Technical Report TR02-008

Department of Computer Science
Univ of North Carolina at Chapel Hill

CobWeb: Visual Design of Collaboration Protocols for Dynamic Group Web Browsing

David Stotts, Jan Prins, Lars Nyland

Dept of Computer Science
Univ. of North Carolina at Chapel Hill
Chapel Hill, NC 27599-3175

stotts@cs.unc.edu

March 1, 2002

CobWeb: Visual Design of Collaboration Protocols for Dynamic Group Web Browsing

David Stotts, Jan Prins, Lars Nyland
Dept of Computer Science
Univ. of North Carolina at Chapel Hill
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).

1. Introduction

The CobWeb system is a collaborative Web browser. While this is not novel of itself, our approach to describing the collaboration rules governing the interactions of users – *the collaboration protocol*, is. Our approach to protocols is adaptable to all forms of collaborative software; we present CobWeb specifically as one example.

The advantage of CobWeb is formal analysis and modularity of software. The rules governing group member interactions are critical aspects of collaborative software systems. The current norm is to express those rules directly in the source code, intermingled among the rest of the software functionality. We take the position that the architecture of a collaborative system is better structured by having the rules separated, easily identified, and expressed in a manner amenable to analysis.

Our approach is to develop protocols around a Trellis hypermedia model. Trellis models are annotated, colored P/T nets and express concurrency and synchronization among the components of the collaboration. We develop CobWeb protocols with a graphical editor (*xTed*), using a visual notation appropriate to the formalism. Once developed, we can analyze them with model checking for logical flaws, and then generate the Java code to implement them for CobWeb directly from the editor.

In summary, our research is developing technology for:

- isolating collaboration protocols as modular elements of the software architecture of collaborative systems
- dynamically loading and replacing a protocol during system use (browsing in the case of CobWeb)
- graphical creation and model-checking verification of collaboration protocols

We present CobWeb, a collaborative web browser, as an example collaborative system employing our results.

In the remainder of this paper we first discuss other approaches to collaborative system rules. We then give a brief summary collaboration protocols and the Trellis model on which we build them. We follow this with a description of the CobWeb system architecture and how we encode a protocol in Java. We also discuss the xTed editor for creating protocols, and the process of generating Java implementations from it. We conclude with examples of specific protocols developed for CobWeb.

2. Related work

The CobWeb project extends two basic (though related) areas of hypermedia and Web research: collaborative Web browser technology, and hyperdocument browsing semantics in general.

Collaborative browsing of Web structures has been investigated previously in many projects. Notable ones include the Sociable Web from MIT Media Lab [3], Chiba-MOO from Sensamedia [4], 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 [10], 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.

Kaplan developed ConversationBuilder [9] to allow flexible rules. The rules are flexible, but are not abstracted well from the body of code defining the application. Hence the rules are not analyzable, being effectively arbitrary program fragments. Our approach differs in providing this flexibility by making the rules defined by an analyzable model that is not an arbitrary Turing machine.

3. Collaboration protocols

A collaboration protocol, however it is expressed, 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. Stotts and Furuta have developed a method for verifying the browsing properties allowed by the structure of a hypermedia document [15], including documents with structure specified by Petri nets. Current work is extending this analysis to colored net models that encode collaboration protocols [11].

Collaboration protocols must however be flexible, not unuseably 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.

Trellis hypermedia, collaborations, and CobWeb

The concept of hyperdocument browsing semantics was defined and first investigated in the Trellis project [12,6,13,14,7,8]. 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 [7]. 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.

4. Automating creation of CobWeb protocols

Our first collaboration protocols were written manually as Java classes. We have produced a shoulder surfing protocol with chalk passing, a classroom lecture protocol (shoulder surfing without leader changes), and a moderated meeting protocol. Some of these are explained in detail in following sections.

Once we understood the details of interactions needed for the CobWeb application specifically, we automated the creation of protocols for it and re-architected the system to allow protocols to be dynamically loaded, even during an existing group session.

