CobWeb: Tailorable, Analyzable Rules for Collaborative Web Use

David Stotts, Jan Prins, Lars Nyland, Tianli Fan

Contact author:

David Stotts
CB 3175
Dept. of Computer Science
Univ. of North Carolina
Chapel Hill, NC 27599-3175
USA

ph: (919) 962-1833
fax: (919) 962-1799
email: stotts@cs.unc.edu

ABSTRACT

CobWeb is a collaborative web browsing system that allows the rules governing the interactions of multiple users (the *collaboration protocol*) to be externally specified and dynamically changed. We explain the architecture of the CobWeb implementation, and conclude by showing how the Java classes defining collaboration protocols are generated from visual formal specifications.

We note also that, though CobWeb has been used as an experimental platform for group Web browsing, the notion of dynamically loaded protocols can also be used to create novel single-user browsing behaviors on top of Netscape's Navigator (or any other Java-supporting browser).

CobWeb was developed as part of the DARPA CAETI program (Computer Aided Education and Training Initiative) and was supported through NRad grant N66001-95-C-8615. Information about the CAETI initiative, and on CobWeb specifically, can be found on the Web at

`http://www.cs.unc.edu/~stotts/cobweb/`
`http://ito.darpa.mil/research/caeti/CAETI/`

KEYWORDS

browsing semantics, group browsing, collaboration, programmable navigation

CobWeb: Tailorable, Analyzable Rules for Collaborative Web Use

David Stotts, Jan Prins, Lars Nyland, Tianli Fan
Department of Computer Science
University of North Carolina
Chapel Hill, NC 27599-3175
+1 919 962 1833, stotts@cs.unc.edu

ABSTRACT

CobWeb is a collaborative web browsing system that allows the rules governing the interactions of multiple users (the *collaboration protocol*) to be externally specified and dynamically changed. We explain the architecture of the CobWeb implementation, and conclude by showing how the Java classes defining collaboration protocols are generated from visual formal specifications.

We note also that, though CobWeb has been used as an experimental platform for group Web browsing, the notion of dynamically loaded protocols can also be used to create novel single-user browsing behaviors on top of Netscape's Navigator (or any other Java-supporting browser).

This project is part of the DARPA CAETI program (Computer Aided Education and Training Initiative) and was supported through NRad grant N66001-95-C-8615. Information of the CAETI initiative, and on CobWeb specifically, can be found on the Web at

```
http://www.cs.unc.edu/~stotts/cobweb/
http://ito.darpa.mil/research/caeti/CAETI/
```

keywords

browsing semantics, group browsing, collaboration, programmable navigation

INTRODUCTION

The CobWeb project extends two basic (though related) areas of hypermedia and Web research: collaborative Web browser technology, and hyperdocument browsing semantics in general. CobWeb is a web browsing system that allows Web users to gather as a group and explore links under control of different interactions rules. We refer to a particular set of group interaction rules as a *collaboration protocol*. CobWeb supports several different collaboration protocols, and allows new ones to be created and added to its functionality without recompiling CobWeb. A CobWeb collaboration protocol can even be modified or replaced during its use.

One of the main differences in CobWeb and other approaches to this problem is our use of a formal (hence analyzable) notation for collaboration protocols; we discuss our analysis methods in a later section. Another difference is the dynamic nature of protocol replacement; this requires Java and dynamic class loading, so may not be possible in non-interpreted implementation languages.

A collaboration protocol, regardless of the notation used to express it, must capture several aspects of Web documents and how users can interact with them:

- 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 Web pages (URLs) are involved in the tasks?
- What concurrent threads can exist in overall group activity?
- At which pages (URLs) must the threads be synchronized?
- What timing constraints exist on activity durations?

This is just our starting list; we expect to discover more characteristics as our work with CobWeb progresses.

Collaboration protocols should be unambiguous and formal. Informal notations may 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. This is an issue of implementing, or enforcing, the rules. We see it as orthogonal to specification of the rules themselves.

We want to develop a protocol specification and implementation 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. We see a browser like CobWeb as furthering this goal. Group use of CobWeb will identify areas where the rules of group interaction are too constrained, or perhaps too unconstrained. New rules can be developed easily and incorporated into the system, changing its behavior. This can be done by *users* and does not require rewriting CobWeb itself.

In summary, our research is developing technology for:

- isolating the collaboration protocol as a modular element of the CobWeb system architecture
- dynamically loading and replacing a protocol during CobWeb browsing
- creation and verification of collaboration protocols for CobWeb

RELATED WORK

Collaborative browsing of Web structures has been investigated previously in several projects. Notable ones include the Sociable Web from MIT Media Lab [6], ChibaMOO from Sensamedia [7], and Virtual Places, which is available commercially (http:// www.vplaces.com/).

