CONTEXT-AWARE HYPERMEDIA IN A DYNAMICALLY CHANGING ENVIRONMENT, SUPPORTED BY A HIGH-LEVEL PETRI NET

A Dissertation

by

JIN-CHEON NA

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

December 2001

Major Subject: Computer Science

CONTEXT-AWARE HYPERMEDIA IN A DYNAMICALLY CHANGING ENVIRONMENT, SUPPORTED BY A HIGH-LEVEL PETRI NET

A Dissertation

by

JIN-CHEON NA

Submitted to Texas A&M University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Approved as to style and content by:

Richard Furuta (Chair of Committee)	
John J. Leggett (Member)	Frank M. Shipman III (Member)
John D. Oswald (Member)	Jennifer Welch (Head of Department)

December 2001

Major Subject: Computer Science

ABSTRACT

Context-Aware Hypermedia in a Dynamically Changing Environment, Supported by a High-Level Petri Net. (December 2001)

Jin-Cheon Na, B.S., Hanyang University;

M.S., Oklahoma State University

Chair of Advisory Committee: Dr. Richard Furuta

In modern hypertext systems, with the increased availability of personal computers with environmental sensors (e.g., GPS sensors), adaptation support to meet the needs of the users in dynamically-changing environments will become essential. Hypertext systems supporting computing devices with environmental sensors will therefore benefit if they can also become *context-aware*. In this research, we introduce caT (for Context-Aware Trellis), a context-aware hypertext model with associated tools, which supports flexible user (or agent) adaptation to changes in environmental information, such as time, location, bandwidth/cost, etc. The context-aware hypertext model incorporates high-level Petri nets, context-awareness, user-modeling, and fuzzy knowledge handling features into the previously-described Trellis hypertext model. Major features of the high-level Petri net that are explored are its hierarchical net, structured tokens, and flexible net description. Fuzzy knowledge (context) handling is supported by the integration of a fuzzy logic tool into the Petri net. In caT, the author who is accustomed to the caT model and its underlying formalism can specify various dynamic documents using the structured GUI (Graphical User Interface) authoring tool. Afterward, the

reader, who does not necessarily need to know the model, can browse the dynamic documents using various browsers, including Web browsers. The browsing behavior of documents specified in the caT model can reflect the reader's contextual (such as location and time) and preference information. To show the usability of caT, we introduce several of its prototype applications that represent various documents such as customized Web documents, formal specifications of software requirements, and cooperative documents. Since caT is based on a formal model, the behavioral characteristics of developed caT models can be analyzed. An analysis tool, which is integrated into the authoring tool, supports the verification of application models. The successful result of the user evaluation showed the potential usability of the model as a context-aware hypertext system. With the increased availability of computing devices equipped with environmental sensors, further research work will enhance the usability of the caT system.

To my parents and wife and daughter for their love, patience, and encouragement.

ACKNOWLEDGMENTS

I could not have accomplished my doctoral degree without the support of many great scholars at Texas A&M University. First of all, I would like to express my deep gratitude to my committee chair, Dr. Richard Furuta. His invaluable advice and patience allowed me to undergo and finish the challenging process of my doctoral degree program. I would also like to express my appreciation to my other committee members, Dr. John J. Leggett, Dr. Frank M. Shipman III, Dr. John D. Oswald, and Dr. David L. Busbee, for their valuable comments and time. I also would like to thank the graduate students, who work with the Center for the Study of Digital Libraries (CSDL), for their participation in the experiments of evaluating the prototype software system that provides a major part of my dissertation material. Finally, I would like to thank all other graduate students in CSDL, whose advice and friendship gave me a lot of encouragement for my study at Texas A&M University.

TABLE OF CONTENTS

Page

Page

Page

Page

LIST OF FIGURES

Page

Page

LIST OF TABLES

Page

CHAPTER I

INTRODUCTION

1.1 Context-awareness in the computing world

When humans converse with humans, they are able to use implicit situational information, or context, to enhance the conversational bandwidth. Unfortunately, this ability to convey ideas does not transfer well to humans interacting with computers. By providing computer applications with access to the human's context, we increase the richness of communication in human-computer interaction and make it possible to produce more useful computational services. Dey and Abowd [1999a] define context in the following way: *"Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves."*

Due to the increased availability of personal computers with attached sensors that capture data about their physical environment, a new range of applications called context-aware applications [Schilit et al. 1994] became attractive. These applications, using contextual information, provide an opportunity to customize the provision of information to reflect the user's current needs. Context-aware applications typically

---

The journal model is *ACM Transactions on Information Systems*.

focus on a mobile user who is carrying a mobile system such as a Personal Data Assistant (PDA) that incorporates environmental sensors such as GPS receivers, electronic compasses, etc. These environmental sensors could identify location, time, identification, and companions or objects nearby, and they could detect if the user is stationary or in motion. Additionally, super sensors that correlate information from lower-level sensors in order to deduce some higher-level state could identify more complex situation data; for example, whether the occupant of an office is having a meeting. In the near future most mobile systems are expected to be context-aware.

In context-aware applications, *location*, *identity*, *time*, and *activity* are the primary context types for characterizing the situation of a particular entity [Dey and Abowd 1999a]. This context information not only answers the questions of who, what, when, and where, but also provides other sources of contextual information indirectly. For example, knowing an entity's location, we can determine what other objects or people are near the entity and what activity is occurring near the entity.

Over the last decade, some researchers have built context-aware applications that take advantage of environmental information to provide better interaction with the user. Some examples of context-aware applications are as follows:

- Office awareness systems [Hodes and Katz 1999; Want et al. 1992; Want et al. 1995] sense users' locations, help people find each other, and keep up their location awareness or forward phone calls to the correct person. Teleporting

[Bennett et al. 1994] automatically brings up users' desktop on nearby computers as users move around.

- Mobile tour guides [Abowd et al. 1997] familiarize a visitor with a new area.
- Context-based retrieval applications [Rhodes 1997] collect and save context information and support subsequent information retrieval based on context information. Conference Assistant [Dey et al. 1999b], which supports conference attendees and presenters, assists users in taking notes on presentations and aids in the retrieval of conference information after the conference concludes.
- Brown [Brown 1998a; Brown 1998b; Brown et al. 1997] provides a framework for discrete context-aware applications and services based on a simple *Post-It Note* metaphor, where discrete pieces of information (called stick-e notes) are attached to individual contexts. The stick-e notes are then triggered when the user enters those contexts. In addition, Brown [Brown 1998b] suggests the following applications: equipment maintenance software shows field engineers the equipment information they need to be aware of; farm management software shows farmers the information attached to tagged farm animals or tagged equipment when the object of interest is around; real timetables for public transport guide travelers to the nearest stop so that they arrive shortly before the public transport vehicle does; office equipment control software shows the user personally tailored control panels on a PDA as he (or she) passes office

equipment, such as a photocopier, and the panels allow the user to control the equipment.

- In her book *Plans and Situated Actions* [1987], Suchman argues that plans are resources for situated action, but do not in any strong sense determine its course. Actions are always situated in particular social and physical circumstances; the situation is crucial to action interpretation. Suchman introduces a concept of a user model as a representation of the user and his (her) situation, and intelligent tutoring systems as applications using user modeling.

Most of the context-aware systems require detecting, interpreting, and responding to context. In other words, the context-aware systems have a responsibility to detect and interpret context information, in addition to providing customized information to reflect the user's current needs. However, Dey and Abowd [1999a] propose a narrower view of context-aware systems: *"a system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task"*. They argue that context-aware systems are required only to respond to context, allowing the detection and interpretation to be performed by other computing entities. When the burden of sensing (and distributing) context information moves to other special computing entities, it will be easier to build context-aware applications.

As one of these efforts, Salber et al. [1999] introduce the context toolkit. This supports the concept of context widgets, which mediate between the environment and the application in the same way graphical widgets mediate between the user and the

application. Therefore, the approach to context-aware application development is to collect contextual information through automated means, make it easily available to a computer's run-time environment, and let the application designer decide what information is relevant and how to deal with it. The user may not be required explicitly to express all information relevant to a given situation. When context-aware building tools like the context toolkit are commonly available, we expect to see more varied and interesting context-aware applications.

1.2 Context-aware information delivery in hypertext

Vannevar Bush is credited with first describing the hypertext concept in his article "As We May Think" [Bush 1945]. He introduces a machine he calls the Memex for browsing and making notes in an extensive on-line text and graphics system. The Memex is for personal knowledge workers; with it they do their own knowledge work. The essential feature of the Memex is the ability to tie related items together while they are doing their knowledge work.

The basic concept of hypertext is fairly simple: windows on the screen are associated with objects in a database, and links are provided between these objects [Conklin 1987]. Recently, due to the popularity of the WWW (World-Wide Web), which is a kind of simple hypertext system compared to other hypertext systems (such as KMS [Akscyn et al. 1988], HAM [Campbell and Goodman 1988], and Trellis [Furuta and Stotts 2001]), hypertext has been more widely accepted.

In modern hypertext systems, with the increased availability of personal computers with environmental sensors, adaptation support for users in dynamically-changing environments will become essential to meet their needs. Hypertext systems supporting computing devices with environmental sensors will thus benefit if they also can become *context-aware*. Hypertext systems can be context-aware application building tools with the incorporation of dynamically-changing information from the external environment.

A focal point in adaptive hypermedia systems [Brusilovsky 1996] has been support for user modeling. Users with distinct goals and knowledge may be attracted to different portions of information displayed on a hypermedia page and may use different links for navigation. The adaptive hypermedia field endeavors to conquer this problem by using knowledge represented in the user model to adapt the information and links being presented to the given user. Providing context-awareness will require user modeling. However, it also will require the incorporation of dynamically-changing information from the external environment. In addition to reflecting individual differences, information providers may wish to tailor presentation to user location, access time, network bandwidth/cost, etc.

For example, we can think of a context-aware application, a university digital library, that provides a full version of a Web page to its on-campus patrons and a limited one to its off-campus patrons. However, off-campus faculty members retain access to the full version. Because the page includes a real-time help desk, only the limited version is available outside of normal operating hours. Some databases are only made available for

general use outside of working hours. Patrons inside of the physical library building are given more detail for reference material located on the current floor than for other floors.

1.3 Formal model-based hypermedia systems

The model implicit in many hypermedia systems (e.g., Neptune [Delisle and Schwartz 1986]) is the *labeled directed graph*, where the information content of the application is associated with the contents of a set of nodes, and the labels of the arcs are associated with the user's logical selections. Selecting a label follows the arc(s) associated with the label, resulting in the change of the display of the hypertext.

