

A Hypermedia Basis for the Specification, Documentation, Verification, and Prototyping of Concurrent Protocols*

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

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

ABSTRACT

The Trellis project examines hypertext as a mechanism not only for structuring information, but also for structuring *process*. Process can be specified by a protocol, and the number of areas of study in which protocols are of relevance is large.

In this paper we present the definition of a new Trellis model, an expansion of earlier models, that incorporates colored tokens into the previously-described timed-Petri-net-based definition. We give examples of using Trellis to define protocols in the areas of Software Engineering and Computer Supported Cooperative Work (CSCW). 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, protocols, colored Petri nets, CSCW, software engineering

1 INTRODUCTION

A unifying focus in hypertext research is the identification, definition and development of commonalities with previously orthogonal areas of study. Hypertext has been suggested as an important component of systems supporting the software engineering process [4, 21] and computer-supported cooperative work (CSCW) [23, 7]. Hypertext can be generalized even further; systems such as Apple's HyperCard [1, 2] are used to implement general-purpose computer applications.

A characteristic of these investigations is the central nature of *protocol specification* in addressing the domain's requirements. Software engineering protocols include formal and semi-formal artifacts that provide, for example, requirements specifications. The relationships among these artifacts and the procedures that are followed in their manipulation are in turn defined by higher-level protocols. In the CSCW domain, the coordination of the interactions among humans and be-

tween human and computer can be formalized and refined through the expression of protocols [29, 13].

The Trellis project is investigating the structure and semantics of hypertextually-described interaction [24, 23]. As the work has developed over a number of years, we have broadened our scope from the study of hyperdocuments to encompass *hyperprograms*. A hyperprogram associates user-manipulatable information (the hypertext) with user-directed execution behavior (the process). Consequently a hyperprogram can be said to integrate task with information.

Generalizing, the hyperprogram's execution behavior can be defined by the collective actions of a group of entities, not only the direction of a single user. Such actions can be generated by, and can coordinate the activities of, a collection of human users, computer-based processes, and indeed reflect and are driven by the intentions of the hyperprogram's author. The structural characteristics of a *hyperdocument* permit the specification and documentation of a concurrent protocol and the characteristics of the specification's definition potentially enable the protocol's verification. It is the dynamic characteristics of the *hyperprogram* that permit the prototyping and deployment of the resulting specification.

Section 2 will present the current description of the Trellis hypertext model. This model is a refinement of those previously described [24, 6, 25, 27], is based on colored, timed Petri nets, and has been implemented in several versions using a client-server-based architecture [8]. Following the description of the model, section 3 will give three brief examples its use in protocol specification in two example domains—Software Engineering requirements specifications and CSCW protocols. Section 4 contains discussion about some of the issues in implementation of protocols in the model and section 5 presents conclusions.

2 TIMED, COLORED TRELIS 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 [24], with a followup framework for highly-interactive

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

time-based programming (termed *temporal hyperprogramming* [26]). 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. The discussion in this section is modified from an earlier paper [23].

The following short section outlines some of the basic concepts and terminology of Petri nets, their structure, and common behaviors; readers already familiar with these notions may choose to skip it. Following that we introduce the group- and timing-specific net definitions, and finally the model of collaboration structures based on these nets.

2.1 Net theory basics

A Petri net¹ [20] is a bipartite graph with some associated execution semantics. The two types of nodes in a net are termed *places*, represented visually as circles, and *transitions*, represented visually as bars. Activity in the net is denoted with *tokens*, drawn as dots in the places. Two nodes of the same type may not be joined by an arc. Given the arc structure of a net, the set of inputs to a node n is termed the *preset* of n , and the set of output nodes is termed the *postset* of n .

A transition t in a Petri net is said to be enabled if each place in the preset of t is *marked*, i.e., contains at least one token. Once enabled, a transition t may *fire*, causing a token to be removed from each place of the preset of t and depositing one token in each place of the postset of t . A *net marking*, or *net state*, is a vector of integers, telling how many tokens reside in each place. Execution of a Petri net begins with some initial marking, and continues through a sequence of state changes caused by choosing an enabled transition in the current state and firing it to get a new state. Execution certainly terminates if a state is reached in which no transitions are enabled, but it may also be defined to terminate with any marking tg that has some special significance to the user of the net.

2.2 Colored timed net

The Trellis model is based primarily on a synchronously executed, transition-timed Petri net as the structure of a hyperprogram. Recently, we have employed a form of net model known generically as *high-level* nets. High-level nets have been introduced in several forms by different researchers, including predicate-transition nets [9], colored Petri nets [14], and nets with individual tokens [19].

¹A Petri net is a specific instance of the general place/transition net, which describes the place and transition net syntax that is common to many forms of concurrent computation model. The Petri net is a form of PT net with the specific (and familiar) execution semantics that will be described here.