EIT (Enterprise Integration Technologies) was created as a company to develop software to support corporate-level endeavors; before being bought by Verifone several years ago, they produced a suite of collaboration support software. It included the SHARE system [12], which allowed a group of designers to use the Web to construct designs; it also included the more recent Ewgie (Easy Web Group Interaction Enabler, at http:// www. eit. com/ ewgie/), which allows people to see where others are browsing on the Web. You can choose to follow people in Ewgie; if the person you're following types a URL, you will be taken there as well.

In each of these projects, the manner in which the group of browsers may interact is fixed and defined in the immutable code of the application. For example, Ewgie (and most others) offers what is commonly called *shoulder surfing*, in which a leader browses the Web and all other group members follow the leader, "looking over his shoulder" so to speak. CobWeb differs from these early approaches in that the collaboration protocol is not fixed. The system will dynamically load new protocols, allowing users to create their own interaction rules.

If we broaden our interest to non-Web software, Dewan's Suite system [5, 4, 14] allows flexible rules governing the interactions multiple users have in a collaborative software system. The rules in Suite, however, are essentially code that is integrated throughout the application. Suite is, then, a toolkit for more easily building collaborative applications, but does not separate the collaboration protocol from the rest of the code. To change rules, the application must be recompiled.

Kaplan developed ConversationBuilder [11] for constructing collaborative systems, but to also allow flexible collaboration rules. The collaboration protocol is separated out, and therefore more easily altered. The rules governing user interactions in a system built with ConversationBuilder can be changed without recompiling the system.

The main advantage of the CobWeb approach over these previous efforts is analyzability. As in ConversationBuilder, CobWeb protocols can be dynamically changed, even during a collaboration session; however, the notation in which we express the protocols is not arbitrary programming code. Rather, we use a formal computation model for which we can determine the correctness of important browsing properties. From this formal description, we then derive the Java classes that are dynamically loaded into CobWeb to implement the new behavior.

The concept of hyperdocument browsing semantics was defined and first investigated in the Trellis project [15, 16, 10]. In the various Trellis prototypes, browsing semantics was specified *within each document* by using the links to implicitly define a parallel automaton (specifically, a Petri net) that formally defined the multi-threaded "behavior" of the document.

Early uses for Trellis included controlling multiple threads of activity on a reader's screen while browsing, much like *frames* currently in HTML. This multi-threading was extended to include *synchronization* of different readers on multiple screens, i.e., collaborative browsing semantics [9]. We refer to the specification and synchronization of multiple user activity threads as a *collaboration protocol*.

In earlier projects we have migrated some of the results from Trellis research into use in the WWW. We first produced the MMM (Multi-head/Multi-tail Mosaic) prototype Web browser [2]. MMM was a modification of Mosaic, and an extension of HTML, to allow expression of multi-headed/multi-tailed links (as in Petri net arcs); like Trellis, then, MMM allowed expression of a rudimentary collaboration protocol implicitly in document links, but it worked for the Web. We built a later version of MMM was a proxy server for Netscape Navigator [1].

CobWeb takes this work further. We are experimenting with an approach in which the collaboration protocols are specified independently of document pages (instead of Trellis-like implicitly in the links). We have retained the Trellis notion that a collaboration protocol can be expressed formally as a colored Petri net. Figure 1 illustrates this concept with a colored net encoding the rules of a simple moderated meeting.

COBWEB SYSTEM ARCHITECTURE

CobWeb operates with Netscape Navigator, using it Java applet capabilities. A CobWeb collaboration server (Java application) also has to be installed and executed on

