

# Subdocument Invocation Semantics in Collaborative Hyperdocuments \*

Richard Furuta  
Department of Computer Science  
Texas A&M University  
College Station, TX 77843-3112  
furuta@cs.tamu.edu

Jaime Navon  
P. David Stotts  
Computer Science Department  
University of North Carolina  
Chapel Hill, NC 27599-3175  
stotts@cs.unc.edu

## Abstract

In this paper we informally explain a new Trellis model that incorporates colored tokens into the previously-described timed-Petri-net-based definition. We give examples of using Trellis to define protocols for Computer Supported Cooperative Work (CSCW). We then explain an interesting analog to procedure call we have developed for subdocument invocation in collaborative hyperdocuments. Trellis prototype implementations are based around a client-server architecture and interpret their specifications. Consequently they provide an environment for the rapid prototyping and incremental development of multi-user distributed protocols.

*Keywords:* Trellis, hypermedia, collaboration protocols, colored Petri nets, CSCW, groupware

## Introduction

In this paper we explain an unusual analog to procedure call we are developing for subdocument invocation in collaborative hyperdocuments. The differences arise due to the way in which users are mapped into the workings of a group. The work is presented in the context of the Trellis hypermedia model and our current Trellis implementation.

For clarity of discussion, we include an informal description of the extended Trellis model, based on colored timed Petri nets. This model is more expressive than that of earlier Trellis models [14, 6, 16, 17], and can

---

\*This work is based upon work supported by the National Science Foundation under grants IRI-9007746, IRI-9015439 and IRI-9496187, as well as the Texas Advanced Research Program under Grant No. 999903-155.

now used for specifying collaboration protocols (specifications of how group members interact in a collaboration). We have shown how to use collaborative protocols and other process specifications in Trellis for software process definition [18], prototyping CSCW systems [5], and multi-behavior image browsing indexes [4].

Hypertext has been suggested as an important component of systems supporting the software engineering process [3] and computer-supported cooperative work (CSCW) [7]. Hypertext can be generalized even further; systems such as Apple's HyperCard [1, 2] are used to implement general-purpose computer applications; we call such hyperdocuments *hyperprograms*. In Trellis, the functionality of a hyperprogram is attained via encoding state machines in the hyperdocument link structure; in other systems, a scripting language is used.

In the CSCW domain, the coordination of the interactions among humans and between human and computer can be formalized and refined through the expression of protocols [19, 9]. With Trellis, we have formally expressed such actions in a hypermedia model, and provided a client/server implementation for experimentation.

In section following we explain the details of the Trellis model; this material is not all new, but we include it for complete understanding of the later material. In section we discuss how the colored tokens in a Trellis model affect the group interface of a CSCW application. We conclude in section with an explanation of the subtle issues we have uncovered in subdocument invocation when a collaborating group is using a hypermedia document.

## Trellis hypermedia model

The Trellis project is an ongoing effort to create interactive systems that have a formal basis and that support analytical techniques. The first such effort was a hypertext model [14], with a followup framework for highly-interactive time-based programming (termed *temporal hyperprogramming* [15]). The model we present here is an extension of these earlier designs that explicitly distinguishes the various agents acting within a linked structure, and that provide an analyzable mechanism

with which agents may exchange data. This new model basically follows the Trellis framework of annotating a form of Petri net, and using both graph analysis and state-space analysis to exploit the naturally-dual formalism.

We briefly discussed the Trellis model in WETICE '94 [13], in the context of a VR project using the model. In this paper, we are interested in some technical aspects of the model itself, so we will present a more thorough explanation of its components and its execution semantics — particularly in how Trellis encodes group collaborative activity.

#### *Annotated nets*

The Trellis model is an annotated place/transition net (Petri net).<sup>1</sup> We annotate the places, transitions, arcs, and net as a whole with control information and with information specifying the contents and characteristics of nodes in a hypertext. Annotations include Unix file names (on places, for basic document contents); link anchor names (on transitions); timing information (on transitions); display control (on arcs); and color expressions for group control (on arcs).

#### *Timing on links*

Most execution in a hyperdocument is caused by the browsing actions (link selections) of the users. Timing is used to give some behavior to the document without requiring user input. We employ a time structure very similar to that of Merlin's *Time Petri nets* [10], in which two time values — a delay, and a time-out — are associated with each transition. Either can range from 0 time units to  $\infty$ , as long as the delay is not greater than the time-out. Time values in Trellis are thought of as defining ranges for the *availability* of a link or an event. After a transition becomes enabled (one or more tokens in each input place), the amount of time indicated by the *delay* must pass before the net will allow the transition to fire. If a transition is enabled but the user continually fails to select it for firing, then the net will fire it automatically after an amount of time equal to the *time-out* goes by.