We present our ideas in a hybrid notation. We will use Jensen's terminology of colored nets, but the simplified syntax presented by Murata in his high-level net summary [18]. All forms of high-level nets can be translated into one another, and are thus equivalent, but the simple syntax we use creates explanations that are more clear.

In colored nets, tokens have type (color) and may carry a data structure. A token of one color is distinguishable from a token of another color; within a color class, however, individual tokens cannot be distinguished from one another. The timing of the original Trellis model has been retained and combined with color to produce this model:

Definition 1 Colored timed net structure

A colored timed net structure CTN is a 5-tuple,

$$CTN = \langle S, T, F, \kappa, \tau \rangle$$

in which

$S = \{p_1, \dots, p_n\}$ is a finite set of places with $n \geq 0$,

$T = \{t_1, \dots, t_m\}$ is a finite set of transitions with $m \geq 0$, and $S \cap T = \emptyset$,

$F \subseteq (S \times T) \cup (T \times S)$ is the flow relation, a mapping representing arcs between places and transitions.

$\kappa : \{\kappa_1, \dots, \kappa_r\}$ is a finite set of colors for typing tokens, where each color is a function $\kappa_i : S \rightarrow \{0, 1, 2, \dots\}$;

$\tau : T \rightarrow \{0, 1, 2, \dots\} \times \{\infty, 0, 1, 2, \dots\}$ is a function mapping each transition to a pair of values termed release time and maximum latency respectively. For any transition $t \in T$, we write $\tau(t) = (\tau_t^r, \tau_t^m)$ and we require that $\tau_t^r \leq \tau_t^m$.

In this model, we have simplified the notation used in Reisig [20] by assuming that the weight on each arc is 1, and that the token capacity of each place is unbounded. A net marking is a vector of token counts, with each token count being a vector of color counts; a marking provides a snapshot, at some point during execution, of how many tokens of each color reside in each place.

For a transition $t \in T$, its release time represents the number of time units that must pass once t is enabled before it can be fired; its maximum latency represents the number of time units that may pass after t is enabled before it fires automatically.

This temporal structure is very similar to that of Merlin's *Time Petri nets* [16, 17], with a few differences. The two time values for each transition here are integers, whereas Merlin used reals. We also have a need for the maximum latency to possibly be unbounded, using the special designation ∞

which is not in Merlin's model. Finally, times are not thought of as durations for transition firing in Trellis. Transitions are still abstractly considered to fire instantaneously, like the clicking of a button in a hypertext interface. Time values in Trellis are thought of as defining ranges for the *availability* of an event.

2.3 Collaboration protocol structure (CPS)

The timed Trellis model of hypertext uses the structure and execution rules of timed Petri nets to specify both the linked form and the browsing semantics of a hypertext. This logical structure then is interpreted through a layer of indirection to arrive at a displayed form for reader consumption and interaction. Hypertext content and linked structure are thus effectively separated by the timed Trellis model.

Definition 2 Collaboration protocol structure

A collaboration protocol structure is

$$CPS = \langle CTN, M_0, C, W, B, P_l, P_d \rangle$$

in which

$CTN = \langle S, T, F, \kappa, \tau \rangle$ is a colored timed net,

$M_0 : S \rightarrow \langle c_1, c_2, \dots, c_r \rangle$ is an initial marking (or initial state) for CTN , where $r = |\kappa|$ and $\forall p \in S, M_0(p)_i = c_i = \kappa_i(p)$,

C is a set of document contents,

W is a set of windows,

B is a set of buttons,

P_l is a logical projection for the document,

P_d is a display projection for the document.

A CPS consists of a CTN representing the document's linked structure, a marking to tell how many tokens of each color start in each net place, several sets of human-consumable components (*contents*, *windows*, and *buttons*), and two collections of mappings, termed *projections*, between the CTN, the human-consumables, and the display mechanisms. A window from W is a logically distinct locus of information. A button from B is an action that causes the current display to change in a specified way. Content elements from C can be many things: text, graphics, tables, bit maps, executable code, sound, or, most importantly, *another CPS*.

A logical projection P_l provides mappings from components of a CTN to the human-consumable portions of a group work environment as mentioned above. Each place in the CTN has a content element from C mapped to it, as well as an element of W for the abstract display of the content. Each transition in the net has a logical button from B associated with it. The display projection P_d is a set of mappings that take the logical components and produce tangible representations, such

as screen layouts, sound generation, video, etc. P_d determines how things like text and buttons are visibly displayed, e.g., whether a user selects a link from a side menu or from highlighted words (or icons) within the content display.

The net marking M_0 enables a CPS to represent both the logical structure of a collaboration and the current state of inter-activity within it. Together with the execution rules of the CTN, every marking is a characterization of the possible paths in a collaboration from the browsing point it represents. Different browsing patterns (for, say, different classes of reader) can then be enforced on a single CPS simply by choosing appropriate different initial markings.