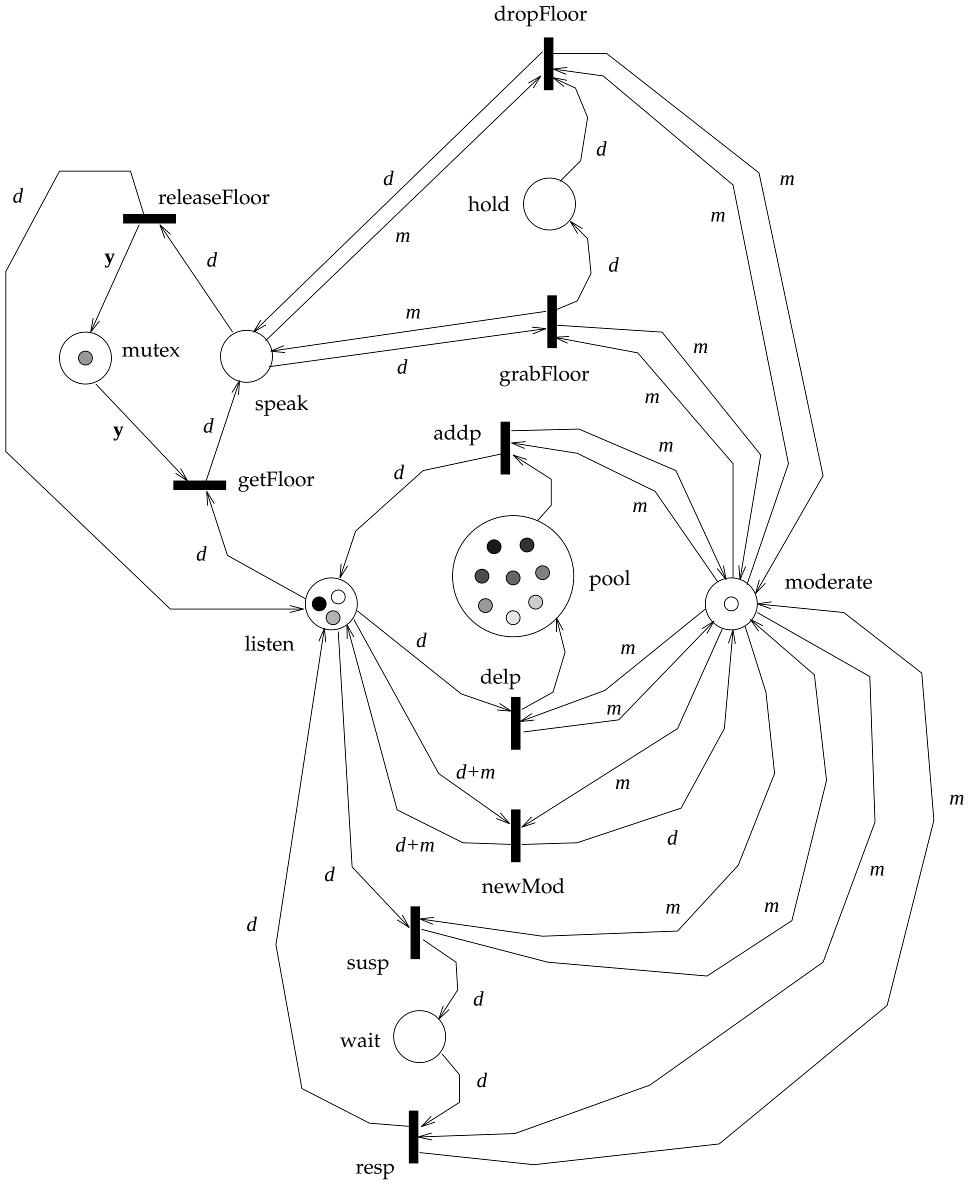

Figure 1: Moderated meeting collaboration protocol

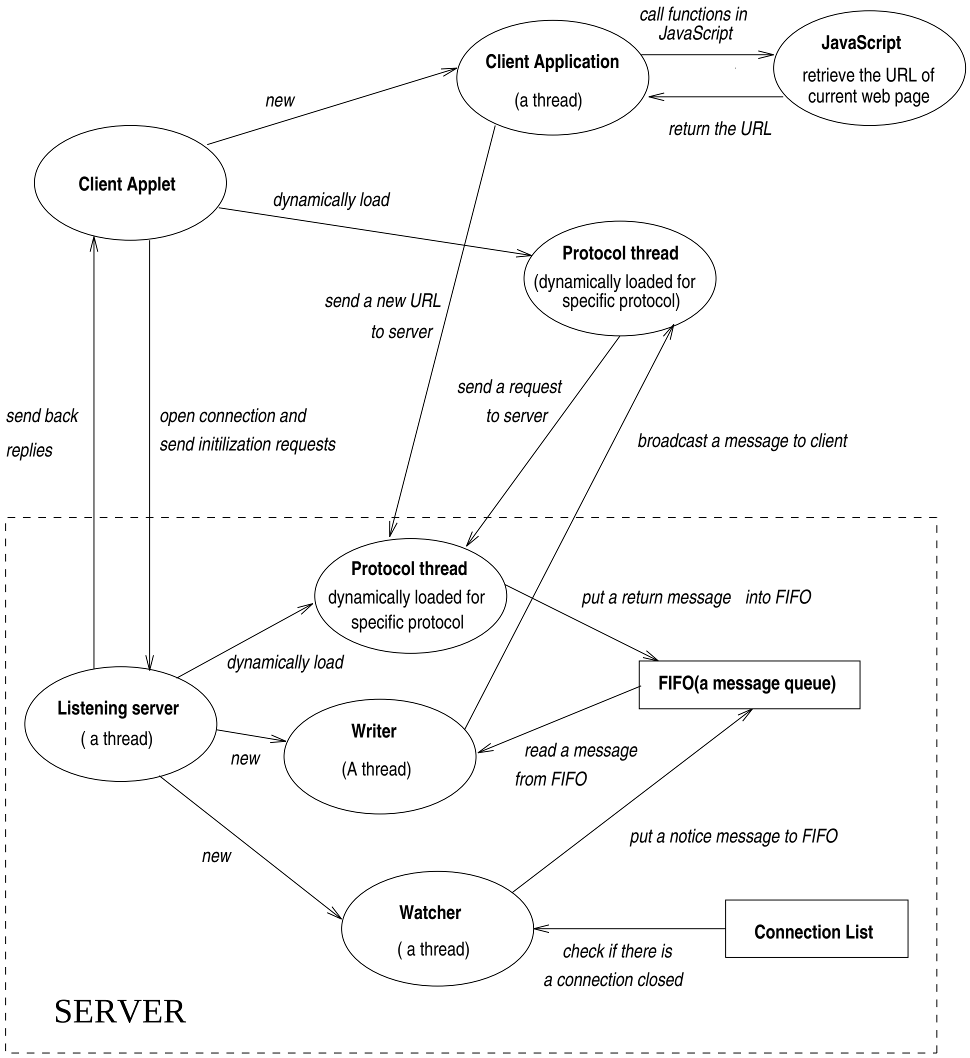

Figure 2: Architecture of the CobWeb system.

a Unix host to manage group membership. Since the Netscape browser is widely available at little or no cost, and is used all over the world, anyone can therefore easily attend a CobWeb session from anywhere in the world. Users also enjoy all functionality developed by Netscape; future enhancements to Navigator are automatically incorporated into CobWeb without requiring us to respond or accommodate.

Figure 2 shows the major components of the CobWeb architecture.

Java Applets/JavaScript

CobWeb is realized with a combination of Java applications, applets and JavaScript. JavaScript retrieves the current URL of each page. Communications between an individual workstation and the server is done through applets. An applet can only communicate with the server machine from which it is loaded (the applet is specified within a HTML page), so we had to design a work-around for this limitation.

CobWeb comprises a Java application running as a "collaboration" server, a Java applet that is downloaded by a person who wishes to join a CobWeb session, and a JavaScript component of the Web page that is loaded for CobWeb itself. The server application tracks group membership (there can be numerous different groups active at any time) and also enforces the rules of whatever collaboration protocol each group is using.

Figure 3 shows the main screen of CobWeb. It is divided into two frames; the Web page of interest is in the left frame, and a "chat room" is in the right frame. When a user executes CobWeb, he selects the protocol under which he wishes to operate (under the chat window). In the current prototype, this also selects the group to join, as we run only one group per protocol. This is an arbitrary limitation, and not fundamental to the CobWeb concept.

The selected protocol, a Java class, is dynamically loaded and new buttons are then displayed under the chat window. The buttons loaded represent the actions allowed by the protocol. The user will be asked to register a name with the server, and then a note will be posted to the chat window informing others in the group that a new user has joined.