Some formal model-based systems, including the Trellis project [Furuta and Stotts 2001; Stotts and Furuta 1989a], investigate the use of automaton-based specification in defining hypertextually-based interaction. The behavior of the hypertext is described by the *browsing semantics* of the specification—i.e., the contents displayed and links presented are tied to the state changes that occur as the automaton is executed. A primary focus in the automaton-based systems has been to generalize the browsing semantics of a document and to move implementation of such mechanisms into the document (or hypertext) specification rather than the system implementation.

The automaton-based models provide good abstraction and separation of the structure from the content of hypertext, using them helps to discipline the authoring activity by encouraging development in a structured fashion. This development is accomplished by requiring that the structure is designed before the actual contents are

associated to the nodes in the structure [De Oliveira et al. 2001]. In addition, the models that are based on automaton formalisms, such as Petri nets and Statecharts, support systematic analysis techniques. Authors can use the analysis techniques to verify and validate the behavioral characteristics (e.g., reachability of certain information) of developed documents before the materials are handed over to users.

However, most graph-based models suffer from graphical complexity. Therefore, specification simplification efforts that reduce the graphical complexity are necessary. In addition, a noticeable cost of using automaton-based systems is that the author interacting with such a system must be accustomed to the model and its underlying formalism. Good interface design and proper training may reduce the cost of using the automaton-based systems.

1.4 The approach and corpus of our choice

The main goal of this research is to develop caT (called “caT”, for Context-Aware Trellis) [Na and Furuta 2000a; Na and Furuta 2000b; Na and Furuta 2001], a context-aware hypertext model and associated tools. caT is a formal-model-based hypertext system that provides adaptation support for users in dynamically-changing environments.

In developing caT, we extend the existing Trellis hypertext system to concentrate more on new interesting features, such as context-awareness, than on the other intrinsic hypertext features. Trellis was chosen since it provides excellent facilities and potential

extensibility for dynamic adaptations of multiple readers, and supports formal analysis techniques for detecting inconsistencies in the specification. The Trellis project was initiated in the late 1980's to investigate the use of the automaton formalism (colored timed Petri nets [Jensen 1992; Zurawski and Zhou 1994]) in defining hypertextually-based interaction.

Beginning with the Trellis system, the following main research issues are explored: a context-aware hypertext model, fuzzy knowledge handling, flexible information presentation, structured authoring, verification and validation, and improvement of the Trellis implementation. Detailed descriptions of the above issues are presented in the following subsections.

1.4.1 A context-aware hypertext model

Mobile computing environments raise an additional requirement for hypertext systems—namely, the need to be able to reflect the current context of use in the structure and content of the hypertext. For example, the relevance of information may change as the reader moves from location to location (e.g., the location of the nearest gas station). Relevance may reflect time of day, speed of connection, the administrative categories corresponding to a system's current user (e.g., faculty, staff, or student), the proximity of other, interacting devices, etc. A main goal of this research, therefore, has been to support specification of context aware hypertexts.

Even though conventional hypertext models provide a powerful modeling framework, they are weak when supporting context-aware adaptation. For supporting

context-aware adaptation, they need incorporation of dynamically-changing information from the external environment. A focus of adaptive hypermedia systems has been support for user modeling, but this focus has been more on static personal preference information than dynamic context information from the external environment.

We need a new context-aware hypertext model that supports context-awareness in dynamically-changing environments. Consequently, we have extended the Trellis hypertext model in support of context-awareness. For incorporation of the external environmental data, caT provides hooks that allow hypertexts to reflect changing, external events. However, handling the instrument that detects those events remains outside of the caT system, as Dey and Abowd [1999a] suggest.

1.4.2 Fuzzy knowledge handling

Fuzzy logic [Zadeh 1989] primarily is motivated by observing that human reasoning can utilize concepts and knowledge that do not have well defined, sharp boundaries (i.e., vague concepts). If we integrate the fuzzy logic concept into the hypertext model, the new model that results will have the ability to handle uncertain knowledge in hypertext applications. For example, the simple link display decision is based on showing or not showing (i.e., true or false) for all users; but we may need the following link display condition: *show a link when the access right of the current user is rather high*. In this case, we may need a fuzzy rulebase for inferring the *access right* value from the current user's information. For example, one of the rules will be as follows: the *access right* value is *high* when he (or she) accesses the system at *daytime* and is *close* to the location

of the system. Fuzzy knowledge handing will help to represent and infer the user's vague contextual information more naturally. For these reasons, we have integrated an existing fuzzy logic tool [The Mathworks, Inc. 1999] into the Petri net structure.

1.4.3 Flexible information presentation

Since the Trellis (i.e., caT's parent system) implementations are based on a client-server architecture, several client browsers can be active simultaneously, each presenting different interpretations of the document and its browsing state as stored in the server. This feature provides good separation between document specification and presentation. Using this characteristic of the system, new browsers have been developed in this research, such as an image browser and a Web-based information presentation tool for Web browsers, in order to allow multiple presentations of a particular document's specification.

The early Trellis clients assume implementation directly within the windowing environment that provides the primary user interface for the hypertext's reader. Consequently, interface tasks within the Trellis prototype system are distributed across multiple clients and multiple windows. For example, one client may have the task of playing audio while another has the task of displaying images. Commonly, the clients that display textual content display each of the multiple elements that may be active simultaneously in separate windows. A separate window, which logically could be considered to be associated with its own client process, allows invocation of links.

The architecture of the World-Wide Web suggests instead that the clients' interaction with the users is to be funneled through the intermediary of a HTTP server. The target presentation is within the context of a Web browser, and not in the context of a windowing system. To maintain consistency with the World-Wide Web's user interface paradigm, we define a mechanism to create a composite node from the active content elements and to embed the link anchors that correspond to firable transitions (i.e., links).

As a result, the Web-based information presentation of the caT model will allow us to define sophisticated hypertext applications in caT and to browse the resulting information using the familiar user-interface of WWW browsers such as Netscape Navigator. We may say that the support for browsing with Web browsers provides a connection between the caT hypertext model and the WWW to get synergy from the two different models. Web-based flexible information presentation will increase the usability of caT.

1.4.4 Structured authoring

Structured authoring (top-down or bottom-up) support will be essential for developing large-sized Petri net applications since it reduces the graphical complexity problems that are common to most graph-based modeling tools. Thus, in the Petri net research field, hierarchical Petri nets (extensions of basic Petri nets) have been developed to support authoring of multiple Petri nets that are hierarchically linked [Jensen 1992]. With the hierarchical net, the author can build hierarchically structured subnets in a top-down or bottom-up way. However in terms of the authoring of both the nets (i.e., document

structures) and their annotations (i.e., contents and links), caT more strongly supports a top-down design, in comparison to the bottom-up design common in Web document authoring. In caT, authors initiate their design by specifying a document structure (i.e., a Petri net); in the Web, authors create fragments of content, and include pointers to other pieces of content within these fragments.

Concerning Trellis, one immediate approach to reducing graphical complexity problems is to incorporate notions of stepwise decomposition (i.e., hierarchy) into the specification, thereby providing a means by which significantly-sized specifications can be defined in manageably-sized pieces. Trellis' Graph Grammar approach [Stotts and Furuta 1990] is a step in this direction, but it was developed separately and not incorporated into the general Trellis prototypes. The mechanism provided in the prototypes, namely the inclusion of hypertexts as a permissible content element, does introduce a notion of hierarchy but strongly separates the state of the content-embedded element from that of the parent net.

In Trellis, a Petri net engine (i.e., an information server that receives client requests for its service) holds only one net, and hierarchical subnets are handled by the browser (i.e., client-side programs). Therefore, for hierarchical subnet support, multiple Petri net engines and browsers are involved in a loosely-coupled fashion. It is necessary to shift the responsibility for subnets from the browser to the Petri net engine, in essence combining the nets into a single unit. In other words, the Petri net engine should be

extended to hold multiple hierarchical subnets. Consequently, caT extends the Petri net engine in the Trellis model to support hierarchical Petri nets.

In addition, the current Trellis authoring tool is a single-window-based tool, which does not support hierarchical subnet editing facilities such as easy creation of multiple subnets (or sub modules) and easy browsing between parent and child nets. For supporting flexible structured authoring, a multiple-window-based authoring tool is developed to allow building and browsing of hierarchical subnets. The author, using the authoring tool, first builds a Petri-net-based hierarchical document structure and then associates net segments with the corresponding content segments.

1.4.5 Verification and validation

One of the major advantages of using Petri net models is that the same model is used for the analysis of behavioral properties as for the systematic construction of discrete-event systems [Zurawski and Zhou 1994]. Authors can use the analysis techniques to verify and validate the behavioral characteristics (e.g., reachability of certain information) of developed systems before the systems are handed over to users. To support this feature, at first we study the conformance of the extended model to current analysis techniques, and then develop and integrate analysis tools into the authoring tool.

1.4.6 Improving the Trellis implementation

Trellis has been viewed favorably by the hypertext research community (see, for example the comments in Frank Halasz's 1991 Hypertext Conference keynote speech

[Halasz 1991]). However, it has not received widespread use, even though it has been freely available via FTP in binary form since the mid-1990's. This relative lack of use seems due to a number of factors that, combined, make it inconvenient to use the system, especially in the current computing environment. These factors, such as the user interface and running environment, are identified and managed in the development of caT.

1.5 Contributions of this research

caT, a context-aware hypertext model and associated prototype tools, has been designed and built in order to support flexible adaptation in a dynamically-changing environment through extension of the earlier Trellis model.

In summary, the major contributions of this research work are as follows:

1. Introduce a new context-aware hypertext system for the development of context-aware hypertext documents.
    - Provide a flexible net specification with adaptive behavior by incorporating high-level Petri nets, fuzzy (or uncertain) knowledge handling, and user modeling features into the Trellis model.
    - Provide hooks that allow hypertexts to reflect changing, external events (dynamically-changing environmental information).
2. Present mechanisms for reducing complexity of specification representation in Petri net-based hypertext systems.

- Provide mechanisms for simplifying the specification representation by introducing hierarchical Petri nets and structured tokens (tokens with local variable/value pairs) features to the Trellis model.

3. Support flexible information presentation of a document specification.
    - Define a mechanism to create a composite node from the active content elements, and as well embed the link anchors for Web-based information presentation of the caT model.
    - Develop (or support) multiple browsers, such as a text browser, an image browser, and Web browsers, to allow multiple presentations of a particular document's specification.
4. Develop structured authoring and analysis tools for the high-level Petri net-based hypertext system.
    - Develop a multiple-window-based structured authoring tool for the specification of reasonably large and complex documents that are linked hierarchically.
    - Develop an analysis method for the extended nets and build an analysis tool that is integrated into the authoring tool
5. Improve the Trellis implementation.
    - Enhance the user interfaces and the functionality of the components of the Trellis implementation, as well as its porting to Linux

The caT system also provides a good environment for the specification and hypertextual browsing of various documents, even though it has been mainly targeted at context-aware hypertext documents. Thus, for investigating the potential usability of the system, we have built various hypertext documents: context-aware documents, adaptive documents, hierarchical documents, CSCW protocol specifications, software requirement specifications, etc.