#### *Browsing a net*

The current Trellis prototype,  $\chi$ Trellis, is written for X-windows and has a client-server distributed architecture (as described in [8]). A central server process (the *engine*) implements the semantics of the colored net, providing the specification of the collaboration protocol, but it has no user-visible interface of its own. One or more (possibly distributed) client processes must communicate with the server to give users some visible (or consumable) representation of the information annotated on the net engine.

The net and its annotations are interpreted by interface clients as the links and node contents of a hypermedia document. Each place in the net that is marked with

a token is said to have *active content*; for each such place, the client takes several actions. First, the client will query the net engine to get the content annotation on the marked place; that file (text, graphics, audio, etc.) will be rendered appropriately for user consumption. The client further will query the engine to get the names of all transitions that are enabled leading out the marked place. These transitions represent links to other content nodes, and they are displayed appropriately in the interface as selectable anchors (in a link menu, as highlighted words in the text, etc.). When the user selects a link anchor by clicking with the mouse, the client requests the net engine to fire the associated transition. Tokens move around, and the client again renders the active contents.

#### *Colors on tokens*

Colors are associated with tokens, giving a simple typing system; colors can be used to distinguish one individual from another, one group from another, one task from another, or for other distinctions. A color expression appears on each arc entering and leaving a transition; when the transition is to be fired, the tokens in the input places are examined to see if their colors satisfy the color expressions. An appropriate set of input tokens is chosen, the transition is fired, and tokens with colors matching the output arc expressions are generated into the output places.

Consider the example shown in Figure 1. The color expression  $a + b$  on the left arc specifies two tokens of arbitrary colors, with the two colors being different. The expression  $2b$  on the right arc requires two tokens of one color, and it must be the same color as selected for “b” from the left place. The expression  $3a$  indicates that when fired, the transition will place 3 tokens of the color selected for “a” into its output place. Given the colors shown in the input places, there are two possible color substitutions:

- substitution 1: a is red, b is green
- substitution 2: a is blue, b is green

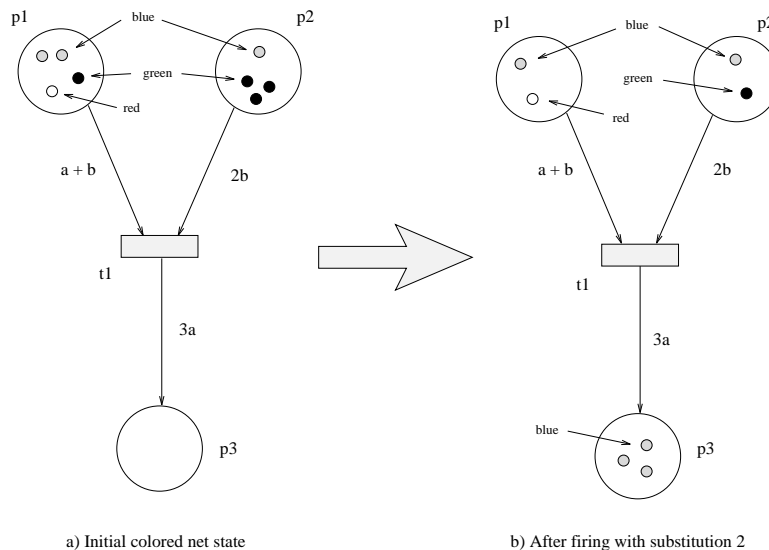
Figure 2 shows a Trellis net for directing a moderated meeting. This protocol is discussed in detail in an earlier paper [7], so the current discussion will just touch on some general points for complete understanding.

In this collaboration protocol, we use colors to distinguish *individual* participants (as opposed to *classes* of participant). The moderator has one color token, and each other participant has a different color token. There is a pool of new colors to select from when new participants are added by the moderator. In addition, one color (different from all participants) is used for mutual exclusion in floor control.

The meeting protocol is designed to limit some operations to the moderator and to permit the moderator to override participants if desired. It even has a section

---

<sup>1</sup>We assume some familiarity with basic net theory; an overview can be found in Murata's survey [11] or Peterson's survey [12].



**Figure 1: How colored place/transition nets work.**

that implements swapping the moderator job with one of the current participants. Notice that a moderator-colored token appears in both the “moderator” place and the “listen” place; thus, in this protocol, the moderator is also a participant in the discussion.