The user might then browse on the Web page in the left window, contribute notes to the chat window, or execute protocol functions by clicking the buttons below the chat window. The user will be allowed to do these things, or prohibited, at different times, depending on the structure of the collaboration protocol. For example, one protocol might not allow a reader to leave the page displayed on the left, allowing only the leader to do so. Another protocol might allow the leader to "drag" readers with him/her, but then allow readers to leave the page of interest if they wish to. A protocol might allow only the leader to contribute to the chat window, or it might allow free access to chat at all times by all users; it might prescribe a voting mode in which each user can contribute, but only once. A protocol might not even have a notion of leader, having a mechanism in which no one follows anyone else unless they specifically request to go where another is.

The amount and manner of access to CobWeb resources is entirely dependent on the actions defined and controlled by the collaboration protocols, implemented in CobWeb as dynamically loaded Java classes with a specific structure and interface. New collaboration protocols can be created for CobWeb by instantiating the class methods with different implementations.

Client Side

The applet that runs on the client-side machine is *chatclient*. The class *chatclient* is derived from the class *Applet* and performs the following actions:

- Establishes the connection with the server
- Creates a user interface (input fields, output areas, protocol choice box and three buttons which are "Reload", "Back" and "Forward"
- Instantiates a *pagectl* object that retrieves each attendee's current URL and displays it in the address field, and if required, sends it to the server.
- According to the user's request, dynamically loads the Java class which describes a selected protocol on client side.
- Instantiates a *UserInfoFrame* object that pops up a window frame through which attendees can enter user ID at the server registration phase.