Finally, we have performed an informal evaluation of the proposed system. The result of this successful evaluation shows the potential usability of context-aware hypertext systems. We expect that the experience and the results gained from this work will provide some background techniques and experiences for further context-aware research in hypertext and related fields. With the increased availability of computing devices equipped with environmental sensors, further research will improve the usability of the context-aware hypertext system.

CHAPTER II

REVIEW OF PREVIOUS WORK

2.1 High-level Petri net

Petri nets [Jensen 1992; Peterson 1981; Zurawski and Zhou 1994], as graphical and mathematical tools, provide a uniform environment for modeling, formal analysis, and design of discrete event systems. Formally, a basic Petri net is a bipartite directed graph defined as follows (the notation used here is taken from [Reisig 1985; Stotts and Furuta 1989a]):

A Petri net structure is a tuple, <P, T, F>, in which:

- $P = \{p_1, p_2, \ldots, p_n\}$ is a finite set of places with $n \geq 0$;
- $T = \{t_1, t_2, \ldots, t_m\}$ is a finite set of transitions with $m \geq 0$ and $P \cap T = \varnothing$;
- $F \subseteq (P \times T) \cup (T \times P)$ is the flow relation, a mapping representing arcs between places and transitions.

The arcs denoted by F set pre- and post-relations for places and transitions. The set of places that are incident on a transition *t* is called the preset of *t*, and is represented by

$$\bullet t = \{p \mid (p,t) \in F\}.$$

The set of places that come after a transition *t* is called the postset of *t*, and is represented by

$$t\bullet = \{p \mid (t,p) \in F\}.$$

For a marking of a Petri net structure <P, T, F>:

- $M : P \rightarrow I$, $I = \{0, 1, 2, \ldots\}$, is a function that associates a marking with each place in the net.

Each integer in a marking specifies the number of tokens existing in the corresponding place. An execution of a Petri net generates a series of markings, starting with the initial marking M_s and finishing in some final marking M_f. To move from the current state M to some next state M', any transition *t* that is enabled under M is chosen and fired. For a transition *t* to be enabled under the marking M, it must satisfy the following condition that is represented by

$$\forall\, p \in {\bullet t} : M(p) \geq 1.$$

Firing *t* involves of deleting one token from each place in •t and adding one token to each place in t•—we assume that the weight on each arc is 1. The next state M' is then the vector of integers indicating the number of tokens in each place after the firing.

Graphically, a Petri net is represented by indicating its places by circles, its transitions by bars, its arcs by arrows, and its tokens by small black dots. A place containing one or more tokens is said to be marked. When each place incident on a transition is marked, that transition is enabled. For example, consider the marked Petri net shown in Figure 1. The initial marking is $M_0 = [110]$ (i.e., $P_1 = 1$, $P_2 = 1$, $P_3 = 0$), and T_1 is an enabled transition under M_0. If T_1 were fired, the resulting next state would be $M_1=[001]$, as shown in the right side of the figure.

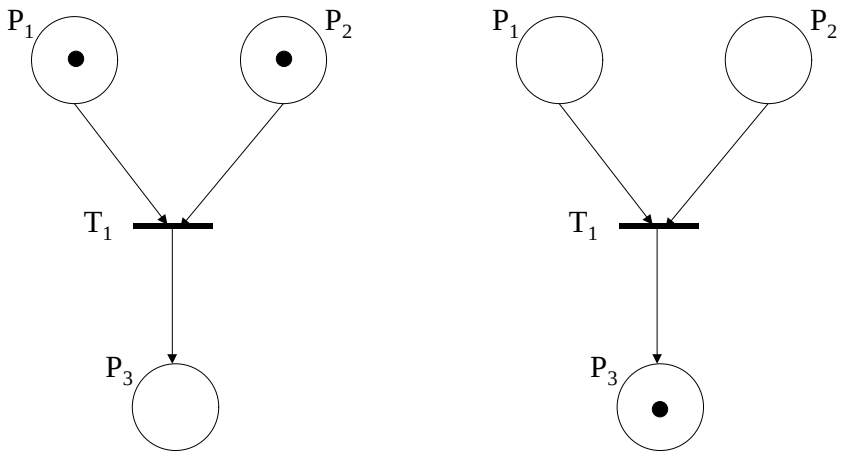


Figure 1: A simple Petri net

The basic Petri net is not always convenient for representing and analyzing complex systems because tokens in the basic Petri nets have no identity. This poses problems for modeling systems, which demand that physical resources or messages, if represented by tokens, have identity. Without this identity, it is unrealizable to trace the flow of different resources or messages in the system. A likely solution is to build a model in such a way that the flow of each resource or message is associated with a separate subnet. Because the resources or messages share, in most cases, the same system, all these subnets are duplicates. This approach increases the graphical complexity of the model [Zurawski and Zhou 1994].

In order to overcome this complexity problem, Petri nets that allow for tokens that have distinct identity, called high-level Petri nets, were proposed—these mechanisms may give identity to a group of tokens not necessary to individual tokens. These nets include predicate-transition nets [Genrich and Lautenbach 1981], colored Petri nets [Jensen 1992], and object-oriented Petri nets [Gudwin and Gomide 1998; Sibertin-Blanc 1985]. In high-level Petri nets, a token can be a composite object carrying data. This data can be of arbitrary complexity involving integers, reals, text strings, records, lists, and tuples. Nonetheless, it should be noted that ordinary and high-level Petri nets have the same descriptive power, even though high-level Petri nets supply much better structuring facilities than basic nets [Zurawski and Zhou 1994]. A more thorough exposition of net theory can be found in the texts by Peterson [1981] and Jensen [1992]. The caT model extends the Trellis model by introducing to it some features of the high-level Petri nets such as tokens carrying data.

2.2 Fuzzy Petri net

The recognition of the need for qualitative specification of the system, as well as the need for representing approximate and uncertain information, has led to the development of various types of Fuzzy Petri Nets (FPN). Since 1988, with the work of Looney [1988], several authors from the Petri net and AI communities have proposed different kinds of FPN [Cardoso and Pradin-Chézalviel 1997; Cardoso et al. 1996]. FPN, combining fuzzy logic and Petri net theory, aims at solving two fundamental issues:

allowing the representation of the evolution during the time, given by Petri nets, and the representation of uncertain knowledge about a system state, given by fuzzy logic.

Although sharing the same name, Fuzzy Petri Net, the models that have been described are based on different notions. Their variances first are related to the fuzzy tools used (fuzzy logic, possibility theory, etc.) and then related to which elements of the Petri nets are fuzzyfied (tokens, markings, and transition thresholds). Some of these FPN systems are used to represent fuzzy expert systems [Chen et al. 1990]. In these cases, Petri nets are used to describe formally the rule chaining mechanisms (i.e., reasoning algorithms) with a marking corresponding to some step within the reasoning; most approaches associate a logical proposition with any place of the Petri nets. The benefit of this approach is that concurrency (several rules may be applied in any order) and choices (selections of one rule in a set of conflicting rules) can be formally captured. In another class of application [Cardoso et al. 1990], a FPN is used to represent directly imprecise or vague knowledge. The FPN represents a dynamic system, and the marking denotes the knowledge about its state at some time point. Another uncomplicated way of supporting uncertain knowledge in Petri nets is integrating an existing fuzzy logic tool (or a fuzzy expert system) with current Petri net structures. Here, the Petri net can interact with the fuzzy logic tool to handle uncertain knowledge. The caT model follows this approach.

2.3 Formal model-based hypermedia systems

In the following subsections, Petri-net-based hypermedia systems (including Trellis) are introduced, followed by statechart-based hypermedia systems.

2.3.1 Petri-net-based systems

2.3.1.1 Trellis

The Trellis project was initiated in the late 1980's [Furuta and Stotts 1989; Stotts and Furuta 1989a] to investigate the use of automaton-based specification in defining hypertextually-based interaction. Essentially, the Trellis model is based on a colored timed Petri net as the structure of a hyperprogram, and the behavior of the hypertext is described by the browsing semantics of the specification—i.e., the contents displayed and links presented are tied to the state changes that occur as the automation is executed.

The underlying information engine supporting Trellis is related to other hypertext engines that have been used in experimental software support systems, such as HAM (Hypertext Abstract Machine) [Campbell and Goodman 1988]. The uniqueness of Trellis is its foundation on a parallel collaborative computation model, i.e., colored timed Petri nets. This gives the model an elegant structure that can be both programmed and analyzed. In the colored timed Petri nets (CPN) of the Trellis model, tokens have type (color), and a token of one color is discernible from a token of another color. However, within a color class, individual tokens cannot be distinguished from one another.

In Figure 2, each of the three tokens in places P_1 and P_2 has a different color. The notation on the arc from P_1 to T_1 defines a variable, a. The color of the token extracted

from P_1 when T_1 is fired matches the color of the token deposited back into P_1. Similarly, the color of the token extracted from P_2 matches that deposited into P_4 (either of the tokens in P_2 will satisfy this condition). Because a constant value is used on the arc from T_1 to P_5, the color of the token deposited there always is *"red"*. Similarly, a specific color could have been required on the input arc. Both arcs leading into transition T_2 have been notated with the same variable name, *c*; hence the color of the tokens from P_2 and P_3 consumed when T_2 fires must match.

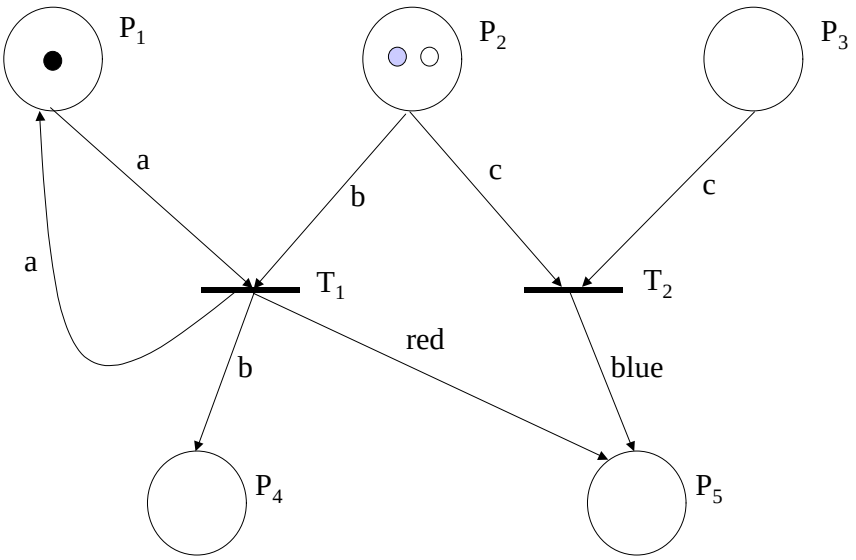


Figure 2: A colored Petri net example

Each transition in a Trellis net has two time values, which represent, roughly, a delay and a timeout. Time values in Trellis are thought of as defining ranges for the

availability of an event. The first value in the pair describes a delay that must pass between when the transition is enabled and when the system allows its firing. The second value, greater than or equal to the first, describes the maximum amount of time that will be allowed to pass before the system fires the transition itself. Thus the time (0, ∞) represents a transition that can be fired immediately and never fires itself. Other examples include (0, 5), a transition that fires itself after 5 time units; (5, ∞), one that requires a delay of 5 time units; and (0, 0), one that fires itself immediately on being enabled.