2.4 Execution rules for a CPS

The execution behavior of a CTN provides an interpretation of the collaborators' experiences when interacting under the control of a CPS. As in the original Trellis model, a token in a place p indicates that the contents of the place $C_l(p)$ are displayed for viewing (or editing, or some other interaction). Content elements come into and go out of view (or begin and end execution) as tokens move through the net. Transitions are fired by selecting logical hypertext buttons. When a transition t is fireable in the timed Petri net, its logical button $B_l(t)$ is displayed in some selectable area of the screen, such as on a (highlighted) word in a text section, or in a separate button menu.

The general execution behavior of the CTN in a CPS requires pattern matching to be done on all arc expressions that are inputs to a transition. The transition is enabled if there is one or more consistent color substitutions for the expressions. When the transition fires, one of the valid substitutions is chosen, the proper color tokens are removed from the input places, and output tokens are produced according to the substitution and the expressions on the output arcs.

2.5 Implementation note

The Trellis model has been implemented in prototype form a number of times. The initial implementation, called α Trellis, ran under SunTools. A current implementation, χ Trellis, uses the X-windows system. Each of the implementations is based on a client-server architecture, as has been described in [8]. A central server implements the semantics of the CTN and the CPS but has no user-visible manifestation. One or more potentially distributed client processes communicate with the server. (In the current implementations, this network communication protocol is achieved using RPC.) These clients produce the external representations of the hyperprogram.

A single user's interface, then, 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

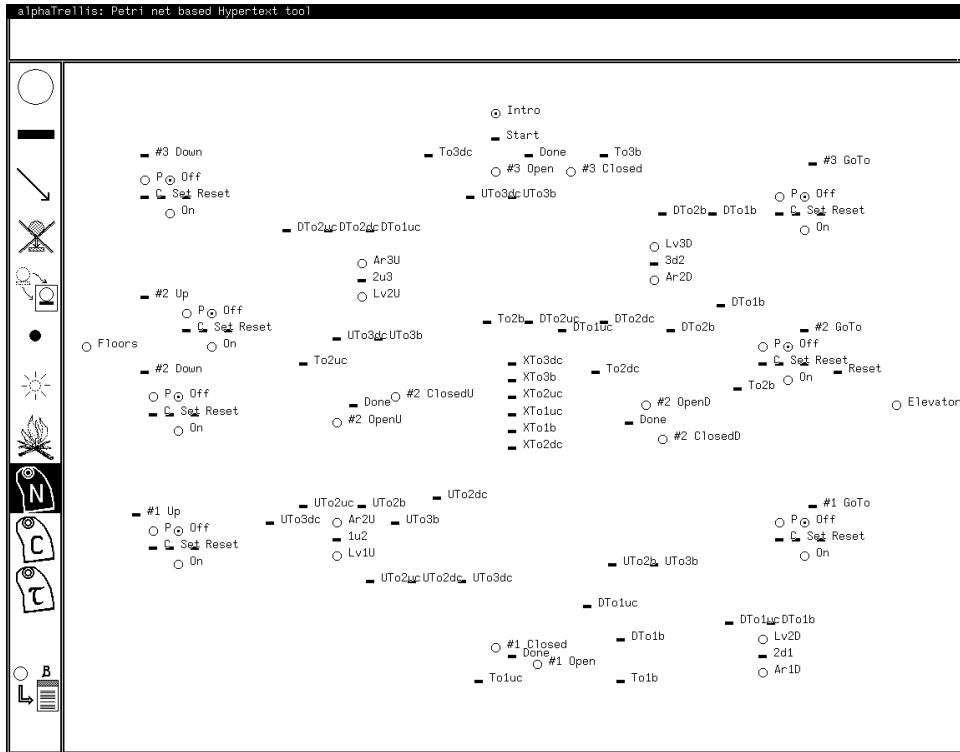


Figure 2: CPS specification with arcs suppressed

middle left of center (travel upward) and to the middle right of center (travel downward). The constructs in the middle of the diagram represent the behavior of the elevator on reaching a desired floor (opening the door and then closing it once again after a period of inactivity). In addition, a matrix associated with the second floor handles the case when a request arrives that requires an idle elevator to change direction.

It is possible to use a CPS to drive a simulation of the protocol. If the standard Trellis text browser is used, then external inputs to the protocol (for example in this case riders pushing buttons in this example) are represented as hypertextual buttons. In other words the selection of a link for following represents the rider's selection of an action. In addition, standard Trellis timing semantics cause the simulation to react dynamically to such external stimuli. It is also possible to create special purpose browsers that more closely simulate the problem domain. Figure 3 shows one such representation for this domain, presenting the protocol's response when the "floor one" button is pushed within an elevator that is initially idle on the third floor. The screens in the figure show two browsers: to the left the standard α Trellis text browser and to the right a custom browser representing the elevator's status. Since the Trellis prototypes are based on a client-server architecture, both browsers are active simultaneously and present different interpretations of the protocol's state. This particular custom browser was generated very quickly (with about half an hour's programming effort) to help provide insight into the CPS's behavior. Certainly with correspondingly greater effort, a more complex graphical representation

Mode	Event	New Mode
Off	@T(Running) when TempOK @T(Running) when TempLow @T(Running) when TempHigh	Inactive Heat AC
Inactive	@F(Running) @T(TempLow) @T(TempHigh)	Off Heat AC
Heat	@F(Running) @T(TempOK)	Off Inactive
AC	@F(Running) @T(TempOK)	Off Inactive

Figure 4: SCR (A7) requirements representation of a simple temperature control system (from [3])

of the elevator and its state could also have been provided.