Class *pagectl* and the Java class used to describe the specified protocol are both derived from class *Thread*. As the result, they can run concurrently with the applet.

Server Side

The Java program which runs on the server machine is *chatserver*. The class *chatserver* is derived from the class *Thread* and performs the following:

- Creates a window frame which keeps displaying the latest group membership list for the collaboration.
- Instantiates a *ServerWriter* object that broadcasts the new URL fetched by the leader as well as the inputs made on the shared whiteboard by special attendees.
- Instantiates a *ConnectionWatch* object that keeps watching the connections between the server and all attendees. If a connection from an attendee is broken, the attendee's name is removed from the list. This event is broadcast to all remaining attendees.
- Establishes a connection with each attendee. According to each client's required protocol, the server dynamically loads a Java class which describes the specified protocol on the server side and uses this class to keep communication with the attendee.

Classes *chatserver, ConectionWatcher, ServerWriter* and the dynamically loaded protocol class are all derived from class *Thread*.

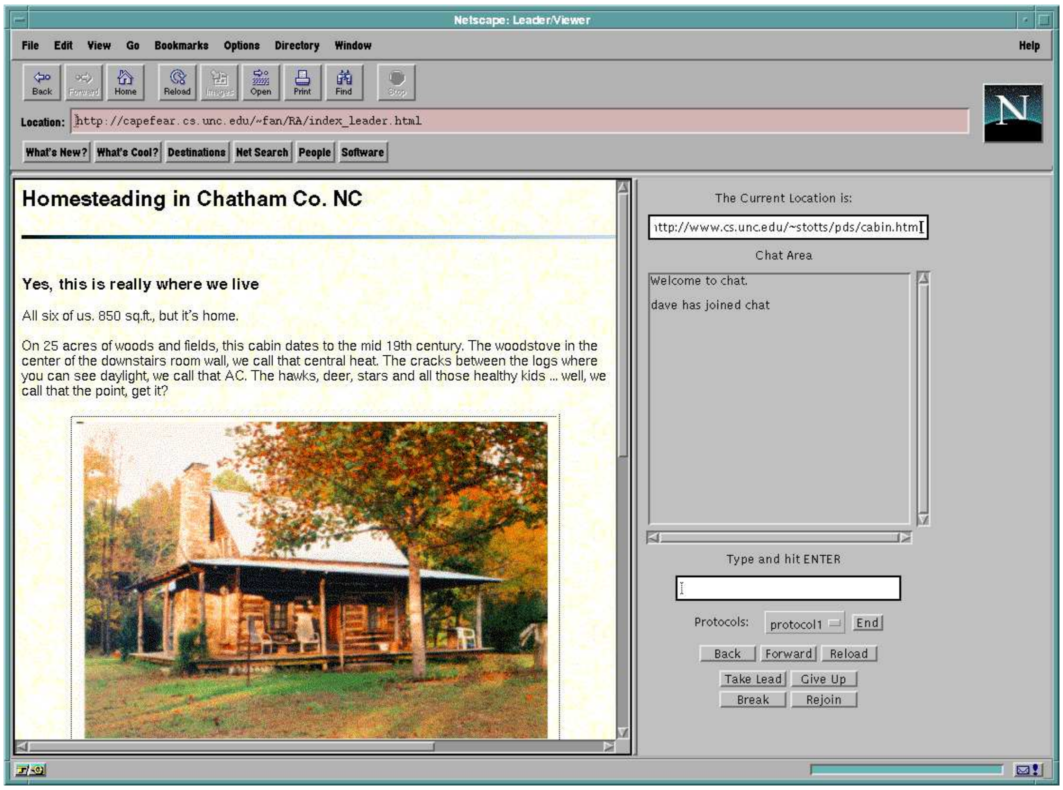

Figure 3: Screen from the CobWeb browser.

Encoding a Protocol in Java

A browsing protocol defines a user's viewing behavior. The system can support multiple protocols and new protocols can be added easily.

The primary server and client are general and protocol independent. As the result, the application programmer only need to write protocol-specific code to add a new protocol. The system can dynamically load the new protocol components to the server and to the client. Thus a user can select the protocol to use on-the-fly.

We need to implement two Java classes to describe a protocol. One is for the client and the other for server. The application programmer should define protocols based on these following rules.

Client Side

We define an interface *protocol.java*. It contains one abstract method *setParams(chatclient a)*. Each Java class in the protocol "extends Panel implements Runnable".