Formally, a colored timed Petri net in the Trellis model is defined as follows [Stotts and Furuta 1989a]:

A CPN is a tuple, $<\Sigma, P, T, F, \tau>$, in which:

- Σ is a finite set of token colors, such as {black, maroon, …};
- P, T, and F are as defined earlier ;
- $\tau : T \rightarrow \{0, 1, 2, \ldots, \infty\} \times \{0, 1, 2, \ldots, \infty\}$ 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 $\tau(t) = (\tau^r, \tau^m)$ and we require that $\tau^r \leq \tau^m$.

A Trellis net is completed by annotating for the components of the CPN. We can consider the CPN as the task description and the different annotations as the information needed by the task. One main category of annotation is content. Fragments of information (text, graphics, video, audio, executable code, and other hyperprograms) are

connected with the places in a CPN. Another category of annotation is events, which are mapped to the transitions of the CPN. A third category of annotation is the attribute/value (A/V) pair. A list of A/V pairs is associated with each place, each transition, each arc, and the CPN as a whole. Annotations can be used by the Trellis implementation to build client tools such as hypertext document browsers and authoring tools. In the Trellis implementations, information presentation is based on a simple paradigm; namely, every node of the Petri net is shown in a different window.

A hypertext in the Trellis structure [Stotts and Furuta 1989a] is a tuple, <CPN, M_0, D, W, B, P_l, P_d>, in which:

- CPN is a colored timed Petri net <Σ, P, T, F, τ>;
- $M_0 : P \rightarrow$ "token instances of Σ" is an initial marking (or initial state) for CPN;
- D is a set of document contents;
- W is a set of windows;
- B is a set of buttons (or links);
- P_l is a logical projection for the document, P_l = <D_l,W_l,B_l>, mappings from components of a Petri net to the human-consumable portions of a hypertext;
- P_d is a display projection for the document, a collection of mappings that associate the logical buttons and windows of a hypertext with physical screen representations and locations.

The Trellis prototype implementations are based on a client-server architecture, which accomplishes cooperative separation between net and interpretation. The central

part of the server is the Petri net engine, which implements Petri net execution semantics. The Petri net engine is wrapped with the hypertext engine, which interprets the state of the Petri net into a hypertext-centric representation. Normally, this denotes that the hypertext engine constructs a list of active content elements (i.e., those associated with marked places) and their related active links (i.e., corresponding to transitions that are enabled for firing). Every Trellis model is an instance of an information server, an engine that services remote procedure call (RPC) requests. The engine has no visible user interface, but does have an API that allows other remote processes to call its functions for building, editing, annotating, and executing a CPN.

Interface clients are separate processes that have visual user interfaces and communicate with one or more engines via RPC. Clients collectively provide the necessary views, interactions, and analyses of a net for some specific application domain. One or more client applications provide the interface with the user and with other interacting agents. Hypertext clients communicate with the hypertext engine and have the responsibility for determining how to display the active content. The implemented clients often specialize. For example, they display only textual or only audible content. Consequently, a complete user interface can require use of several clients. Additionally, separate clients can provide different views of the content for different execution environments—for example, on a text-only display or on a graphical display. A second category of clients, those that display and permit editing of the Petri net representation, communicate directly with the Petri net engine.

The initial implementation of Trellis, αTrellis, supported a basic Petri net (i.e., without colored tokens) along with the timing facilities. In addition, later versions of αTrellis included a Lisp interpreter and allowed the association of Lisp fragments with transitions [Stotts and Furuta 1991]. The fragments were executed when the transition was fired, and could modify values such as those associated with timings. Due to changes in the available computing environment, Stotts and his students implemented a second version of Trellis called χTrellis [Furuta and Stotts 1994a]. χTrellis provided a cleaner separation of Petri net engine from other program functions and added colored tokens. However, χTrellis did not retain the Lisp interpreter and associated transition actions.

Trellis has been used in a wide variety of applications, ranging from hypertextual applications to protocol specification. One particular example of protocol specification is the description of collaborative protocols [Furuta and Stotts 1994a]. The Trellis architecture allows multiple people to interact via a common net through clients on their individual workstations. Subsequent applications include Stotts et al.'s separation and application of multi-headed/multi-tailed links [Ladd et al. 1995], and their application to educational protocols [Capps et al. 1996].

2.3.1.2 Other Petri-net-based systems

Numerous models have been proposed in the literature, which are intended to provide support for multimedia specification and development. Many are Petri-net-based models that focus mostly on the specification of multimedia data stream synchronization

requirements: OCPN (Object Composition Petri Net) [Little and Ghafoor 1990] and its derivations XOCPN (Extended Object Composition Petri Net) [Woo et al. 1994], DTPN (Dynamic Timed Petri Nets) [Prabhakaran and Raghavan 1993], TSPN (Time Stream Petri Net) [Diaz and Senac 1993], and $TSPN_{UI}$ (Time Stream Petri Net with User Interaction) [Cooper 1995]. These models, however, are not general enough for the specification of general hypermedia applications. In these systems, user input to a multimedia service is not described. In other words, these systems mainly provide rendering capabilities with no (or few) user interaction capabilities.

Some of the Petri-net-based models that satisfy the requirements of general hypermedia applications are HTSPN (Hierarchical Time Stream Petri Net) [Willrich et al. 1996], MHPN (Multimedia Hypermedia Petri Net) [Wang and Wu 1995], and MORENA (Multimedia ORganization Employing a Network Approach) [Botafogo and Mossé 1995].

MORENA is targeted at the description and execution of hypermedia applications that allow for flexibility and adaptability through message passing. MORENA provides support for multimedia data, such as video and audio, and structured authoring. Composite nodes have their own structure (nodes, transitions, and arcs) and encapsulate information (such as synchronization specification), defining a hierarchy that can be manipulated easily. In addition, MORENA provides fine-grained synchronization (a dynamic medium can send synchronization messages at specified moments during its execution). However, since MORENA does not fully follow Petri net theory, it may not

be easy to analyze the output model using the existing techniques developed in the Petri net field.

MHEG (Multimedia and Hypermedia information coding Expert Group) [ISO 1997] has established object-oriented standards for real-time interactive communication applications. The MHEG classes are one of the main efforts to standardize all the necessary object classes for multimedia and hypermedia applications. Therefore, MHPN links Petri net objects to MHEG objects in order to express the complex logic requirements of hypermedia applications such as flexible (asynchronous and concurrent) browsing and rendering capabilities, and to provide a formal graphical model for the structured authoring of hypermedia. A prototype object-oriented multimedia information system called the Petri net Object Information System (POIS) has been developed for providing an integrated facility for menu searching within a repository of filed multimedia documents, then extracting, selecting, and merging information from these documents.

The HTSPN model enables a unified specification of temporal and logical synchronization within hypermedia systems using the Dexter model [Halasz and Schwartz 1994] and Time Stream Petri Nets model (TSPN) as background. An interpreted version of the HTSPN model (I-HTSPN) provides the means to specify hypermedia documents on top of the hypermedia structure. In order to generate an information structure that can be exchanged and presented in an open hypermedia

system, a procedure translating an interpreted HTSPN specification into an MHEG representation has been developed [Willrich et al. 1996].

All these Petri-net-based models use a basic token (one without local values), thus the supplement of a structured token (one with local values) will help reduce the complexity of the models. In addition, a support of conditional statements for the threshold of the transition fire will be useful for the dynamic adaptation. Information presentation based on a simple paradigm (the content associated with every Petri net node is shown on a different window) may need to be extended for more flexible presentation of hypermedia documents.

Petri nets also have been widely used as a modeling tool in many other areas, including office information systems [Beslmuller 1988]. Information Control Net (ICN) [Ellis 1979] is an OIS (Office Information System) modeling tool derived from Petri nets. It has been created to model procedures, organizational structures, social norms, and exceptional conditions in offices. Within the context of this model, an office is defined to be a set of office objects together with a set of mappings among these objects. ICN allows analysis of properties such as throughput, deadlock, and streamlining [Ellis and Naffah 1987]. The use of Petri nets as a specification medium for man-machine interaction appears previously in the literature [Holt 1988; Van Biljon 1988].

2.3.2 Statechart-based systems

As an alternative to Petri-net-based models, the HMBS (Hypertext Model Based on Statecharts) model [De Oliveira et al. 2001; Turine and De Oliveira 1997] uses the

structure and execution semantics of statecharts [Harel 1987] to specify both the structural organization and the browsing semantics of a hyperdocument. The major features of HMBS are its ability to model hierarchy and synchronization of information, and its provision of mechanisms for specifying access structures, access controls, multiple tailored versions, and hierarchical views. Analysis of the underlying statechart model allows verification of document reachability and valid paths. HMBS is an appropriate model for applications that have a hierarchical structure, such as books, scientific papers, on-line tutorials and manuals, and instructional material.

XHMBS (the eXtended Hyperdocument Model Based on Statecharts) [Paulo et al. 1998] is an extension of HMBS with additional mechanisms for describing the time sequencing and information synchronization requirements typical of multimedia. Zheng and Pong [1992] also use statecharts for hypertext modeling. However, they are concerned with specifying the hypertext system user interface behavior. They use statecharts to model the behavior of buttons and frames supported by a hypertext system. Masiero et al. [1994] propose a method for analyzing office applications, which relies on a statecharts-based model to record the flow of documents within the system. Each type of document is associated with a statechart, and that enables the automatic update of the links maintained in a hypertext database when new documents are added to the database.

In general, statechart-based systems do not support adaptive, shared readings of a statecharts specification since there is no identity of each reader (like colored token

markings in Petri nets) in the specification. Therefore, they may not be appropriate for the authoring of adaptive hypertext documents, including cooperative documents.