#### *Dynamic protocols*

Since Trellis protocols are interpreted, dynamic modifications can be made to the net. If the functionality already programmed in a net is not sufficient, a person editing the specification “on the fly” can add new functionality without terminating the current use of the net. For example, if a meeting is going on under control of the previously examined protocol, and if it is decided that two people should be able to get the floor and speak at one time (!!!), the Trellis editor can be used to drop another token (of the correct color) into the “mutex” place without ending the meeting. Further, if voting needs to be added to the activities participants can perform (along with “listen” and “speak”), the proper net links to create a voting function can be edited together and linked into the protocol structure while the meeting progresses. In addition to human editing, changes can be made through the actions of a decision-making program (a “silent” client, so to speak). Any process written to communicate with the engine via its RPC interface can request that its actions (like “add arc” or “fire transition”) be executed.

#### **Colors, group interfaces, and subdocuments**

In this section we will explain how colors in the net protocol affect or determine the view of the document specific users get when browsing.

A single user’s interface may be produced by the coordinated actions of one or many client processes. Many client processes may be required, for example, when multimedia material is presented to the user—one client may be responsible for delivering audio, another for video,

a third for graphics, and a fourth for text. Alternately, separate clients may produce different “views” of the information space, for the benefit of the reader.

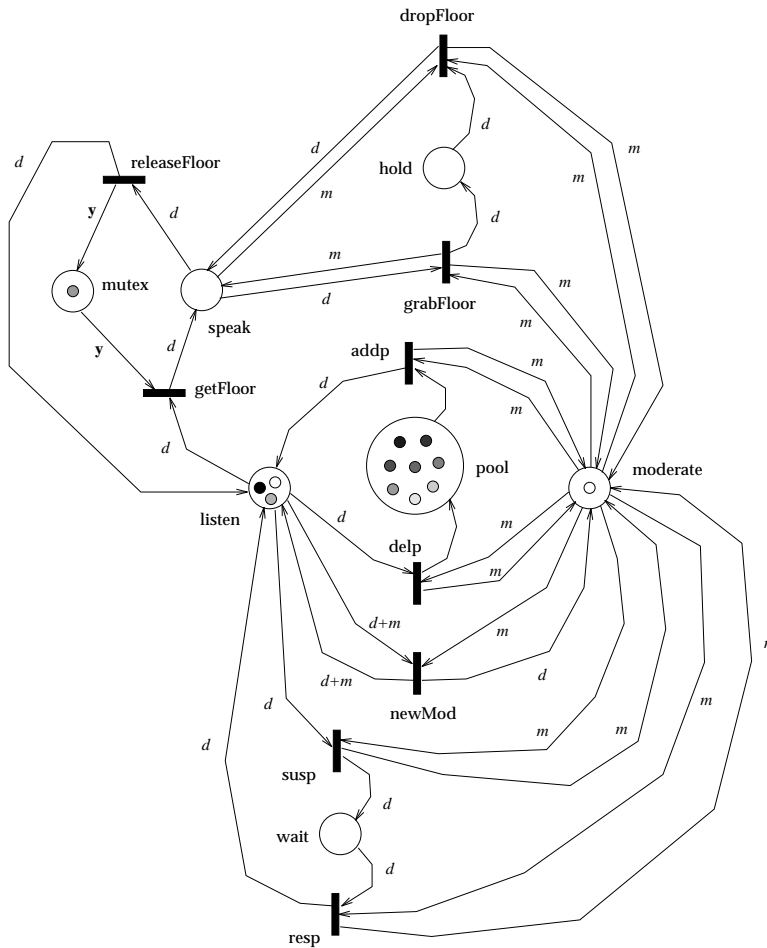
The colored net model and collaboration protocols have been developed primarily to support the coordinated efforts of *multiple* users. These applications also are implemented with multiple clients, where each user invokes separate instances from a set of relevant clients. Certainly a particular user may invoke a collection of clients intended to tailor the information space display, but in most cases there will be separate instances of the same client process, homed on separate workstations but communicating with the same server process. Communication and coordination in this case is managed through the intermediary of the single server process.

#### *Color-specific browsing clients*

Figure 3 shows a color-specific view of the net from Figure 1. In this example, we see the tokens that are of concern to a browser run for color “green”. In part (a), the green browser would display contents for places p1 and p2, since those are marked with green tokens. The engine will still report that transition t1 is enabled, since all colors are considered in the engine, so the green browser will present a link in its interface for t1. After firing with substitution 2, part (b) results. The green browser will now display only content for place p1, since the green token in p2 was consumed in firing t1.

Browsers for other colors will be generating similar constrained views of the hyperdocuments. Each different user, or users of each different class, will run browsers of the appropriate different colors.