3.2 Example two: temperature sensor

Figure 4 shows a condensed version of an SCR requirement, a formalism used initially in specifying the characteristics of the A7 aircraft [12]. This representation specifies the mode transitions that occur when a set of events become satisfied. An example specification, taken from [3], is shown in figure 4. This specification states, for instance, that if the system is in mode "off" and event "TempOK" holds, the system should enter mode "inactive" when event "running" becomes true. Similarly if the system is in state "heat", it will enter state "off" when event "running" becomes false.

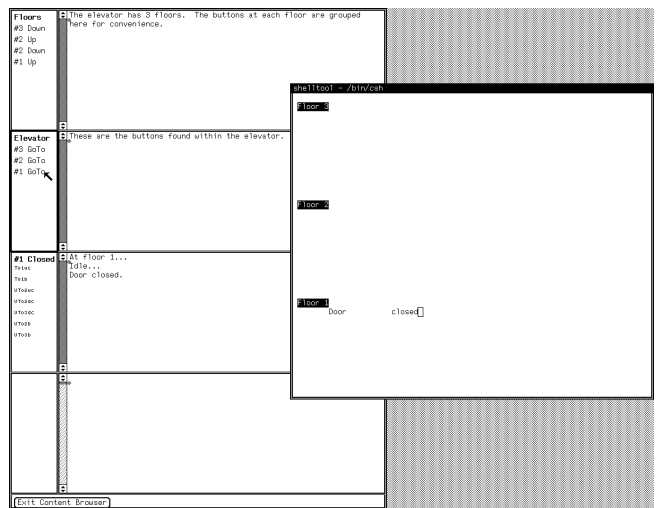
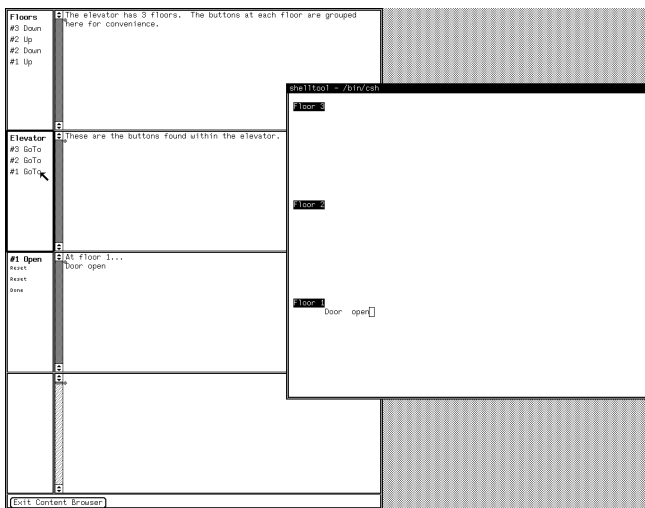
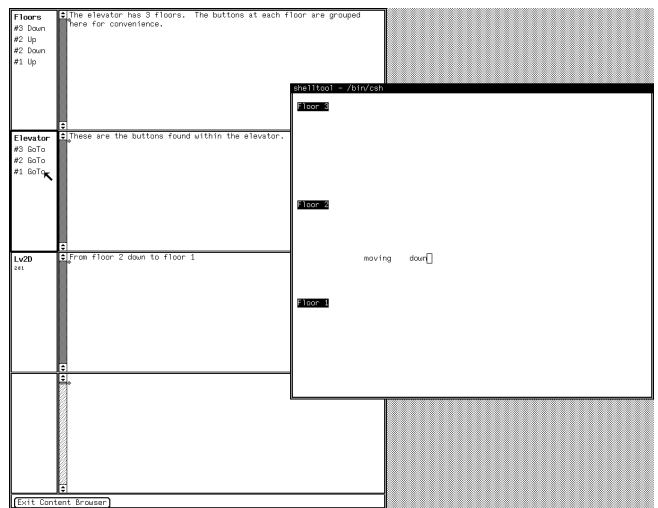
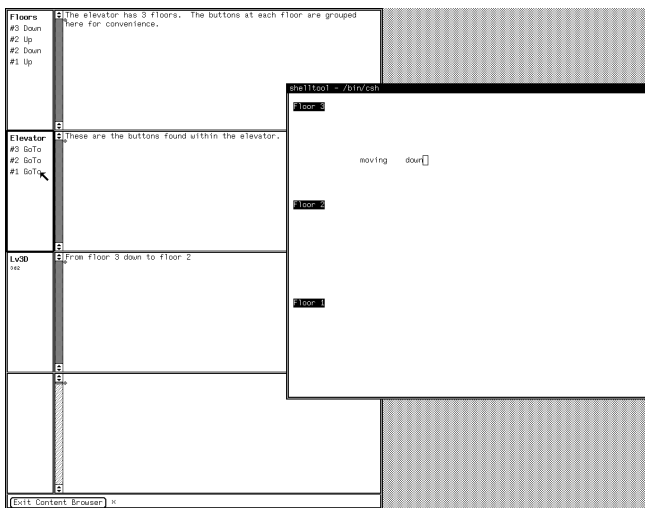
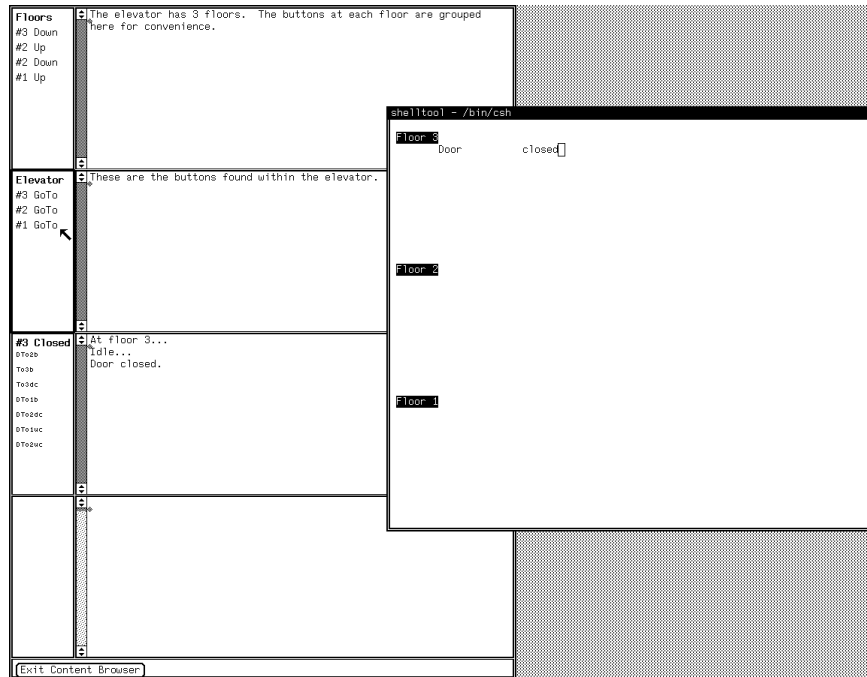