In addition, Tompa [1989] uses a *hypergraph* formalism to model generic hypertext structures. This formalism enables formal identification of common structures (nodes, links, and labels) and direct reference to groups of nodes having common link semantics. This model supports a set-oriented browsing semantics, and incorporates an arbitrary number of layers of personalized and system structures.

2.4 Adaptive hypermedia systems

In most hypertexts, the contents of nodes and the positions of links are fixed. However, a number of hypertext researchers have demonstrated that nodes and links need to be dynamic, for instance to help authors both solve structural problems and reduce cognitive overhead by providing the most relevant adapted information for readers. In dynamic (or volatile) hypertexts, links can vary to suit the changing circumstances of reading, and nodes can change dynamically in response to the different situations in which the reader places them.

For example, an author using Storyspace [Joyce 1991] can create conditional links that are accessible to the reader only if certain criteria have been met, such as a specified node having been previously read or not read. Trellis can express prerequisite relationships between nodes by imposing sequence constraints on the browsing behavior of the reader. “A Life Set for Two” [Kendall 1996] introduces floating links that are

positioned dynamically in response to the reader's progress and variable nodes that change their texts according to factors such as their context within the current reading and global states.

With the help of active research into dynamic hypertexts (in the hypertext field) and user modeling (in the adaptive user-interface field), adaptive hypermedia (or hypertext) has been introduced as a fairly new research field. Brusilovsky [1996] provides the following definition of adaptive hypermedia systems:

> *"All hypertext and hypermedia systems which reflect some features of the user in the user model and apply this model to adapt various visible aspects of the system to the user."*

To put it in a different way, the system should satisfy three principles: it should be a hypermedia system; it should have a user model; and it should be able to adapt the hypermedia using this model (i.e., the same system can appear different to users with different models). Thus, adaptive hypermedia systems build a model of the individual user (called the user model) and apply it to adapt the content of a hypermedia page to the user's knowledge and goals, or to propose the most relevant links to follow. What can be adapted in adaptive hypermedia are the content of regular pages (content-level adaptation) and the links from regular pages, index pages, and maps (link-level adaptation). Adaptive presentation of information can be achieved through conditional text, stretchtext, fragment variants, page variants, and frame-based techniques; and adaptive navigation can be achieved through direct guidance, link sorting, link

annotation, link hiding, link removal, and link disabling [Brusilovsky 1996; De Bra et al. 1999a].

There are three steps in the adaptation process: gathering data about the user, processing the data to construct or update the user model, and applying the user model to provide the adaptation. Brusilovsky [1996] argues that there are some general problems related to automatic user modeling in adaptive systems. Firstly, automatic user modeling is not fully reliable. Systems that perform both user modeling and adaptation without user influence are twice as unreliable. Secondly, some components of the user model, such as background and preferences of the user, cannot be deduced at all and have to be provided directly by the user. Therefore, the only way for the system to get the required information about the user is to involve the user in the process of user modeling and to collaborate with the user in collecting the information. This is called collaborative or cooperative user modeling [Kay 1995].

Adaptive hypermedia systems are used now in several application areas where the hyperspace is reasonably large, and where individuals with different goals, types of knowledge, and backgrounds use a hypermedia application. The following is a list of these application areas: educational hypermedia systems, on-line information systems, on-line help systems, information retrieval hypermedia, institutional hypermedia, and systems for managing personalized views [Brusilovsky 1996]. AHAM [De Bra et al. 1999b] introduces a reference model for adaptive hypermedia applications, which includes most features supported by adaptive systems that exist today or that are being

developed. AHAM enhances the Dexter model [Halasz and Schwartz 1990; Halasz and Schwartz 1994] with features for adaptation based on a user model that persists beyond the duration of a session. The caT model uses a simple form of user modeling in support of adaptation.

2.5 Multimedia language for Web

SMIL (Synchronized Multimedia Integration Language) [SMIL 2001] is a meta-language that allows authors to write interactive multimedia presentations for the Web. It supports effective timing and synchronization, adaptation to users and systems, and modeling of a flexible and consistent presentation and user interface. SMIL 1.0 became a W3C Recommendation on 15 June 1998, while SMIL 2.0 (including SMIL Profile and SMIL Basic) is expected to become a W3C Recommendation soon. SMIL needs special players that understand the script language. Currently, the RealPlayer 8, GR*i*NS, and Quicktime 4.1 players support SMIL 1.0.

SMIL, defined with a XML DTD, is a declarative language using attribute/value pairs. In SMIL, the *par* element groups elements that are played in parallel. The *seq* element groups elements that are played sequentially—based on the textual order, children are played one after the other. In addition, *par* and *seq* can be nested. Using additional attribute values such as *dur*, *begin*, and *endsync*, SMIL can flexibly control presentation time (start and end time) of multiple multimedia items. Using the *switch* element, SMIL supports adaptation to characteristics of the user (disabilities and languages), the hardware environment (the bandwidth and available CPU time), and the

document purpose (the selection of appropriate content). In the *switch* element, at most, one of its children is played; since the first acceptable element is chosen, the specification ordering should be best first. In addition, the spatial layout provides flexible specification of documents' spatial locations. SMIL's basic layout is isomorphic and replaceable with CSS (Cascading Style Sheets). CSS is used for separating the content of the HTML documents from their presentation.

In general, since SMIL is based on XML syntax, it is difficult for common users to author SMIL specifications (e.g., the nested *seq* and *par* blocks) using a simple text-editor. Therefore, several SMIL authoring tools [Bulterman et al. 1998] have been developed. Since Petri nets provide an easy way of authoring sequential and parallel tasks, SMIL structure may be authored in Petri nets and converted to SMIL statements. In this case, the SMIL structure described in Petri nets can be analyzed using the analysis techniques of Petri nets. However, the conversion from Petri nets to SMIL statements may only be feasible in some limited fashion since some features in SMIL may not be represented in Petri nets, and vice versa.

Kim et al. [2000] have performed an analysis on SMIL structures. They try to analyze the following two document structures: the circular reference of the identifier (deadlock situation) and the conflict between regions simultaneously represented in the *par* element. However, the analysis discusses only a subset of possible analysis properties of SMIL and is based on rather ad-hoc analysis techniques compared to the

Petri net's analysis techniques, which are based on its own mathematical foundation. More systematic solutions for the analysis of SMIL structures may be necessary.

CHAPTER III

SYSTEM OVERVIEW AND DESIGN

3.1 System overview of the caT system

In this chapter, we introduce context-aware Trellis (caT), a context-aware hypertext model and associated tools. Firstly, the context-aware hypertext model is developed by incorporating both high-level Petri net features (such as structured tokens, flexible net description, and hierarchical nets) and user-modeling into the Trellis model. Fuzzy knowledge handling also is added by the integration of a fuzzy logic tool with the Petri net. The system uses the fuzzy logic tool for inferring the values of token variables using fuzzy rulebases.

Since Petri nets allow the specification of a document's browsing semantics—that is, "the dynamic properties of a reader's experience when browsing a document" [Stotts and Furuta 1989a]—caT generalizes the browsing semantics of a document and moves implementation of such mechanisms into the document specification rather than the system implementation. Therefore, the documents described in caT provide dynamic, interactive interfaces rather than static document pages. In caT, a structured authoring tool and an analysis tool are developed to support structured authoring and verification of application models, respectively. The author, using the structured authoring tool, first builds a Petri-net-based document structure and then associates net segments with the corresponding content segments.

As the reader browses a document using document browsers, the corresponding Petri net's states change. Dynamic documents are generated based on the net state and the document specification—in general, the document components associated with active net components are displayed. To allow multiple presentations of a particular document's specification, new browsers have been developed (or supported), such as an image browser and a Web browser. The Web-based flexible presentation is supported by a template file (Meta file), which specifies how active content elements are to be displayed and how links are to be embedded.

Consequently, by enhancing the Trellis model, caT provides good separation among a document's (1) structure, (2) data, and (3) browsing semantics. In the following subsections, the formal definition of the caT model is first introduced. That is followed by detailed descriptions of the system design. In the descriptions, when we refer to caT, we generally mean the extensions to Trellis; however, sometimes our use of caT also embraces Trellis. Thus, when it is necessary to distinguish between the two systems, we explicitly indicate caT and Trellis.

3.2 System design

3.2.1 Formal definition of the caT model

The foremost objective of this research is to develop a context-aware hypertext model, i.e., the caT model. The formal definition of the caT model is derived by adding its extended features to the existing formal definition for Trellis [Stotts and Furuta 1989a].

The formal definition of the Petri net used in the caT model is as follows (some notation used here is taken from Jensen [1992] and Stotts and Furuta [1989a]):

A caT Petri net structure is a tuple, HCPN = <S, ST, STM, IOM, I>, in which:

- S = <Σ, P, T, A, τ, C, G, E> is a set of pages; each page, s $\in$ S, is a non-hierarchical CPN = <Σ_s, P_s, T_s, A_s, τ_s, C_s, G_s, E_s>;

  Σ, P, T, A, τ, C, G, and E represent all corresponding data (or functions) in S, a set of pages; Σ_s, P_s, T_s, A_s, τ_s, C_s, G_s, and E_s represent data (or functions) in each page s. The detailed description of each item is shown below.

  - Σ_s is a finite set of token types called color sets;
  - $P_s = \{p_{s1}, p_{s2}, \ldots, p_{sn}\}$ is a finite set of places with $n \geq 0$;
  - $T_s = \{t_{s1}, t_{s2}, \ldots, t_{sn}\}$ is a finite set of transitions with $m \geq 0$ and $P_s \cap T_s = \varnothing$;
  - $A_s \subseteq (P_s \times T_s) \cup (T_s \times P_s)$ is the flow relation, a mapping representing arcs between places and transitions;
  - τ_s: $T_s \rightarrow \{0, 1, 2, \ldots, \infty\} \times \{0, 1, 2, \ldots, \infty\}$ is a function mapping each transition to a pair of values termed release time and maximum latency, respectively. For any transition $t \in T_s$, we write $\tau_s(t) = (\tau_s^r, \tau_s^m)$, and we require that $\tau_s^r \leq \tau_s^m$;
  - C_s: $P_s \rightarrow \Sigma_s$ is a color function mapping each place to a color set (type);

  - G_s: $T_s \rightarrow$ *Boolean Expression* is a guard function mapping a Boolean expression to each transition;
  - E_s: $(T_s \times P_s) \rightarrow$ *Arc Expression* is an arc function mapping an assignment expression to each output arc.

- $ST \subseteq T$ is a set of substitution transitions; T is a set of transitions in all pages;
- STM: $ST \rightarrow S$ is a substitution transition/page mapping function; no page is a subpage of itself;
- IOM: $ST \subseteq (P_{source} \times P_{target})$ is a input/output mapping function; P_{source} is a set of places (especially input or output places of ST transition) in the source page, and P_{target} is a set of places in the target page (STM(st));
- $I \in S$ is a start page.