There are three key methods in each class:

• setParams(chatclient a): In this method, you need to initialize all the variables and add the necessary graphical user interface components.
• handleEvent(Event evt): This method handles all the events created by the components added in the method setParams(chatclient a).
• run( ): In this method, the client and server communicate with each other; the client receives and processes commands from the server.

Besides these three key methods, you can write more methods as necessary.

We have already written two Java classes: *chatclient.java* and *pagectl.java*. There are two variables *imgSend* and *wordSend* in *chatclient.java*. You may update them in your new protocol Java class. If you want to send the URL of the page which you are browsing to the server, set the value of *imgSend* as true (default is false). If you want to send the words entered through the input text field to the server, set the value of *wordSend* as true (default value is false).

Example protocol code is available on line at the URL given in the abstract.

Server Side

We define an interface *connection.java*. It contains one abstract method *setParams(Socket s, ConnectionWatcher w,ServerWriter sw, String username, String protocol)*. Each Java class "extends Thread implements connection".

There are two key methods in each class:

• setParams(Socket s, ConnectionWatcher w,ServerWriter sw, String username,String protocol): In this method, you need to initialize all the variables.
• run( ): In this method, server and client communi-

cate with each other. The server provides service to the client's requests.

Besides these two key methods, you can writer more methods as necessary.

We have already written three Java classes: *chatserver.java*, *ServerWriter.java*, and *ConnectonWatcher.java*. If you want to broadcast a message or a URL to all the clients, call method *outdata.push(String protocol, String msg)* of class *ServerWriter*. We wrote thirteen useful methods in *ServerWriter.java* to use in new Java protocols; for example, to update an attendee's status.

Details can be seen in the code for *ServerWriter.java*, which is available on line at the URL given in the abstract.

Example collaboration protocols

Our current collaboration protocols are written manually as Java classes. We have produced a shoulder surfing protocol with chalk passing (Web Lecture) and a moderated meeting protocol (Web Conference).

Web Lecture

Protocol 1 defines a Web Lecture. It currently provides four basic functions. When you select the "protocol 1" from the protocol choice box, four buttons will be displayed on the bottom of the user interface.

- Take Lead: by clicking this button, if no one is leading now, you will become the leader; when you first enter the system, you are by default a viewer.
- Give Up: if you don't want to lead any more.
- Break: if you want to browse a page by yourself a while and don't want to be interrupted by the leader's movements, click this button; you are still able to communicate with other attendees in the chat window.
- Rejoin: if you want to come back and the follow the leader.

Web Conference

Protocol 2 defines a Web Conference, which is very nearly the moderated meeting illustrated as a colored net in figure 1. When you select "protocol 2" from the protocol choice box, a button, "Moderate" will appear on the bottom of the user interface. If you want to be the moderator, click this button. If no one is moderator, you will be given the job. Then, five extra buttons will appear for the functions of the moderator:

- Add Person: by clicking this button, the moderator can add a person from the pool (outside the conference) to listening status. Everyone is in the pool by default when they first enter the system.
- Delete Person: by clicking this button, the moderator can delete a person who is listening and return them to the pool.
- Grab Floor: if the moderator wants to speak, he will click this button. If someone else is speaking, he is put on hold and so the moderator can have the floor. Having the floor means you are allowed to post to the chat window. Only the current speaker can do so.
- Drop Floor: when the moderator finishes speaking, click this button. If a previous speaker is on hold, he will be returned to the floor to continue speaking.
- Swap Mod: the moderator and a listener can swap jobs with each other. The moderator will become a listener and the listener will be the moderator.

When a person in the pool is added to the listening group (but not as moderator), four new function buttons will appear on his panel:

- Get Floor: for a listener to be allowed to speak; If no one else is currently speaking, the listener is allowed to post to the chat window.
- Release Floor: when the current speaker is finished and wishes to return to listening status.
- Suspend: the listener wants a break; during the rest, he cannot get any information from the meeting.
- Rejoin: the resting listener becomes an active listener in the meeting again; posting to the chat window are now visible again.

ANALYSIS OF COBWEB PROTOCOLS

We are developing a modification of the Trellis colored net editor *xTed* to have the Java code for the protocol classes generated automatically from formal specifications (colored Petri nets). When complete, a user can encode the various aspects of a protocol outlined as annotations on the synchronization structure of a colored net. Figure 4 shows this net editor on the moderated meeting protocol from figure 1.

When a collaboration protocol is expressed as a colored net, we can analyze it for correct browsing properties using a model checking approach, first developed by Clarke [3] for analysis of algorithms in concurrent programming languages. We first applied these methods to analysis of hypermedia documents in the original Trellis model [17], encoded as basic nets (not colored nets). Current Ph.D. research is extending our early analysis to colored nets and full collaboration protocols [13].