Figure 4 shows the xTrellis system browser running on green tokens for the moderated meeting net from Figure 2. The xTed editor is also running, visible in the background. Notice that the green participant sees an



**Figure 2: Meeting protocol implementation**

explanation of the listening function, and has one action (getFloor) to choose in the link menu to the left of the text window.<sup>2</sup>

In contrast, Figure 5 shows for the same meeting protocol the xTrellis browser running on red tokens — the Moderator color in this case. The Moderator sees two browser windows, since two places are marked with red tokens. One window shows the same “listen” contents the green participant sees. The other window shows the contents of the “moderate” place. Note that from this window, the Moderator can select several meeting-controlling actions like “addp”, “delp”, and “swapMod”; since only the red browser will show this window, only the Moderator will be able to invoke the control functions.

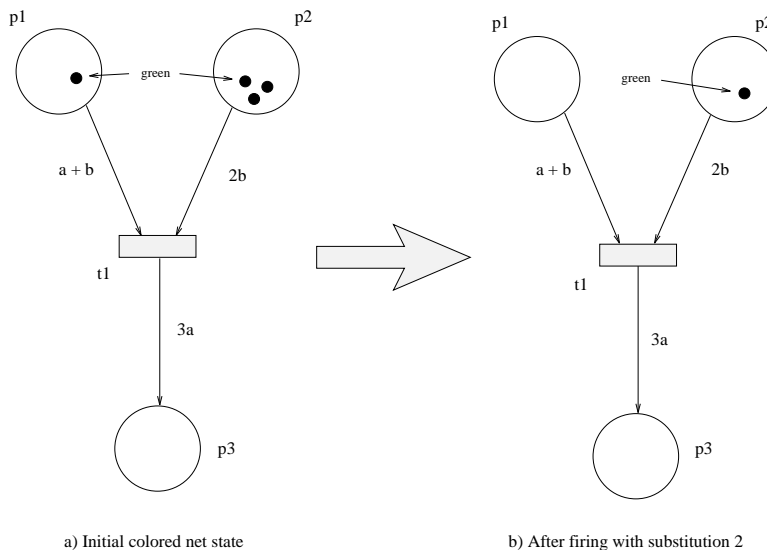
Consider what happens if Green clicks on the ‘getFloor’

<sup>2</sup>The “SELECTOR” window to the left of the “listen” window is where the browser places buttons that have no content place. In this case, we see only a “done” button to end execution of the browser; if the browser were displaying images with an external viewer (like *xv*) the buttons leading out of the image places would be put in the “SELECTOR” window.

button in his interface (Figure 4). In the meeting protocol (Figure 2) we see that the green token will move to the “speaking” place; Green will then see the browser interface shown in Figure 6. Green now has the option of selecting “releaseFloor” to stop speaking and return to listening. Figure 7 shows the view the Red view of the document when Green gets the floor. Notice that the Red “listen” window shows no selectable links; since Green has the floor, Red (in the listener role) is limited to listening (the protocol enforces one-at-a-time mutual exclusion on the floor). However, in the Moderator role, Red can still perform control functions. Note that in this case, since there are only two colors participating, when Green has the floor, the Moderator cannot remove anyone from the meeting, so the “delp” link does not appear in the Moderator window. This happens due to the semantics of color expressions. The color expressions on the arcs coming into the “delp” transition in the net (Figure 2) are “d” and “m” indicating that two different colors must be involved; the Moderator, therefore, cannot remove himself from the meeting.

### *Subdocument hierarchy*

We use subdocuments as a means of controlling the complexity of a Trellis hyperdocument. A hierarchy is ob-



**Figure 3: Color-specific view of a net**

tained by allowing the content of a place to be another Trellis hyperdocument. We will speak of a net being invoked as a subdocument as a *subnet*. A net containing a place whose content is a subnet is termed the *parent net*. Browsing a parent net may cause a subnet to be invoked; this happens when a token in the parent net moves into a place that has a subnet as content.

In the older Trellis model (without token colors) the semantics of invoking a subnet was straightforward. When the subnet place in the parent net first became marked with a token, a net process was started (for the subnet structure) and a browser was invoked to interpret the subnet for the user. Two browsers would then be concurrently active (one for the parent net, one for the subnet) and the user could select links in either structure. When browsing the parent net, if the place containing the subnet lost all tokens, the subdocument browser was terminated. The initial marking of the subnet was predefined in its specification.

Adding colored tokens to the model, though, has introduced an ambiguity that is not present in the un-colored case. How shall colors be mapped to the tokens of the invoked subnet — that of the parent net or new, unused color . . . or some mixture?

In Trellis, we treat subdocuments as independent documents. In other words, our data model (or, as is more accurately the case, process model) does not distinguish between documents and subdocuments. A subdocument in one structure might be a main document on its own in another context. Any Trellis document can be a subdocument of another; conversely, any Trellis subdocument can be independently browsed outside the enclosing context.

However, browsing a subdocument may not produce the same run-time effects in a subdocument context as

it would when browsed independently. The differences arise from the multi-reader semantics we have developed for collaborative hypermedia. The addition of colors to the Trellis model has led us to discover semantics of subdocument invocation that are not evident in “normal” hypermedia systems (i.e., non-collaborative or single reader systems).