The caT model has additional (or enhanced) properties in addition to the ones it shares with the Trellis model. In both caT and Trellis, each token can have a color value. caT extends this to provide optional local token variables as well. Thus, each place can have different types of tokens (i.e., tokens with different local variables), declared by the color function, $C_s(p_s)$. The guard function, $G_s(t_s)$, is used for mapping a Boolean expression to each transition. The Boolean expression specifies an additional constraint (threshold) that must be fulfilled before the transition is enabled. The arc expression function, $E_s(t_s \times p_s)$, is used for mapping an assignment expression to each output arc, which changes current token values when the transition is fired. STM(st) and IOM(st)

are functions for building hierarchical nets. STM(st) is a page (or net) assignment function for mapping a transition to a subpage (or subnet). IOM(st) is an input/output assignment function that maps the input/output places of the current substitution transition in the current page to the places in the target page.

A hypertext in the caT structure is a tuple, <HCPN, M_0, D, W, B, P_l, P_d>, in which:

- HCPN is a hierarchical colored timed Petri net <S, ST, STM, IOM, I>;
- $M_0 : P \rightarrow$ "token instances of Σ" is an initial marking (or initial state) for HCPN;
- D is a set of document contents;
- W is a set of windows;
- B is a set of buttons (or links);
- P_l is a logical projection for the document, $P_l = <D_l, W_l, B_l>$, which maps components of a Petri net to the human-consumable portions of a hypertext;
- P_d is a display projection for the document, a collection of mappings that associate the logical buttons and windows of a hypertext with physical screen representations and locations.

3.2.2 Context-aware hypertext model

We begin with the χTrellis implementation developed by Stotts et al., and add several features to it to provide the context-aware hypertext model—i.e., the caT model. This section describes the details of the implementation of the caT model.

3.2.2.1 Structured tokens

Each colored token can carry its own local variable/value pairs; the tokens with the local values are called structured tokens. Structured tokens allow net status information to be associated with the token rather than requiring its encoding into the state of the Petri net. This feature helps to reduce specification complexity.

For example, Figure 3 shows two colored tokens, *black* and *pink*, with three local variable/value pairs: *class*, *user*, and *enterTime* (the behavior of the net will be discussed in the next sections). The local token values can represent dynamically-changing characteristics of the current token, and the token values can be updated while moving around the net. Petri nets supporting structured tokens provide different behavior to different valued tokens.

For hypertext applications, each colored token may represent a different reader who is in a dynamically-changing environment. With this token information, caT can provide different behavior to the user under different contexts (e.g., different access times, such as morning, evening, spring, and winter). Users can share the same Petri net, but have context-aware, customized views from the net. On the other hand, in a CSCW application, the colored tokens might represent different roles (e.g., access control).

Providing an individual identity to tokens requires a mechanism that determines what values should be used if tokens are combined or replicated during transition firing. In caT, these transformations will be specified through predicates associated with the arcs between the transition and the place. The token expressions are used for this purpose,

and different color variables in the expression are instantiated to different color tokens. Table 1 describes the syntax and gives examples of the token expression.

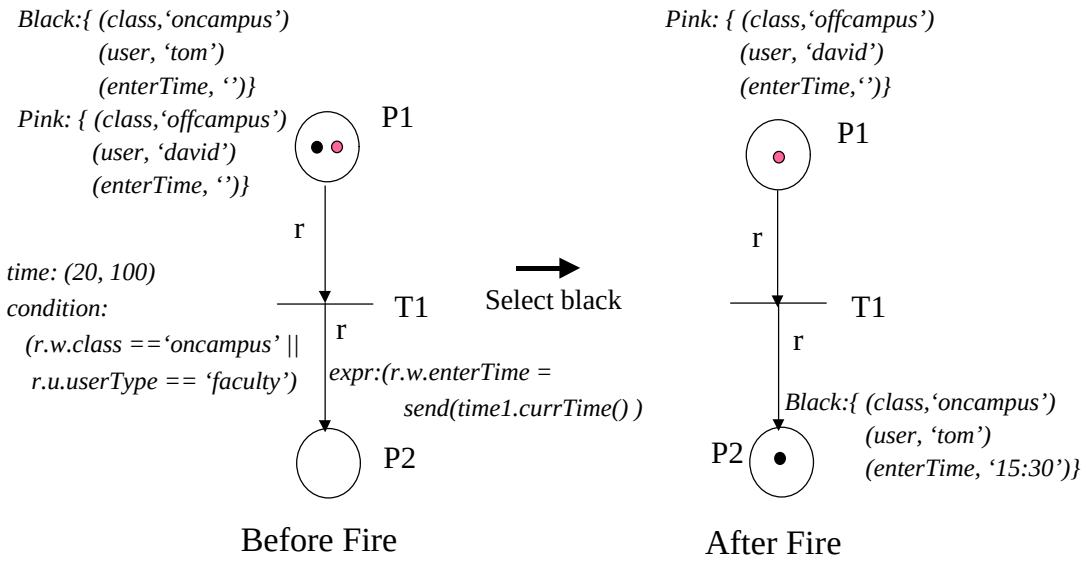


Figure 3: caT Petri nets: colored tokens with local variable/value pairs

For instance, in Figure 3, the notation on the arcs from *P1* to *T1* and from *T1* to *P2* defines a variable, *r*. The color of the token from *P1* that is consumed when *T1* fires must match the color of the token deposited into *P2* since the notation on the arcs defines the same variable, *r*. Additionally, a constant value can be used on the arcs for specifying the constant (same) color, such as *black* and *green1*. This definition of

variables was provided in ÷Trellis. caT, however, must extend the notation, as will be described next.

Table 1: Syntax of the token expression

<VAR_EXP>*	::= <LIST_OF_TERM>
<LIST_OF_TERM>	::= <TERM> | <TERM> '+' <LIST_OF_TERM>
<TERM>	::= *integer***<COLOR>
<COLOR>	::= <VARIABLE> | <COLOR_CONSTANT>
<VARIABLE>	::= *string*[*:integer*]
<COLOR_CONSTANT>	::= *black* | *blue1* | *blue2* | *blue3* | *magenta1* | *magenta2*| *magenta3*| *magenta4* | *green1* | *green2* | *green3* | *green4* | *pink1* | *pink2* | *pink3* | *pink4* | *maroon1* | *maroon2* | *maroon3* | *maroon4*| *red1* | *red2* | *red3* | *red4* | *orange1* | *orange2* | *orange3* | *orange4* | *purple1* | *purple2* | *purple3* | *purple4*
*: <BRACKETS> is a non-terminal symbol; 'quotes' is a terminal symbol. **: *Italic* value is a terminal symbol (or a constant value). Examples: *2black* // specifies 2 tokens of *black* color *2x + 3y* // specifies 2 tokens of one color (*x*) and 3 tokens of a second color (*y*).	

In Figure 4, the notation on the input/output arcs of the *T1* transition defines a variable, *r*. However, the color variables have an additional identifier after the variables, such as *r:1* and *r:2*. The additional identifier is necessary to merge control flows of the same colored tokens that may have different token values. For example, *black* tokens in *P1* and *P2* have different token values: *black* in *P1* has three local token values, while

black in *P2* has no local token values. The *black* token extracted from *P1* when *T1* is fired is deposited into *P3* since the notation on the arcs defines the same variable, *r:1*. Similarly the *black* token extracted from *P2* is deposited back into *P2*.

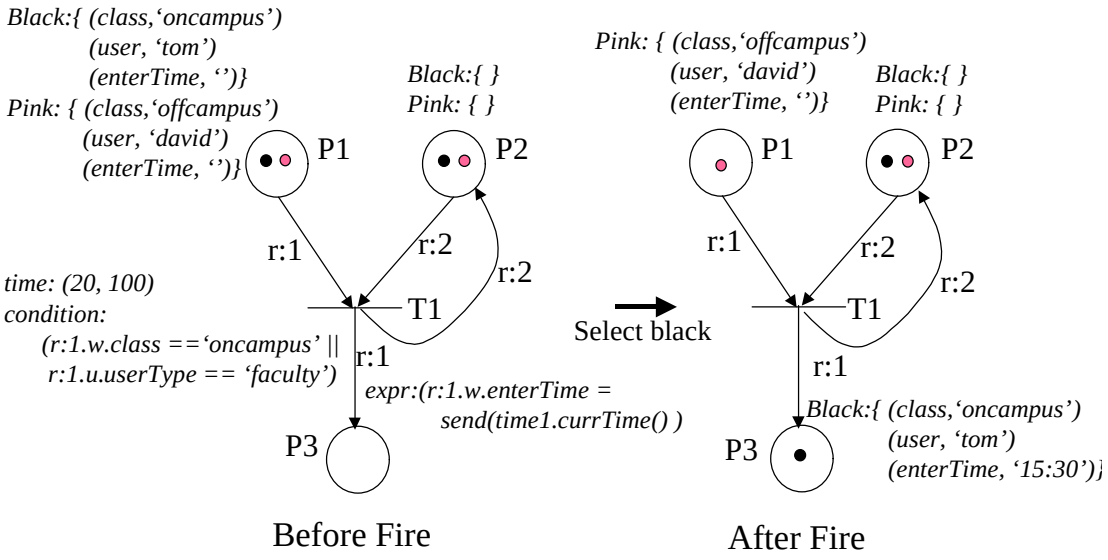


Figure 4: caT Petri nets: flow control specification of colored tokens

3.2.2.2 User modeling

Each caT colored token can have a link to a user-modeling profile that contains information associated with the user's preferences and with the external world. For example, profile data might include a user's ID or information about a user's educational background or job category. caT can use this user modeling information, in addition to

local token values, to customize its behavior to different users. When a user does not have a user profile in the system, predefined default values stored in the default user profile are used.

Additionally, a process external to the caT system might maintain a record of the user's current location—this location information is communicated to caT by updating the associated user-model profile value. Consequently, the user model profile is visible globally, and hooks are provided to allow modification of its values by external processes; compare this to the local token value/variable pairs, which are maintained separately for each individual token.

For context-aware applications, user modeling may help supplement the limits of current sensor devices. In the real world, some sensor devices are not available to common users, and sometimes it is impossible to get some environmental data from the sensor devices, such as whether or not the user is busy. Moreover, we can use the user-modeling feature for simulating a pretend world; the user may want to see context-aware information by simulating his context information in the user profile.