Figure 3: Executing the specification

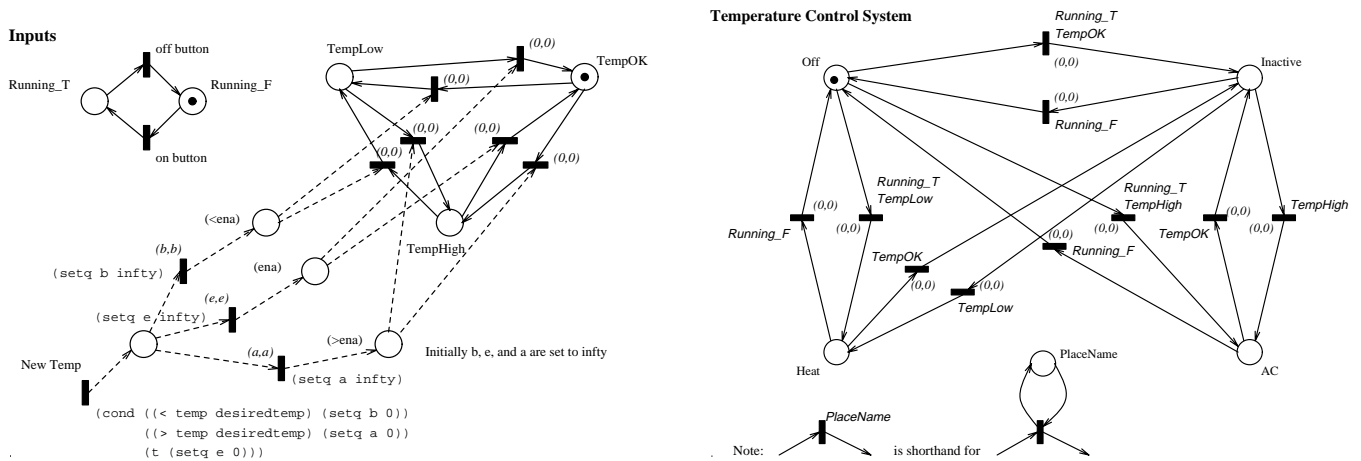


Figure 5: CPS representation of requirements given in figure 4

The example specification represents a simple temperature control system. The system has an “on”/“off” switch and a temperature sensor that is in one of three states: “TempOK,” “TempLow,” or “TempHigh.” Figure 5 shows a CPS encoding of the SCR specification. The net fragments on the left represent the encoding of the system’s controls while the fragment on the right represents the encoding of the SCR specification itself.