To do this we developed a modification of the Trellis colored net editor *xTed* to have the Java code for the protocol classes generated automatically from formal specifications (colored P/T nets). When the net is complete, a user can encode the various aspects of a protocol as annotations on this synchronization structure. After development of the annotated net, verification is done and the Java code implementing the protocol is generated. Figure 2 shows this net editor displaying the moderated meeting protocol from Figure 1.

Protocol design proceeds as follows. The P/T net structure needed to express the concurrent threads of activity in the collaboration is drawn in xTed. Appropriate information is added to the net components as annotations. Some of these annotations are hypermedia pages (shown in Figure 2 as popup windows), explaining the system functionality that will be active at the various points in the net as the collaboration progresses.

Verification of the protocol is then both a formal process and an informal process, done hand-in-hand. Informally, the Trellis hypermedia tool can directly read the P/T net and its document components and present the functionality of the system for "browsing". This browsing simulates with synchronized multiple windows the actions of the final system during execution. Formally, the P/T net is exported to the input format needed for model checking [15,16]. Model checking, developed by Clarke [17,18], is an automated technique for determining if a finite state system had desired properties (like deadlock freedom, correct component relationships in time, etc.). The properties are expressed in a temporal logic [19], and the model checker tells which are upheld and which are violated. Though we are currently using Clarke's symbolic checker, there are other approaches to model checking; see [20] for example.

If informal browsing or formal model checking indicate flaws in the protocol, then xTed is used to continue the design process. Eventually the designer is satisfied that the CobWeb protocol is complete and correct. At this point the P/T net and it's annotations are used by xTed to generate appropriate Java classes to fit into the CobWeb architecture in the protocol locations. CobWeb will then behave during web browsing as the P/T net protocol indicates. The various protocols available to CobWeb are selectable from a pull-down menu during execution. Figure 3 shows the CobWeb window during browsing; the buttons in the lower right window are the names (and actions) of the transitions in the P/T net for the active protocol. The protocol is selected from the list above these buttons (*protocol1* is active in this example).

Users will be allowed or restricted from the floor, one or more at a time. Group sizes are set. Users are allowed or restricted from browsing away from the group. Decisions are made as to who is forced to follow the moderator when the moderator browses. Use of the chat tool is established; users are either allowed to chat, or only the floor holder (surf leader) is allowed to chat. These are some of the aspects of the CobWeb protocol that are encoded into the generated Java classes.

The goal of our work with the *xTed* graphical editor is to move as many of the previously enumerated aspects of collaboration protocols 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 display of categories of users and tasks; annotations on tokens express user locations (by IP address, for example); annotations on the places (circles) express URLs if specific pages are needed at some point in collaboration; annotations on transitions (bar) 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. If needed, they must be hand coded as additions to the Java framework generated from xTed.

5. CobWeb System Architecture

CobWeb operates with Netscape Navigator. No modified browser is required, though a server does have to be installed and executed on some host to manage group membership. Anyone user can easily attend a CobWeb session from anywhere in the world. Figure 4 shows the main components of the CobWeb system 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 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. It 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 dynamically loaded Java protocols.

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

- Creates a window frame that 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 that 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*.

6. 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 needs 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 that 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 is false). Example protocol code is available for inspection online at http://www.cs.unc.edu/~stotts/cobweb/doc/index.html

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 communicate 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 can be seen online at the URL http://www.cs.unc.edu/~stotts/cobweb/doc/index.html.

7. Example collaboration protocols

So far, the system supports several predefined protocols. We describe two commonly understood ones here: Web Lecture, and 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 close to 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.

8. Conclusions

Collaboration protocols are the rules under which a group of people interacts 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 using the xTed graphical editor to produce new Java classes expressing the desired interactions, and then dynamically loading them into CobWeb.

Note that the CobWeb technology can complement or augment other tools for structuring Web content, such as *Walden's Paths* [5]; this tool allows related Web pages to be linked in sequence and presents sequences as 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 structuring 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 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.