3.2.2.3 Link adaptation

For supporting dynamic link adaptation, conditional statements (or conditional predicates) attached to transitions determine the threshold values for transition firing. The conditional statements are evaluated with the current user model and token values—Table 2 describes the syntax and examples of the conditional statement. Therefore, caT can provide different user behavior with different token and user model values. caT's

conditional predicate is similar to a Storyspace guard field [Joyce 1991]. However, in addition, caT can provide different behavior to a user under different contexts (e.g., different access times, such as morning and evening, spring and winter, etc.).

Table 2: Syntax of the conditional statement

<CONDITION>		::= <OR_CONDITION> | <AND_CONDITION>
<OR_CONDITION>		::= <OR_CONDITION> '||' <EXPRESSION> | <EXPRESSION>
<AND_CONDITION>		::= <AND_CONDITION> '&&' <EXPRESSION> | <EXPRESSION>
<EXPRESSION>		::= '(' <TOKEN> <OPERATOR> <TOKEN> ')'
String data type	<OPERATOR>	::= '==' | '!='
	<TOKEN>	::= '*string*' | <VARIABLE>
Numerical data type	<OPERATOR>	::= '==' | '!=' | '<' | '>' | '<=' | '>='
	<TERM>	::= <TERM> '+' <PRIMARY> | <TERM> '–' <PRIMARY> | <PRIMARY>
	<PRIMARY>	::= <PRIMARY> '/' <TOKEN> | <PRIMARY> '*' <TOKEN> | <TOKEN>
	<TOKEN>	::= *integer* | *float* | <VARIABLE> | fis() | send() | '(' <TERM> ')'
<VARIABLE>		::= colorVariable.w.tokenVariable | colorVariable.u.userProperty

Examples:

For string date type operation:

r.w.class == 'oncampus' ; r.w.class != 'oncampus'

For numerical data type operation:

r.w.accessRight >= (0.8 * 2)

(r.w.accessRight >= fis(./libcat/accessright.fis,10,100)) || (r.w.network != '128.194.147')

In the conditional statements, local variables of tokens and user profile property values can be accessed. The following syntax is used for accessing a local variable of the current colored token: *colorVariable.w.tokenVariable*. For example, in *"r.w.class"*, "*r*" is a color variable that is instantiated to the current color token, and "*w*" denotes that the following variable, *class*, is a local variable of the current token. For accessing a user profile property value, the following syntax is used: *colorVariable.u.userProperty*. For example, in *"r.u.userType"*, "*r*" is a color variable, and "*u*" denotes that the following variable *userType* is a user profile property. A specific user profile, implemented as a plain ASCII file, is identified by the value of the current token's *user* variable. If there is no *user* variable in the current token, the default user profile is accessed.

For example, in Figure 3, the *T1* transition has a following condition statement: *r.w.class == 'oncampus' || r.u.userType == 'faculty'*. Only the *black* token satisfies the condition; the *pink* token, which has the *userType* value as *student* in its user profile, does not satisfy the condition. The *black* token can therefore only be enabled and fired for *T1*. When the *T1* transition is fired, a colored token (i.e., the *black* token) in *P1* carries its own local variable/value pairs to the output place (i.e., *P2*) with a new *enterTime* value (the output arc expression will be discussed later in this section). The condition statement in Figure 4 has the same statement as the one in Figure 3 except for the additional identifier specification after the color variable, which allows specifying one of the same colored tokens in input places.

In the condition statement, some functions can be used for getting external values. The "*send()*" function is used for calling system-support functions; "*fis()"* is used for invoking an external fuzzy logic engine. The detailed description of these functions will be discussed in the next section. The conditional predicates can be used with caT's timing values to provide more dynamic control of the reader's traversal. Timing values can be thought of as defining ranges for the availability of an event (refer to the formal definition of the Trellis model in the Trellis section of Chapter II). For example, in Figure 3, the *T1* transition has a time value, (20,100). *T1* will be enabled after 20 time units (i.e., seconds) delay, and fired automatically by the system unless the user fires the transition within 100 time units.

Therefore, when both the conditional and timing predicates are defined, they must be satisfied at the same time before the transition can be active. This allows the specification of transition conditions such as: "show a link to a reader who accesses this document during daytime and has looked at the document for at least 3 minutes". In this example, the "daytime" condition would be evaluated through the conditional predicate, while the "at least 3 minutes" requirement would be specified with caT's timing values.

Additionally, assignment statements attached to output arcs are used for changing current token values while the token moves around the net. For example, in Figure 3, an arc between the *T1* transition and the *P2* place has the following arc assignment statement: *r.w.enterTime = send(time1.currentTime())*. The *send(time1.currentTime())* function returns the current time, and the return time value is assigned to the *enterTime*

variable of the current token. Thus, when the *T1* transition is fired, the *black* token in *P1* carries its own local variable/value pairs to the *P2* place with a new *enterTime* value: *{(class,'oncampus'), (user,'tom'), (enterTime,'13:30')}*. Table 3 describes the syntax and examples of the assignment statement. In the assignment statement, the *send()* and *fis()* functions can be used to get external values.

Table 3: Syntax of the assignment statement

<table>
<tr><td colspan="2"><ASSIGNMENT> ::= <VARIABLE1> '=' <EXPRESSION>
<VARIABLE1> ::= colorVariable.w.tokenVariable
<EXPRESSION> ::= <TERM> | 'string'</td></tr>
<tr><td>Nume-rical data type</td><td><TERM> ::= <TERM> '+' <PRIMARY> |
<TERM> '–' <PRIMARY> | <PRIMARY>
<PRIMARY> ::= <PRIMARY> '/' <TOKEN> |
<PRIMARY> '*' <TOKEN> | <TOKEN>
<TOKEN> ::= integer | float | <VARIABLE2> | fis() | send() |
'(' <TERM> ')'
<VARIABLE2> ::= colorVariable.w.tokenVariable |
colorVariable.u.userProperty</td></tr>
<tr><td colspan="2">Examples:
For string date type operation:
r.w.class = 'oncampus' // need single quotation marks for the string value
For numerical data type operation:
r.w.accessRight = (0.8 * 2)
r.w.accessRight = fis(./libcat/accessright.fis,10,100)</td></tr>
</table>

As a result, the extended net description functionality (i.e., the conditional and arc assignment statements) provides adaptive links and the user can browse adaptive documents based on the adaptive net behavior.

3.2.2.4 Hierarchical nets

One problem with the Petri net model is that users have difficulty specifying large, complex nets. Thus, structured authoring support (i.e., hierarchical nets) is essential for developing large-sized Petri net applications since it reduces graphical complexity problems that are common to graph-based modeling tools. Structured authoring support also allows easy system tracing for simulation and debugging purposes since large nets can be modularized.

In Trellis, a Petri net engine can hold only one net. Thus, hierarchical subnets are handled by the browser—a browsing tool invokes the subnet using a content attribute of a place, and causes invocation of a new, independent instance of the Petri net engine to handle its execution in a loosely-coupled fashion. Consequently, there are no closely-coupled interactions between subnets, such as control-flow coupling or data value (i.e., token value) passing.

caT incorporates a hierarchical Petri net (which is one of the features of the high-level Petri net) into the Trellis model—we mainly follow the hierarchical Petri net model introduced in Jensen [1992]. caT shifts the responsibility for subnets from the browser to the Petri net engine. Essentially, this shift combines the nets into a single unit. Shifting hierarchical net handling from the content-level (i.e., a browsing tool) to the net

structure-level (i.e., an information server) will be more appropriate for providing flexible interactions among subnets.

Therefore, in caT, the Petri net engine has been extended to handle multiple subnets that are hierarchically linked. Additionally, a multiple-window-based authoring tool has been built to allow for the building and browsing of hierarchical subnets. The subnets can be used like functions (or macros) in high-level programming languages. This feature increases their reusability.

In caT's representation, a transition in a higher-level net may be mapped to a separate subnet (see Figure 5) The transition that is expanded to the subnet is called the *substitution transition*. Additionally, places in the higher-level net are mapped to places in the subnet—generally, the input to the subnet corresponds to places that lead into the substitution transition, and the output from the subnet corresponds to places that come from the substitution transition. When a token arrives in a mapped place, it appears simultaneously in both nets. Consequently, tokens in the higher-level net are first conveyed to the subnet, and then back from the subnet to the higher-level net.

The advantage of expanding a transition rather than a place is that this permits specification of multiple input and output places—if we had chosen to expand places, the natural mapping would allow only a single input and a single output place. Alternatively, a place could be expanded to the lower-level net where specification of multiple input/output (or start/finish) transitions is allowed—e.g., in the MORENA model [Botafogo and Mossé 1995], a place can map to the lower-level net that has

multiple input transitions. In this case, we need a specification for selecting one of the input transitions in the lower-level net from the place in the higher-level net. We also need a specification for connecting the output transitions in the lower-level net to the places in the higher-level net. This multiple input/output transitions scheme is more complicated and less natural for specification than the scheme (i.e., multiple input/output places) used in caT.

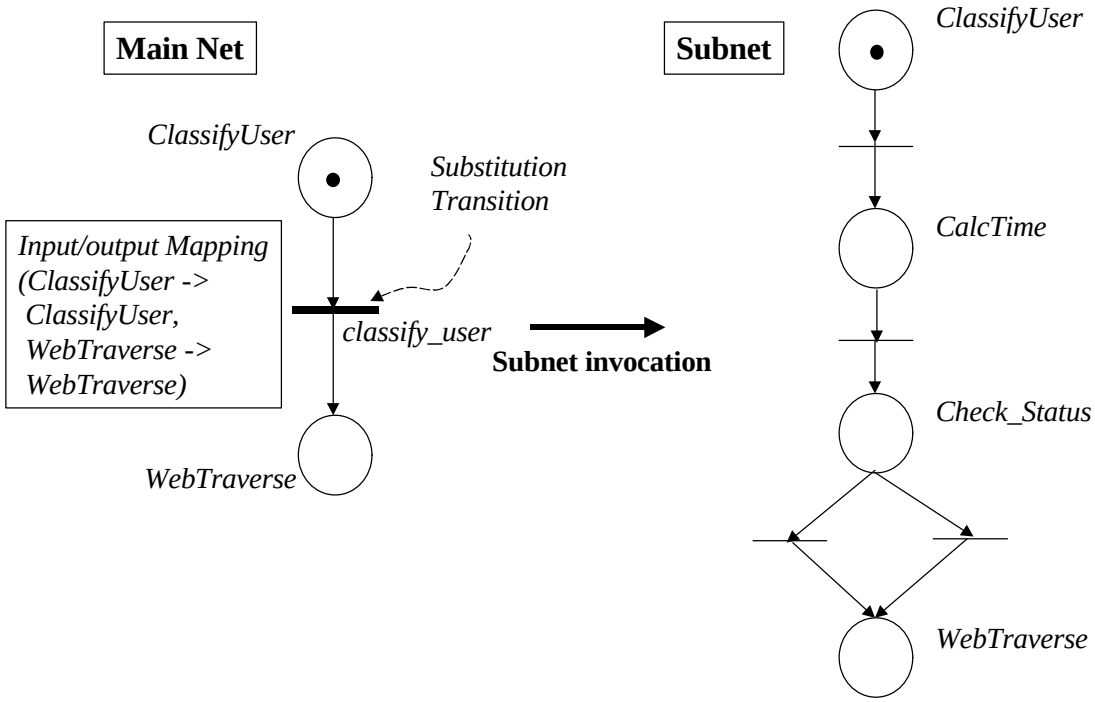


Figure 5: Subnet specification

In Figure 5, the subnet (the net on the right side) is associated with the *classify_user* transition in the main net. The main net's *ClassifyUser* place is mapped to the subnet's *ClassifyUser*, and the main net's *WebTraverse* place is mapped to the subnet's *WebTraverse* (the names are not required to match, but this is permitted, as in this example). Note that the token in the main net's *ClassifyUser* place has been replicated in the subnet's *ClassifyUser* place.