Properties that a CobWeb collaboration protocol should exhibit are encoded in a temporal logic for verification. Without going into detail on the syntax of temporal logics or model checking, we will briefly illustrate the analysis procedure by example. Considering the moderated meeting protocol in figure 1, we might wish these properties to be exhibited:

Informal: It is impossible for a speaker to get put on hold and then left there

Formal: A G (P.hold ==> A F (*not* P.hold))

Translation: on all execution paths it is always the case that if a state is reached where a speaker in on hold, then on all paths from that point eventually a state will reached where the speaker is not on hold

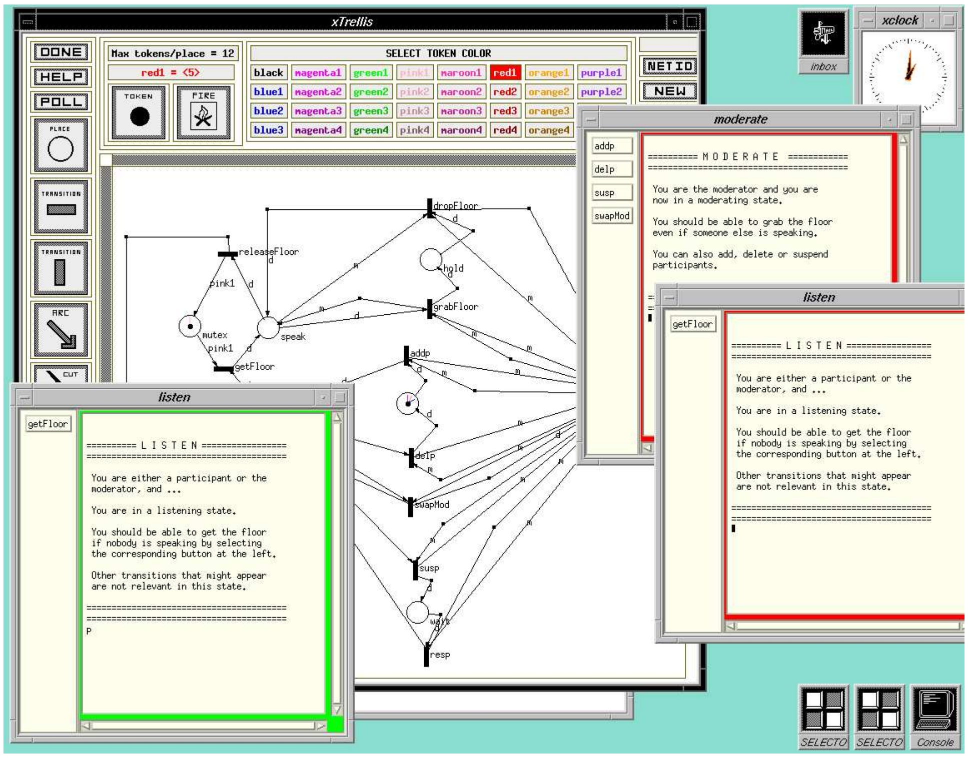

Figure 4: xTed colored net editor

Informal: Only one speaker at a time is allowed to have the floor

Formal: not E F (P.speak *and* T.getFloor)

Translation: There is no execution path where a speaker has the floor and the "get floor" operation can be accomplished by a listener.

Using the knowledge of what basic properties are true at each location in the net (such as "P.hold" meaning a token is in the "hold" place, or "T.getFloor" meaning the "getFloor" operation can be executed), the model checker can determine the truth of the temporal logic assertions over sequences of states (i.e., over program executions). Once a protocol net is verified, "correct" Java code can be derived from it to create a new CobWeb group interaction style.

In our current prototype, any and all of the aspects of Web collaboration mentioned above can be encoded only if you construct the Java protocol classes manually. In fact, with manual encoding, any detail a protocol author can envision can be defined and manipulated in the Java source. For automation, we have a more limited capability in the current implementation. The goal of our work with xTed is to move as many of these aspects as possible into the formal notation of annotated nets. Some are easy: multiple activity threads and synchronizations, for example, are exactly what the net arcs express; colors on tokens allow creation and distinguishment of categories of users and tasks; annotations on tokens can express user locations (by IP address, for example); annotations on the places (circles) can express URLs if specific pages are needed at some point in collaboration; annotations on transitions (bar) can express timing constraints that might apply to actions. Other aspects are harder to formalize: group goals, individual goals, resources required. These have not yet been worked into our editor prototype.

CONCLUSIONS

Collaboration protocols are the rules under which a group of people interact in a groupware application. Group Web browsing is currently done by having one of several well-know collaboration protocols (e.g., shoulder surfing or classroom lecture) embedded in the code of a browsing application.