Acknowledgements

This project was supported in part of the DARPA CAETI program (Computer Aided Education and Training Initiative) and was supported through NRad grant N66001-95-C-8615, and by the National Science Foundation under grant 9732577.

References

[1] M. Capps, B. Ladd, D. Stotts, "Enhanced Graph Models in the Web: Multi-client, Multi-head, Multi-tail Browsing," Computer Networks and ISDN Systems, vol. 28 (Proc. of the 5th WWW Conf., May 6-10, 1996, Paris), pp. 1105-1112. This paper is available on-line at *http://www5conf.inria.fr/fich_html/papers/P19/Overview.html*

[2] B. Ladd, M. Capps, D. Stotts, and R. Furuta, "Multi-head/Multi-tail Mosaic: Adding Parallel Automata Semantics to the Web," *World Wide Web Journal*, O'Reilly and Associates Inc., vol. 1 (*Proc. of the 4th International WWW Conference*, Boston, December 11-14, 1995), pp. 433-440. This paper appears on-line at *http://www.w3.org/pub/Conferences/WWW4/Papers/118/*

[3] J.S. Donath and N. Robertson, "The Sociable Web," Technical Report, MIT Media Lab, 1994. http://big-sleep.media.mit.edu/Social-Web/SociableWeb.html.

[4] S.L.Epstein,, *Collaborative Hyperarchical Integrated Media Environments,* Sensemedia Publishing, 1995.

[5] R. Furuta, F. Shipman, C.C. Marshall, D. Brenner, and H.-W. Hsieh, "Hypertext Paths and the World-Wide Web: Experiences with Walden's Paths," *In Proc. of ACM Hypertext '97* (Apr. 1997), ACM, pp. 167-176.

[6] R. Furuta and P. D. Stotts, "Programmable Browsing Semantics in Trellis," *Proc. of Hypertext '89 (ACM)*, Pittsburgh, November 1989, pp. 27-42.

[7] R. Furuta 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.

[8] R. Furuta and P. D. Stotts, "Dynamic Hyperdocuments: Replacing the Programming Metaphor," *Comm. of the ACM*, short communication in special issue on Hypermedia Design, Aug. 1995, pp. 111-112.

[9] S.M. Kaplan, W.J. Tolone, D.P. Bogia, and C. Bignoli, "Flexible active support for collaborative work with ConversationBuilder," in *Proc. of ACM Conf. On Computer Supported Cooperative Work* (CSCW '92, Oct. 1992), pp. 378-385.

[10] V. Kumar, J. Glickman, and G. Kramer, "A SHAREd web to support design teams," in *Proc. of WETICE '94* (Apr. 1994), Morgantown, W.Va., pp. 178-182.

[11] J. Navon, D. Stotts and R. Furuta, "Subdocument Invocation Modes in Collaborative Hyperdocuments," *Computers in Industry* (Elsevier), vol. 29 (1996), pp. 91-104.

[12] P. D. Stotts and R. Furuta, "Petri Net Based Hypertext: Document Structure with Browsing Semantics," *ACM Trans. on Information Systems* (ACM), vol. 7, no. 1, January 1989, pp. 3-29.

[13] P. D. Stotts and R. Furuta, "Temporal Hyperprogramming," *Journal of Visual Languages and Computing* (Academic Press), vol. 1, no. 3, September 1990, pp. 237-253.

[14] P. D. Stotts and R. Furuta, "Dynamic Adaptation of Hypertext Structure," *Proc. of Hypertext '91 (ACM)*, December 15-18, 1991, San Antonio, Texas, pp. 219-231.

[15] P. D. Stotts, 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.

[16] Stotts, P.D., and J. Navon, "Model Checking CobWeb Protocols for Verification of HTML Frames Behavior," *Proc. of WWW 2002*, Honolulu, Hawaii, May 2002. (to appear)

[17] J. R. Burch, E. M. Clarke, K. L. McMillan, "Symbolic Model Checking: 10^20 States and beyond," Information and Computation, vol. 98, pp. 142-170, 1992.