The fragment in the upper left hand corner of the figure represents the state of the on/off switch and the fragment to the left of center represents the state of the temperature sensor.² Events in the SCR sense are represented as transitions between states. Note that the temperature sensor fragment depends on the Trellis transition timing mechanism and also on Trellis’ previously-described dynamic adaptation agents [27] to translate externally-detected temperature readings into system-invoked transition firings.

The fragment on the right hand side ties the event transitions into the temperature system’s specification. In this net the shorthand notation of *PlaceName* means that the correspondingly-named Petri net place is connected by arcs to the designated transition, as illustrated at the bottom of the diagram.

Taken together, these two examples illustrate some features and shortcomings of the current CPS implementation. On the positive side, the specification of the structure, based on a formal mechanism, is amenable to subsequent analysis. Indeed Atlee and Gannon [3] studied the conversion of SCR specifications into a form that could be checked by Clarke’s MCB CTL-language model checker [5]. In doing so, they discovered some unexpected inconsistencies in previously-published specifications that were believed correct. The target they used is identical to the one we have used in our previously-reported research in the hypertext domain [28]

²There is no distinction in this figure in the sense of the net specification between the dashed lines and the solid lines—they are distinguished in the diagram merely for purposes of graphical presentation.

and consequently similar results can be expected for CPS applications.

Less positively, examination of the examples, particularly the elevator specification, illustrates the need for mechanisms to control the specification’s complexity. These issues will be revisited and discussed in section 4.

3.3 Example three: meeting protocol

Figure 6 shows a snapshot of χ Trellis, our current Trellis implementation prototype. In the upper left corner of this screen is the χ Ted CTN editor client, which is displaying a CPS for directing a meeting among several participants. This protocol is discussed in detail in an associated paper [7], so the current discussion will just touch on some general points.

Colored tokens are used in this specification to distinguish classes of participants. The specific classes of interest that defined in this example are the discussion’s moderator and its participants (in addition, some internal classes are defined to implement higher-level structures).

The CPS is designed to limit some operations to the moderator and to permit the moderator to override participants if desired. The remaining windows in the display represent the view seen by a meeting participants and by the moderator. In actual use, display of these documents would be distributed over multiple, separated, workstations and would serve as “traffic control” for an ongoing meeting.

As the associated paper details, the specification can be flexibly modified to reflect different meeting policies. Examples include creating a separate class of token for each participant (rather than an anonymous pool of participant-class tokens), modifications and additions of moderator-limited actions, and permitting the moderator to exchange roles with a participant. Since protocols are interpreted, dynamic modifications can be made, either by a person editing the specification “on the fly,” or through the actions of Trellis dynamic adaptation agents, which could modify the effective appearance of the

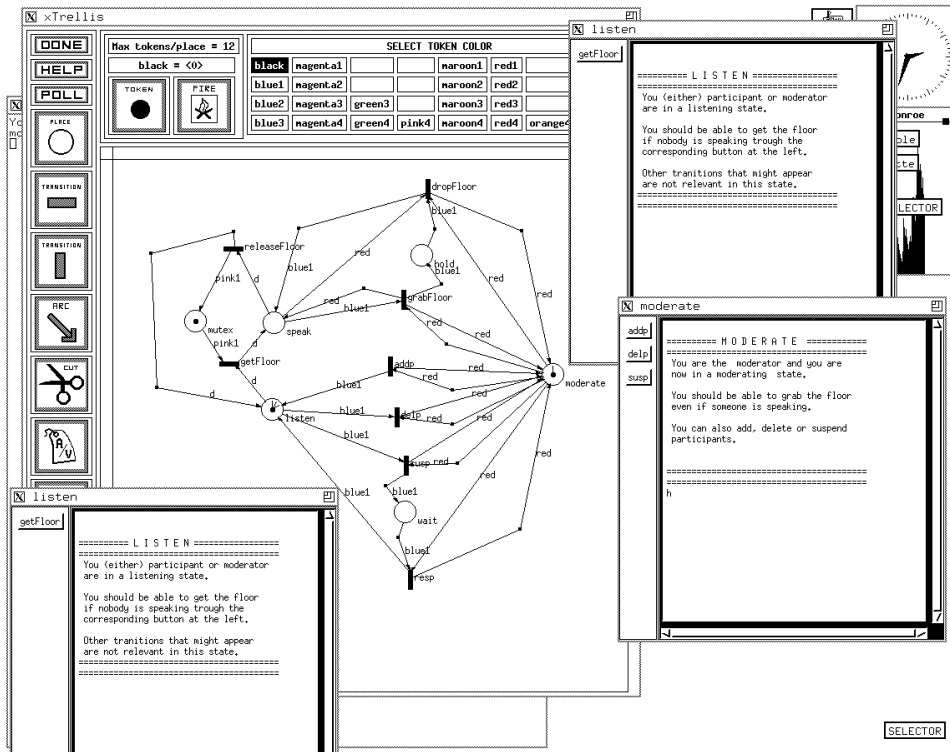


Figure 6: Meeting protocol implementation

protocol based on “observation” of the meeting participants’ actions.