#### Invocation modes

We have identified two ways to invoke a subdocument: **proxy**, and **join**. When one **proxys** a subdocument, one adds tokens to the net that are of some color already active in the net. When one **joins** a subdocument, one adds tokens that have colors *not* active in the net. The difference is subtle: **proxy** allows a reader (by sharing a token color) to “duplicate” some other reader, one already using the subdocument; **join** requires that a reader enter a document as a new entity, distinguished with a new color. Thus **proxy** cannot alter the future behavior of a subdocument, since any action that may be taken by the proxy-reader could also be taken by the original reader with the same color. **Join** can change the future behavior of a subdocument, since **join** in essence adds a new (and different) participant to a group that is executing the protocol of the subdocument.

We have also identified two ways to classify a subdocument: **new**, and **old**. **Old** means that an existing copy of the subdocument (embodied by a currently executing engine) is involved in the invocation. **New** means that a fresh engine is executed to embody the subdocument net. We allow **old** to default to the behavior of **new** if there are no existing active copies of the subdocument.

Crossing these properties gives us ostensibly four subdocument invocation modes, though it turns out that two modes have very similar semantics. We now discuss the nuances of each mode.

- **Join-old.** Invoking reader gets tokens in an existing

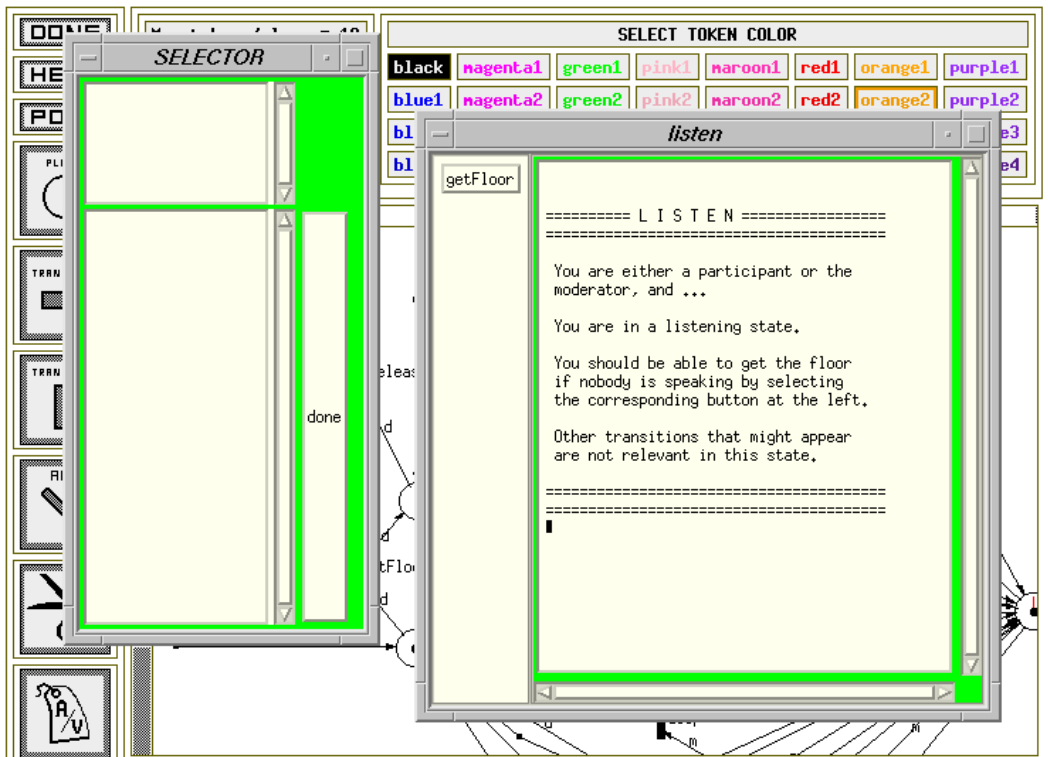


Figure 4: xTrellis screen showing “Green” participant browser