[18] E. M. Clarke, E. A. Emerson, A. P. Sistla, "Automatic verification of finite-state concurrent systems using temporal logic specifications," ACM Trans. on Programming Languages and Systems, vol. 8, pp. 244-263, 1986.

[19] A. Pnueli, "A temporal logic of concurrent programs," Theoretical Computer Science, pp. 45-60, 1981.

[20] "On-the-fly, LTL Model Checking with SPIN", Bell Labs Research, 2001. http://netlib.bell-labs.com/netlib/spin/whatispin.html

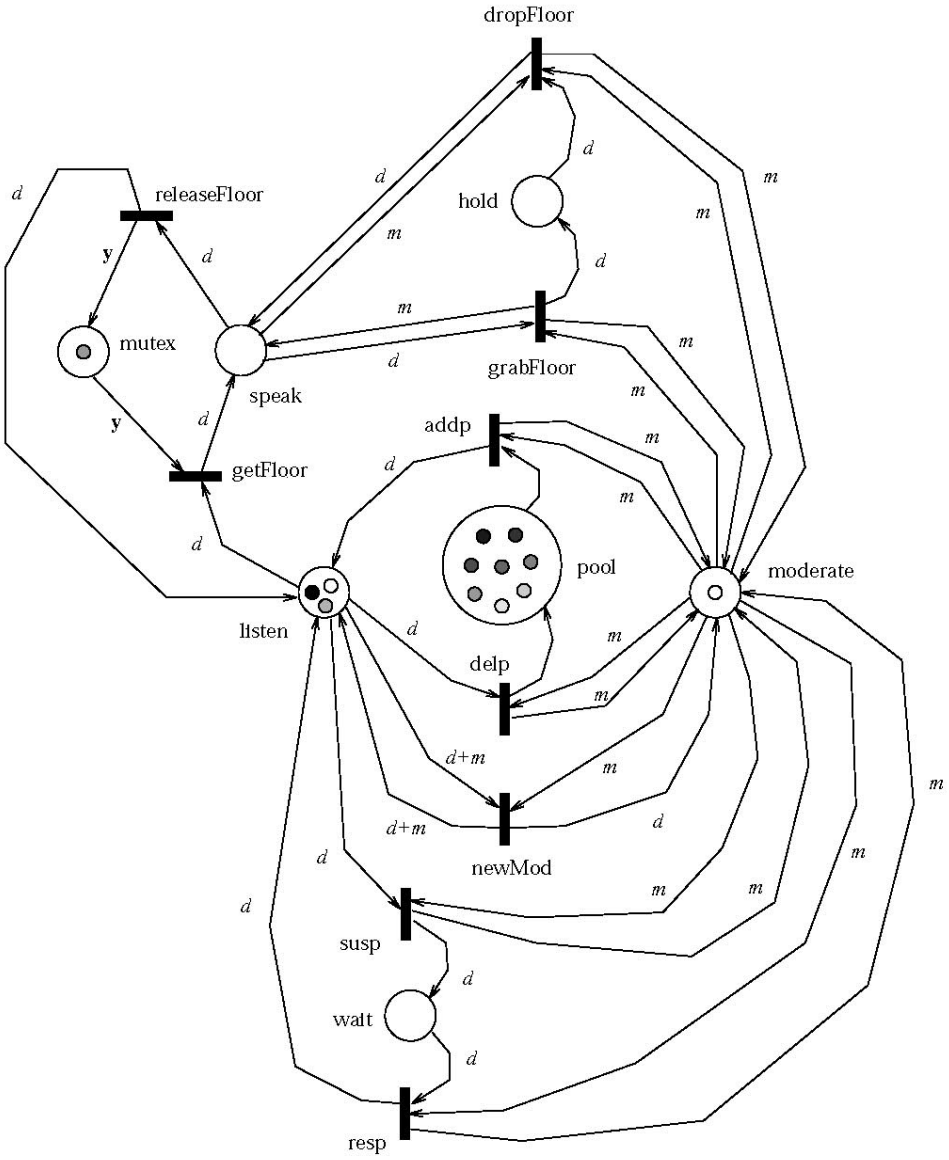

Figure 1. Moderated meeting collaboration protocol