CobWeb is more flexible than current group Web browsing approaches, as the collaboration protocols it uses are modular parts of its architecture and can be replaced. The group behavior enforced by CobWeb, then, is inherently extensible. This is done in the current prototype by manually writing Java classes to express the desired interactions, then dynamically loading them into CobWeb.

We are currently working to formalize the specification and analysis of collaboration protocols so that new CobWeb behaviors can be designed and produced by non-programmers, i.e., without having to manually write

Java classes.

Note that the CobWeb technology can complement or augment other tools for structuring Web content, such as *Walden's Paths* [8] which allows related Web pages to be linked in sequence and presents sequences as a navigable structures. Since CobWeb does not require any specific Web pages and does not require any alteration to page contents, it can be used to gather a group for collective use of a tool like *Walden's Paths*.

Finally, we should note that this paper, and most of our CobWeb experimentation, has been concerned with *multi-user* Web access. However, nothing in CobWeb requires that a dynamically loaded protocol must involve a group. We can as easily write protocols that will prescribe novel or complicated browsing behaviors that are to be navigated by a single user; it would not be, in such a case, a *collaboration* protocol, but certainly it is a way to create, for example, interesting navigation activities for literary Web documents. An author can create novel literary experiences for hyperdocument readers by creating content *in conjunction with* controlling protocols for presentation, access, timing, etc. In essence, CobWeb can be programmed *by individual authors* to simulate single-user behaviors that are considered interesting in other systems.

REFERENCES

1. Capps, M., Ladd, B., and Stotts, D. Enhanced graph models in the web: Multi-client, multi-head, multi-tail browsing. In *Proceedings of the 5th WWW Conference (Computer Networks and ISDN Systems, vol. 28)* (May 6-10 1996), pp. 1105–1112. This paper appears on-line at www5conf.inria.fr/fich_html/papers/P19/.
2. Capps, M., Ladd, B., Stotts, D., and Furuta, R. Multi-head/multi-tail mosaic: Adding parallel automata semantics to the web. In *Proceedings of the 4th WWW Conference (WWW Journal, O'Reilly and Associates Inc., vol. 1)* (Dec. 11-14 1995), pp. 433–440. This paper appears on-line at www.w3.org/pub/Conferences/WWW4/Papers/118/.
3. Clarke, E. M., Emerson, E. A., and Sistla, A. P. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems 8*, 2 (1986), 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. Donath, J. S., and Robertson, N. The sociable web. Technical report, MIT Media Lab, 1994. http://big-sleep.media.mit.edu/ SocialWeb/SociableWeb.html.
7. Epstein, S. L. *Collaborative Hyperarchical Integrated Media Environments.* Sensemedia Publishing, 1995.
8. Furuta, R., Shipman, F. M., Marshall, C. C., Brenner, D., and Hsieh, H.-W. Hypertext paths and the World-Wide Web: Experiences with Walden's Paths. In *Proc. of ACM Hypertext 97* (Apr. 1997), ACM, pp. 167–176.
9. Furuta, R., and Stotts, P. D. Interpreted collaboration protocols and their use in groupware prototyping. In *Proc. of the 1994 ACM Conference on Computer Supported Cooperative Work (CSCW '94)* (Oct. 1994), ACM, New York, pp. 121–131.
10. 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.
11. Kaplan, S. M., Tolone, W. J., Bogia, D. P., and Bignoli, C. Flexible, active support for collaborative work with conversationbuilder. In *Proc. of ACM Conference on Computer Supported Cooperative Work (CSCW 92)* (Oct. 1992), pp. 378–385.
12. Kumar, V., Glicksman, J., and Kramer, G. A SHAREd web to support design teams. In *Proc. of the WETICE '94* (Apr. 1994), pp. 178–182. Morgantown, WV.
13. Navon, J., Stotts, D., and Furuta, R. Subdocument invocation modes in collaborative hyperdocuments. *Computers in Industry 29* (1996), 91–104.
14. Shen, H., and Dewan, P. Access control for collaborative environments. In *Proc. of the ACM Conference on Computer Supported Cooperative Work* (Nov. 1992), pp. 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.
16. Stotts, P. D., and Furuta, R. Dynamic adaptation of hypertext structure. In *Proceedings of Hypertext 91* (Dec. 1991), ACM, pp. 219–231.
17. Stotts, P. D., Furuta, R., and Ruiz, J. C. Hyperdocuments as automata: Verification of trace-based browsing properties by model checking. *ACM Trans. on Information Systems 16*, 1 (Jan. 1998), 1–30.