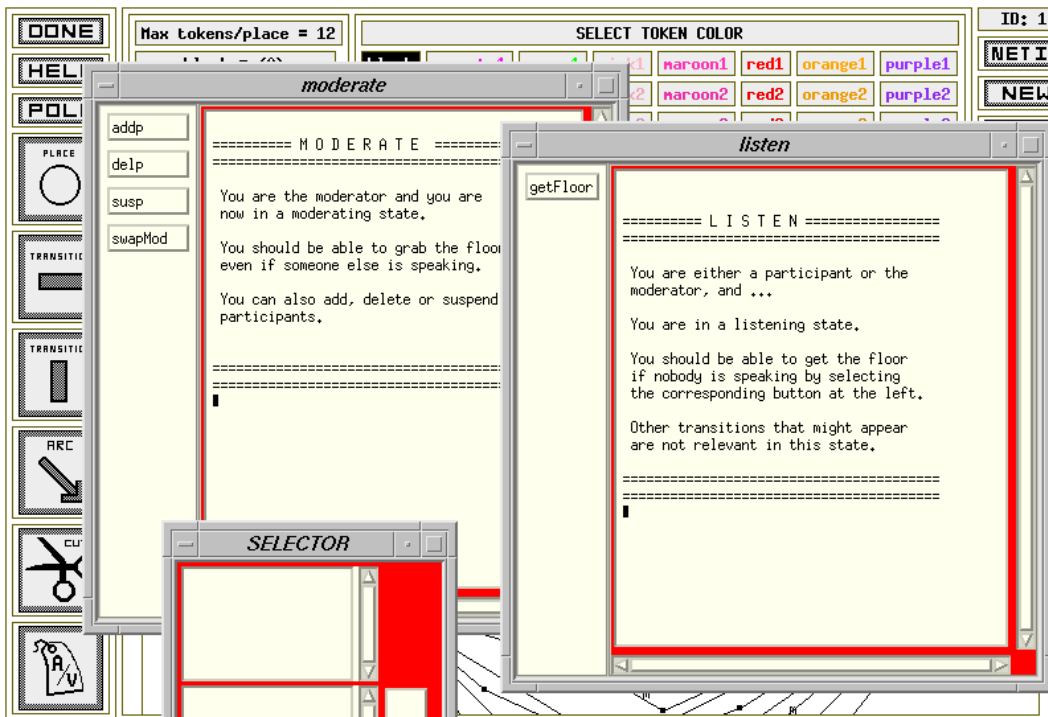


Figure 5: xTrellis screen showing “Moderator” browser

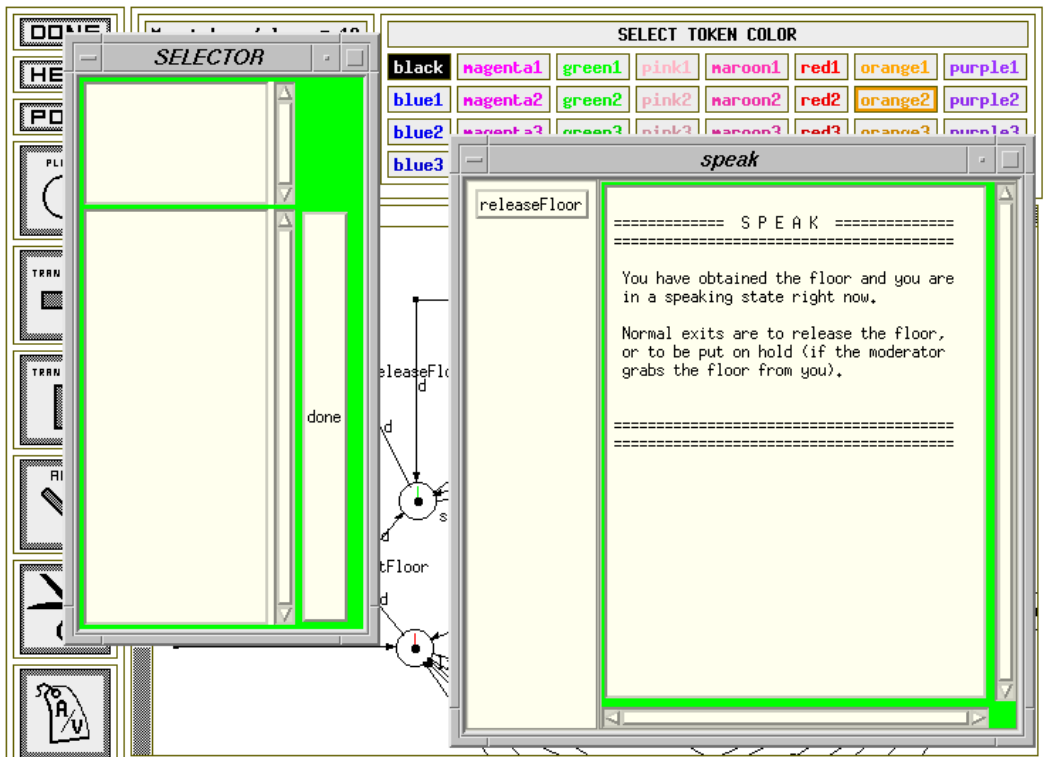


Figure 6: "Green" after getting floor to speak

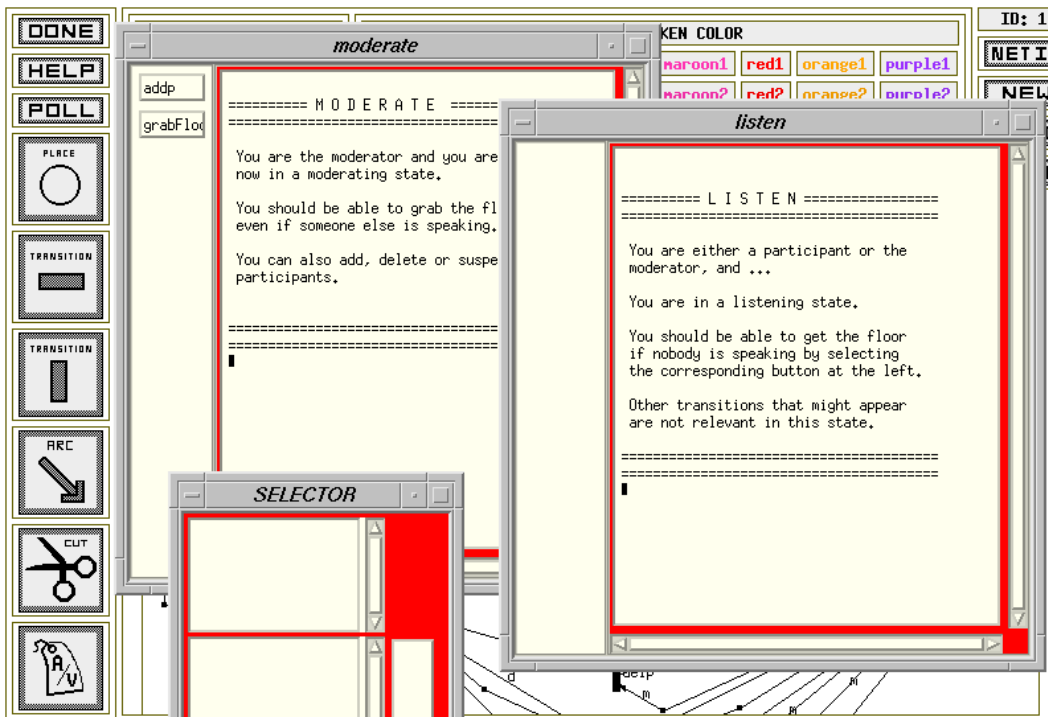


Figure 7: "Moderator" browser after "Green" gets floor

engine running a copy of the subdocument. Invoking reader receives a token color not already represented in the net, and so takes on the role of a new and different protocol participant. The presence of the invoking participant may then alter the future behavior of the subdocument.

- **Join-new.** This mode create a new engine to run the subdocument (even if there are many currently active copies of the document), and the reader becomes the first participant in the document protocol. Activity starts at the initial places of the document. The invoking reader is given a new token color, though this had limited meaning since the engine is new and there are no other participants in the document.
- **Proxy-old.** The invoking reader assumes the role of some participant in an existing copy of the subdocument. The invoking reader must receive the same token color as one of the active colors in the engine. In this case, the reader “steps into the shoes” of an existing participant. The invoking reader does not replace that participant, but rather the protocol capabilities of that participant are duplicated in two readers.
- **Proxy-new.** This mode is very similar to **join-new**; in fact, for our experiments we have defined the semantics to be the same. Another possibility would be to define **proxy-new** to be an error. Since the engine is new, there are no current participants in the document. One could argue that the invoking reader cannot “step into the shoes” of a non-existent participant. We have chosen instead to simply default this case to **join-new**.

We have identified a third dimension of subdocument invocation that is not yet fully explored: **initial token placement**, or where to have the invoking reader begin browsing. For our current experiments, both **join** and **proxy** place new tokens in initial places, identified as special in the Trellis document definition. Other possibilities include adding tokens to places that already have tokens at invocation time (placement based on subdocument activity); and adding tokens to places according to some parameter specified by the invoking reader at invocation time (placement based on reader preferences). In future experiments we will be exploring these and other variations on this dimension.

The specific invocation mode needed in a document depends on how the creator of the protocol imagines the group interaction naturally will progress. In cases where colors are used to distinguish *classes* of actor, it makes sense that a subdocument might be invoked so that the browser uses a color already in use (one of the proxy modes); the specific color chosen would be the one appropriate for the class of user trying to invoke the subdocument, and that color would be determined by the color of the parent browser.

If colors are used to distinguish individuals, then invoking a subdocument might naturally require that a new browser have a unique color; in this case, the parent color could be used, but if it were already present a different one would be chosen.