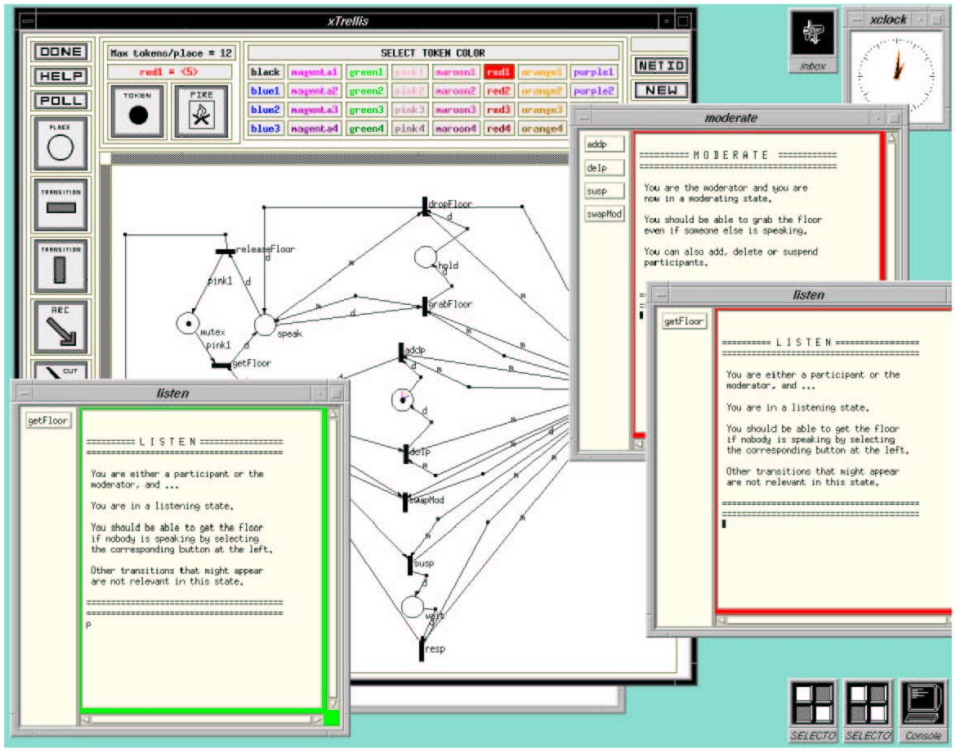

Figure 2. Sample xTed screen, showing graphical protocol editor

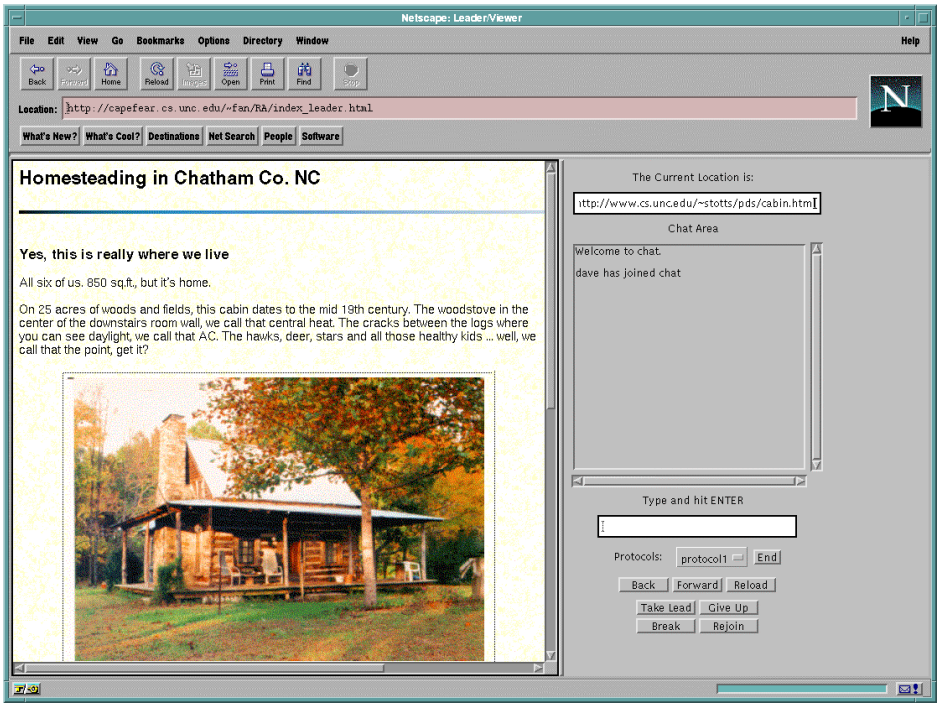

Figure 3. CobWeb screen shot, showing browser window and chat window

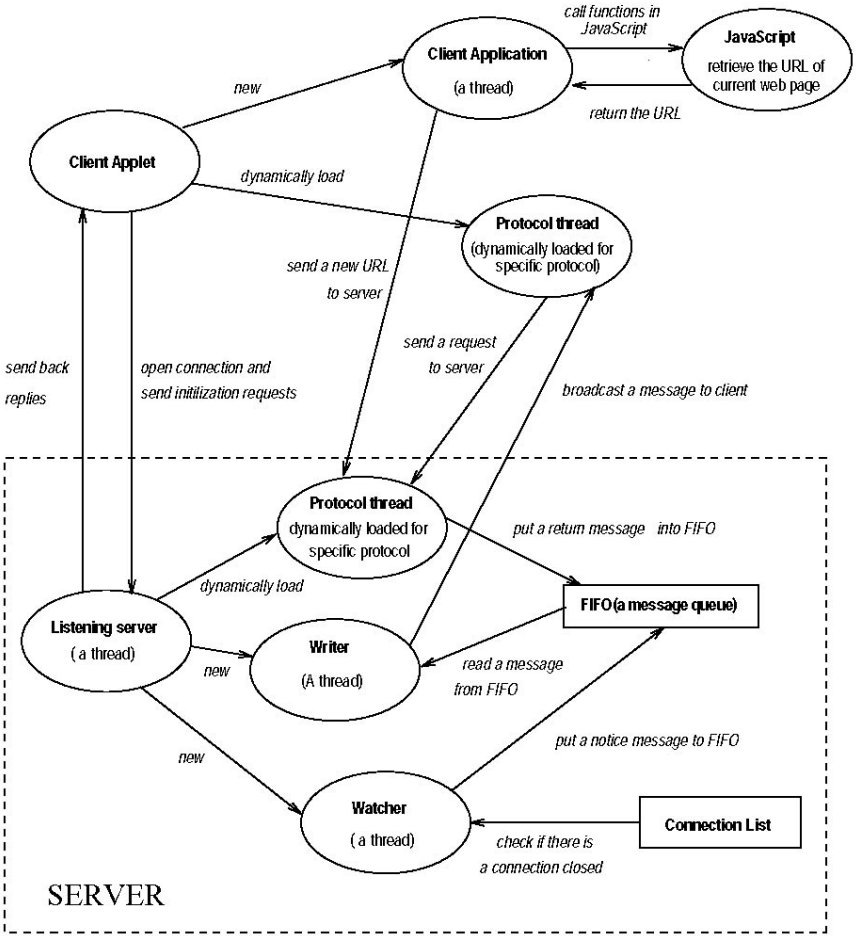

Figure 4. CobWeb system architecture diagram