Further issues relating to interpreted collaboration protocols will be discussed in the next section.

4 ISSUES AND DISCUSSION

4.1 Prototyping in hypermedia

The main point of this paper is to emphasize that with a process-based hypermedia model (like Trellis, or Hypercard, or KMS using the scripting language), hyperprograms can serve as specification documents, and even initial prototypes, for software systems. Process-based hyperdocuments can express the protocols underlying the behavior of software systems.

In a process-based hypermedia model, analysis is usually impossible since the process definition component is a complete programming language (scripting language). The uniqueness of the Trellis model is expression of processes with a notation that is less powerful for programming, but also fully amenable to state space analysis. This has been the point of some of our past research [28]; as shown in the example of SCR (A7 aircraft) requirements, analysis is often desirable, if not absolutely necessary. Little, if any, other hypermedia research has looked at this browsing process analysis. We are currently adapting the Trellis analysis methods to the new colored model.

In the case of collaborative software systems, the colored Trellis model is especially useful for expressing group interactions within a linked information structure. This was the point of the conference protocol example. To apply this capability, We have been designing a groupware prototyping method based on Trellis hyperdocuments. The method first specifies the necessary group protocols as hyperprograms, then migrates the Trellis structures into working programs by replacing the textual annotations with functional code modules [7].

4.2 Abstraction, hierarchy and CPS

Managing the complexity of a graphical specification is a general problem when hyperprograms are represented visually. A straightforward graphical representation rapidly becomes spaghetti-like in appearance. When visual means are used to specify the hyperprogram, the increasing complexity of the display rapidly limits the size of program that can be described.

Our experiments in reducing graphical complexity have centered around the mechanisms of abstraction and of hierarchy. We have described abstraction-based mechanism in earlier papers: a pair-grammar-based mechanism using a textual language as description mechanism [25] and a subsequent study using the graph grammar description to drive a visually-based specification interface [15]. An issue when the net is generated from a separate description rather than specified directly is what the placement of net elements should be.

A second study [11] found that standard graph layout algorithms were applicable in the Petri net domain, and that if the hyperprogram was rooted, that additional advantage could be gained.

Hierarchy—the restructuring of the information space to partition the net into smaller, more manageable chunks related to one another by higher-level structures, has been a component of the Trellis design since its initial definition. As noted above content elements in a CPS are arbitrary in form, and can be another CPS.

The semantics of invoking a nested CPS are straightforward. Following the standard Trellis semantics, a subnet is invoked when the associated place first became marked, and is uninvoked when the place became un-marked. The initial marking of the subnet is predefined.

The introduction of colored tokens has introduced an ambiguity that is not present in the single color case (as was the case in the older Trellis definitions). What color should the subnet tokens be give—that of the parent net or a new, unused color?

4.2.1 Subdocument invocation semantics

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).

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 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.

4.3 Interpreted protocols

The use of CPS to describe an interpreted protocol, as sketched in section 3.3, suggests the possibility of using the specification in a variety of ways during the protocol's life—design, implementation, training, verification, deployment, adaptation, and modification. We are currently examining the challenging research issues surrounding such applications. In this section, we briefly identify some of those issues.

Development of effective protocols to direct collaborative processes will require the identification of higher-level specification mechanisms, as discussed in the previous section. Further, we believe it useful if the specification language is based on a formalism that permits machine-based verification that the protocol does not suffer from inconsistencies. Such verifications might include testing that the protocol does not require the physically impossible (for example assigning two mutually incompatible actions to the same agent³ at the same instant in time) or the undesirable (for example the specification of useless actions whose results are not subsequently used).

Training people in the protocols can be based around the examination and computer-driven simulation of the protocol specifications. Certainly the characteristics of the protocol specification language is important here as well—a self-documenting and executable protocol specification language will aid in these activities.

Initial deployment of protocols will undoubtedly uncover deficiencies in the protocol's specification. Since CPS are interpreted, modifications can be made to the protocols “on the fly.” In such cases timely re-verification of the protocol will be necessary. We are examining techniques for incremental re-verification in response to this need.

It is possible that the computer network supporting the distributed application will become partitioned during a collaborative session. Indeed, this becomes more of a likelihood if some of the participants are using mobile or untethered interfaces. Ideally, the individual participants would be permitted to continue working, although perhaps with reduced functionality. When net connectivity is restored, a systems challenge is the required state resolution necessary to make everyone's world view consistent. Our current work is also investigating responses to this situation.