## Conclusions

We conclude that subdocument invocation in hypermedia initially appears analogous to procedure call in programming languages. In traditional single-user hyperdocuments, this simple analogy holds fairly well, being different only in the question of where in a subdocument browsing should begin. In most programming languages, procedures begin execution when invoked at some specified first instruction. This works well for simple single-user subdocuments as well, though it is not entirely satisfactory as a metaphor, since it is sometimes desirable to begin browsing a subdocument at places other than “the beginning.”

Complications arise with the procedure call metaphor when hypermedia is considered in a multi-reader, collaboration context, as we do in Trellis. There are several different dimensions to the problem that arise from the idea of the *role* of the reader in some group activity. The Trellis model with colored Petri nets has a succinct formal way to represent this situation, and we have developed so far the *proxy* and *join* invocation modes to deal with the new semantics. We have also discovered that collaboration semantics requires specification (when a subdocument is invoked) of whether the browser wishes to read an existing copy of the document (that might have other readers acting in it) or a fresh copy, in which he would be sole reader (initially).

## REFERENCES

1. Apple Computer, Inc. *HyperCard User's Guide*. Apple Computer, Inc., 1987.
2. Apple Computer, Inc. *HyperCard Script Language Guide: The HyperTalk Language*. Addison-Wesley, 1988.
3. James Bigelow. Hypertext and CASE. *IEEE Software*, 5(2):23–27, March 1988.
4. R. Furuta and D. Stotts. Structured dynamic behavior in hypertext. *Int'l Journal of Man/Machine Interaction*. accepted to appear.
5. R. Furuta and P. D. Stotts. Interpreted collaboration protocols and their use in groupware prototyping. In *Proc. of the 1994 ACM Conference on Computer Supported Cooperative Work (CSCW '94)*, pages 121–131. ACM, New York, October 1994.
6. Richard Furuta and P. David Stotts. Programmable browsing semantics in Trellis. In *Hypertext '89 Proceedings*, pages 27–42. ACM, New York, November 1989.
7. Richard Furuta and P. David Stotts. Interpreted collaboration protocols and their use in groupware prototyping, 1994. Internal report.
8. Richard Furuta, P. David Stotts, and Gregory D. Drew. Experiences with a client-server-based architecture for a distributed structured hypertext

- system. In *Proceedings of the EP92 Conference*. Cambridge University Press, 1992. To appear.
9. Anatol W. Holt. Diplans: A new language for the study and implementation of coordination. *ACM Transactions on Office Information Systems*, 6(2):109–125, January 1988.
  10. Philip M. Merlin and David J. Farber. Recoverability of communication protocols—implications of a theoretical study. *IEEE Transactions on Communications*, COM-24(9):1036–1043, 1976.
  11. Tadao Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.
  12. James L. Peterson. Petri nets. *ACM Computing Surveys*, 9(3):223–252, September 1977.
  13. P. D. Stotts and J. Purtilo. Virtual environment architectures: Interoperability through software interconnection technology. In *Proc. of the Third Workshop on Enabling Technologies (WET-ICE '94): Infrastructure for Collaborative Enterprises*, pages 211–224. IEEE Computer Society Press, April 1994.
  14. P. David Stotts and Richard Furuta. Petri-net-based hypertext: Document structure with browsing semantics. *ACM Transactions on Information Systems*, 7(1):3–29, January 1989.
  15. P. David Stotts and Richard Furuta. Temporal hyperprogramming. *Journal of Visual Languages and Computing*, 1(3):237–253, 1990.
  16. P. David Stotts and Richard Furuta. Hierarchy, composition, scripting languages, and translators for structured hypertext. In A. Rizk, N. Streitz, and J. André, editors, *Hypertext: Concepts, Systems, and Applications*, pages 180–193. Cambridge University Press, November 1990. Proceedings of the European Conference on Hypertext.
  17. P. David Stotts and Richard Furuta. Dynamic adaptation of hypertext structure. In *Proceedings of Hypertext 91*, pages 219–231. ACM, December 1991.
  18. P. David Stotts and Richard Furuta. Modeling and prototyping collaborative software processes. In Shimon Y. Nof, editor, *Information and Collaboration Models of Integration*, pages 365–390. Kluwer Academic Publishers, June 1994. Based on the NATO Advanced Research Workshop on Integration: Information and Collaboration Models, Il Crocco, Italy, June 6–11, 1993. Also available as Technical Report TR93-020, Computer Science Collaboratory, Univ. of North Carolina at Chapel Hill, 1993; and as Tech Report TAMU-HRL 93-006, Hypermedia Research Laboratory, Texas A&M University, July 1993.
  19. Willem R. van Biljon. Extending Petri nets for specifying man-machine dialogues. *International Journal of Man-Machine Studies*, 28:437–455, 1988.