions.

A more intriguing possibility is an emerging RF technology such as Bluetooth [2]. In the Bluetooth usage model, enabled devices connect and synchronize automatically when coming within range. This model fits perfectly with the

automatic protocol negotiation already present in ProChat. Unfortunately, there are no commercial devices available yet.

Despite these limitations, the work done so far demonstrates the principles of using collaboration protocols to control the user interface of a collaborative application. We are also considering what other types of protocols could be employed by ProChat to go beyond simple chat functionality. For example, could voting or election protocols be added to a chat protocol to allow public debate and private ballots? Finally, a mechanism for editing or creating protocol specifications – either on a PDA or the desktop – is needed.

CONCLUSIONS

By allowing users of a collaborative application to dynamically create and modify the protocols controlling group interaction, we allow them to tailor the collaboration to their needs. ProChat applies this concept to an HCSCW application for which collaboration protocols have previously been loosely applied (open-floor) or based in broader social protocols. By having formally encoded, yet flexible protocols we give the user freedom to control the collaboration, but within a provably correct framework. We also allow a dynamic switch from social protocols to encoded protocols in situations where social protocols may not be practical (e.g., during a larger meeting). We also have created a tool in which newcomers can learn the rules under which the group is operating, without having to have them in advance as part of a hardcoded application; this is especially important if a system is to allow rules to be altered during collaboration.

Work remains to scale the system to more than two participants and to non-chat protocols, to explore communications media that might be more practical than IR, and to establish mechanisms for the creation and modification of protocols.

Larger, More Complex Protocols

Although the work reported here is necessarily using small protocols, we envision and intend these results to scale to large and complex collaboration protocols. An example of such a protocol would be *Roberts Rules of Order*, used extensively to manage parliamentary procedure world-wide. The advantage of our approach is in the dynamic possibilities admitted by the separation of rules from code, and the ability to transfer rules in use to newcomers. We are able to support collaborative applications, for example,

where the participants create changes to the rules as they work. The model checking algorithms we use for analysis are linear in the size of the net plus the size of the formula being verified, so scaling the protocol size does not obviate the analysis advantage of our approach.

ACKNOWLEDGEMENTS

This work is supported in part by NSF grant #IRI-9732577.

REFERENCES

1. Beigl, M., Multi-Hop Routing in Infrared Networks. Submitted to Workshop on Computer Networks, Internet, and Multimedia. 1998 International Computer Symposium (ICS'98). Tainan, TAIWAN, R.O.C December 17-19, 1998. Online abstract: <http://www.teco.edu/~michael/publication/tw.html>
2. Bluetooth Special Interest Group. <http://www.bluetooth.com>
3. Clarke, E.M., E. A. Emerson, and A.P. Sistla, "Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications," *ACM Trans. On Programming Languages and Systems*, vol. 8 (2), April 1986, pp. 244-263.
4. Dewan, P., and Choudhary, R., Flexible user interface coupling in collaborative systems. *Proceedings of the ACM CHI'91 Conference* (April 1991), 41-49.
5. Dewan, P., and Choudhary, R., A high-level and flexible framework for implementing multi-user user interfaces. *ACM Transactions on Information Systems* 10, 4 (October 1992), 345-380.
6. Furuta, R. and P. D. Stotts, "Interpreted Collaboration Protocols and their use in Groupware Prototyping," *Proc. of the 1994 ACM Conference on Computer Supported Cooperative Work (CSCW '94)*, Research Triangle Park, NC, October 1994, pp. 121-131.
7. Furuta, R., and Stotts, P. D., Dynamic hyperdocuments: Replacing the programming metaphor. *Communications of the ACM, special issue on Hypermedia Design* (Aug. 1995), 111-112.
8. Greenberg, S. and Boyle, M., Moving Between Personal Devices and Public Displays. *Workshop on Handheld CSCW*, ACM Conference on Computer Supported Cooperative Work, November 14, 1998
9. The Infrared Data Association. <http://www.irda.org>
10. Landay, J., et al., NotePals: Sharing and Synchronizing Handwritten Notes With Multimedia Documents. *Workshop on Handheld CSCW*, ACM Conference on

- Computer Supported Cooperative Work, November 14, 1998
11. Myers, B., Collaboration Using Multiple PDAs Connected to a PC. *Workshop on Handheld CSCW*,
 13. Computer Supported Cooperative Work, November 14, 1998
 14. Shen, H., and Dewan, P., Access control for collaborative environments. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work* (November 1992), 51-58.
 15. Stotts, P.D., and Furuta, R., Petri-net-based hypertext: Document structure with browsing semantics. *ACM Transactions on Information Systems* 7, 1(Jan. 1989), 3-29.
 12. Schmidt, A., Lauff, M., Beigl, M., Handheld CSCW. *Workshop on Handheld CSCW*, ACM Conference on
 16. Stotts, P.D., and Furuta, R., Dynamic adaptation of hypertext structure. In *Proceedings of Hypertext 91* (Dec 1991), ACM, 219-231.
 17. Stotts, P.D., R. Furuta, and C. Ruiz Cabarrus, "Hyperdocuments as Automata: Verification of Trace-based Browsing Properties by Model Checking," *ACM Trans. on Information Systems*, vol. 16, no. 1, January 1998, pp. 1-30.
 18. Stotts, P. D., et al. CobWeb: Tailorable analyzable rules for collaborative web use. Tech Report TR99-404, Dept. of Computer Science, Univ. of North Carolina at Chapel Hill, May 1999.