As a final comment, we expect to encounter situations in which it is desirable to combine protocol fragments or in which separate teams, carrying out similar activities but operating with independently-developed protocols, wish to cooperate together but without changing their individual operating

procedures. Consequently we are also examining means to permit the automatic combination of protocols.

5 CONCLUSION

In this paper, we have demonstrated Trellis-based encodings of protocols in the application domains of software engineering and computer supported cooperative work. Furthermore we have asserted that the resulting specification with its basis in a dynamic, analyzable hypertext specification mechanism, as is the Trellis representation, exhibits useful characteristics attributable to the underlying representation.

Hypertext has traditionally been viewed as a means of structuring information. The representations of protocol described in this paper represent an extension of that view to also encompass the structuring of *process*. However, since the information structuring mechanisms remain, documentation of the protocol and of the structure can be carried along with the specification of the protocol itself. It is natural and straightforward to associate additional documentation with the nodes representing the protocol definition, or indeed to define auxiliary structures to provide more precise annotation of the structure.

The Trellis net-based representation provides a means for specifying a protocol that can be used directly for verification, training, and simulation. When the hypertextual basis is dynamic, it also provides the means for rapid prototyping of the protocol. On the other hand, the visual presentation of the net requires attention be paid to the complexity of the image, requiring investigation of structuring through abstraction and hierarchy.

Finally, we close by noting that the Trellis implementation decision to interpret the underlying net suggests experimentation with an incremental methodology in protocol development. As prototyping proceeds, more sophisticated behaviors can be added and refined. There is a nice symmetry between this idea and the world-view of a hypertext as a dynamic web, growing and developing as new paths are encountered and envisioned.

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] J. M. Atlee and J. D. Gannon. State-based model checking of event-driven system requirements. *IEEE Transactions on Software Engineering*, pages 24–40, January 1993.
- [4] James Bigelow. Hypertext and CASE. *IEEE Software*, 5(2):23–27, March 1988.
- [5] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8:244–263, 1986.

³Recall that agents are either humans, computer processes, or both.

- [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] H. J. Genrich and K. Lautenbach. System modeling with high-level Petri nets. *Theoretical Computer Science*, 13:109–136, 1981.
- [10] Carlo Ghezzi, Medhi Jazayeri, and Dino Mandrioli. *Fundamentals of Software Engineering*. Prentice Hall, 1991.
- [11] Dong He. Automatic layout for petri-net-based hypertext. Master's thesis, Interdisciplinary Applied Mathematics Program, University of Maryland, 1992.
- [12] Kathryn L. Heninger. Specifying software requirements for complex systems: New techniques and their application. *IEEE Transactions on Software Engineering*, 16(1):2–13, January 1980.
- [13] 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.
- [14] Kurt Jensen. Coloured Petri nets and the invariant-method. *Theoretical Computer Science*, 14:317–336, 1981.
- [15] Michael Liu. mTrellis: A Petri net browser and its application in browsing structured graphs. Master's thesis, Department of Computer Science, University of Maryland, 1993.
- [16] Philip M. Merlin. *A Study of the Recoverability of Computing Systems*. Ph.D. dissertation, University of California at Irvine, Department of Information and Computer Science, Irvine, CA, 1974. Also available as Technical Report 58, Department of Information and Computer Science, University of California at Irvine (1974).
- [17] 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.
- [18] Tadao Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.
- [19] W. Reisig. Petri nets with individual tokens. *Informatik-Fachberichte*, 66(21):229–249, 1983.
- [20] Wolfgang Reisig. *Petri Nets: An Introduction*. Springer-Verlag, 1985.
- [21] Tina Roth and Peter Aiken. Hypertext support for software development: A retrospective assessment, 1994. Internal report.
- [22] Steven R. Schach. *Software Engineering*. Asken Associates, second edition, 1993.
- [23] P. D. Stotts and R. Furuta. Modeling and prototyping collaborative software processes. In *Proceedings of the NATO Advanced Research Workshop on Integration of Information and Collaboration Models*, June 1993. To appear. Also published as Technical Report TR93-020, Computer Science Col-laboratory, 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.
- [24] 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.
- [25] 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.
- [26] P. David Stotts and Richard Furuta. Temporal hyperprogramming. *Journal of Visual Languages and Computing*, 1(3):237–253, 1990.
- [27] P. David Stotts and Richard Furuta. Dynamic adaptation of hypertext structure. In *Third ACM Conference on Hypertext Proceedings*, pages 219–231. ACM, New York, December 1991.
- [28] P. David Stotts, Richard Furuta, and J. Cyrano Ruiz. Hyperdocuments as automata: Trace-based browsing property verification. In D. Lucarella, J. Nanard, M. Nanard, and P. Paolini, editors, *Proceedings of the ACM Conference on Hypertext (ECHT '92)*, pages 272–281. ACM Press, 1992.
- [29] Willem R. van Biljon. Extending Petri nets for specifying man-machine dialogues. *International Journal of Man-Machine Studies*, 28:437–455, 1988.