For a more detailed explanation in this chapter, we will consider a simple example of an online art gallery, which provides a complete set of resources to its "on-campus" patrons and a limited one to its "off-campus" patrons. In this example, the gallery's owners have decided that full access (on-campus-access) will always be allowed for patrons on the campus subnet. It also will be allowed during daytime hours for people located physically near the campus or for faculty members regardless of their location. The specification of the example consists of seven subnets. Figure 6 shows the segment of the specification that first classifies the current user as being "on-campus" or "off-campus" before invoking the *Gallery_tour:#2* subnet shown in Figure 7. The figures are actual snapshots of the subnets displayed in caT's authoring tool, and irrelevant subnets are iconized.

Beginning in the subnet *Start_tour:#1* in the left side window, user tokens will be placed initially in the *ReaderPool* place. Each colored token has the following local token variable/value pairs: *user, accessRight, class, network,* and *currentTime*. The *user* and *network* values will be set initially, and the other values are inferred or calculated by

the system. A user profile for each user has the following values: *disToCampus* (distance from a current location to campus) and *userType* (user occupation).

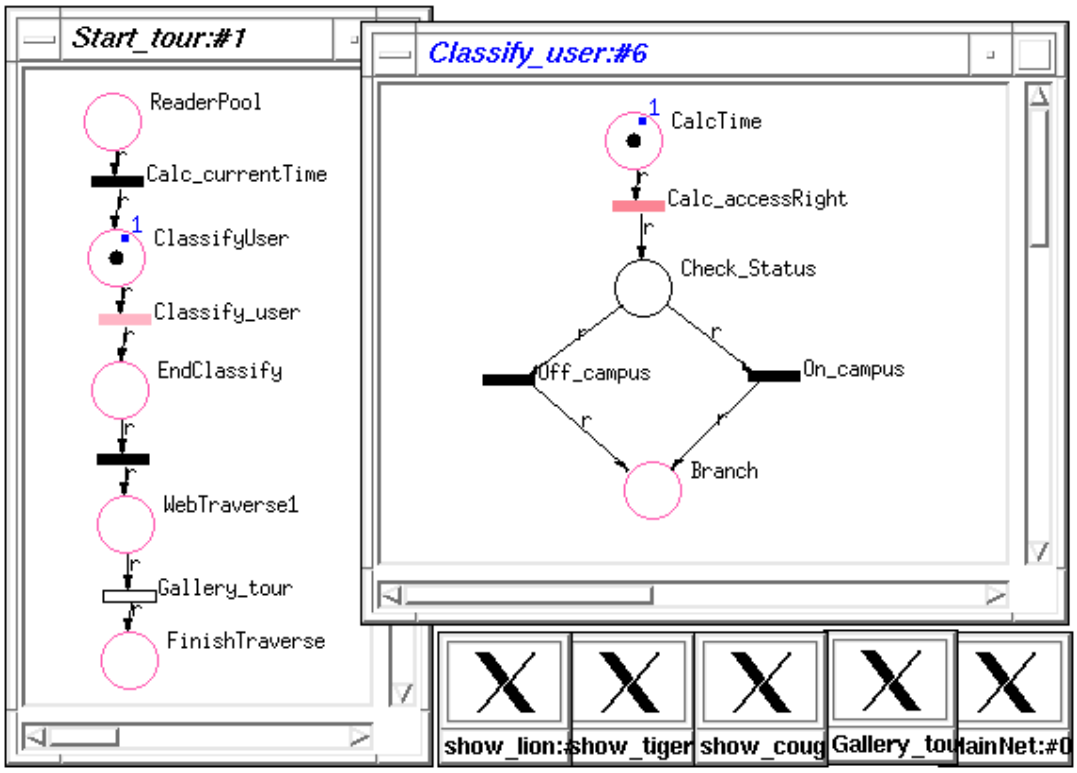


Figure 6: Classify user in gallery tour

The *Calc_currentTime* transition has a timing value of (0,0). (0,0) means no delay and no timeout; therefore, the *Calc_currentTime* transition will be executed immediately by the system when it is enabled. When the *Calc_currentTime* transition is fired, the *currentTime* value gets a new current time value by executing an expression that is attached to the output arc of the *Calc_currentTime* transition: *r.w.currentTime* =

send(time1.currTime()). "r.w.currentTime" is used for accessing the local token variable *currentTime* of the current colored token. The "*send(time1.currTime())*" function is used for calling a system-support time function that returns the current time of day. When the *Calc_currentTime* transition is fired, a colored token in *ReaderPool* carries its own local variable/value pairs to the output place (i.e., *ClassifyUser*) with a new *currentTime* value.

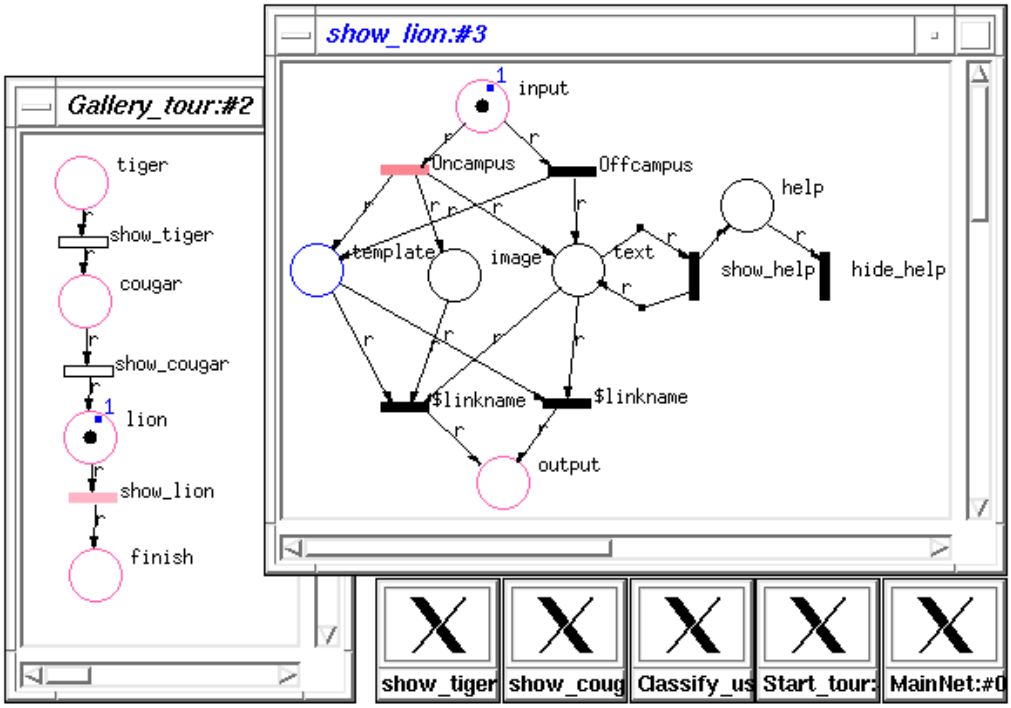


Figure 7: Show lion in gallery tour

Next, the subnet *Classify_user:#6* associated with the *Classify_user* transition in the main net is invoked. The subnet's *CalcTime* place is mapped to *ClassifyUser*, and its *Branch* place is mapped to *EndClassify*—note that the token in *ClassifyUser* has been replicated in *CalcTime*. In caT's authoring tool, the large dot in the center of the place indicates that it has been marked. The smaller, colored dots around the perimeter of the place show which color token(s) is (are) included, and the adjacent number outside of the perimeter shows how many tokens of that color are present. In the authoring tool, the subnet name located on each window's title bar copies the source transition's name and has an additional unique number after the name; an inactive transition is represented as a black rectangle; fireable transitions are flagged by coloring them red. A subnet transition is represented as a rectangle with empty fill. The rectangle is filled with pink color when there are active transitions inside of the corresponding subnet.

Using the current user's context information, the *Classify_user:#6* subnet mainly classifies the current user as being "on-campus" or "off-campus". Here, the following statement on the output arc of *Calc_accessRight* determines whether the user fits either of the "high access" conditions: *r.w.accessRight = fis(./libcat/accessright.fis, r.w.currentTime, r.u.disToCampus)*. The *"fis()"* function invokes a fuzzy inference engine to infer an *accessRight* value from the specified rulebase with input values of the current user's location (*disToCampus*) and the previously-acquired time-of-day (*currentTime*)—the *accessright.fis* rulebase will be discussed in the later section. The current user's location information is stored in the user profile. We assume that the location information is maintained by mechanisms outside of the caT system (e.g.,

environmental sensor devices and related computer tools). The time-of-day value is carried in a local variable of the token.

When a token arrives at the *Check_Status* place, either the *Off_campus* or the *On_campus* transition is enabled to assign a classification to the user. Each condition statement attached to its transition is evaluated with the current token and user profile values. The condition predicate in *Off_campus* is *"(r.w.accessRight < 0.8 && r.w.network != '128.194.147' && r.u.userType != 'faculty')"*, while the condition in *On_campus* is *"(r.w.accessRight >= 0.8 || r.w.network == '128.194.147' || r.u.userType == 'faculty')"*. When the current token's inferred *accessRight* is larger than or equal to 0.8 (this figure is rather high), its *network* is '128.194.147' (if you want to specify only the first two octets of the IP address, '128.194' can be used instead), or its *userType* is *'faculty'*, the *On_campus* transition is enabled; otherwise, the *Off_campus* transition is enabled. When the transition is fired, the assignment statement (*r.w.class = 'oncampus'* or *r.w.class = 'offcampus'*) attached to the output arc of the transition assigns the appropriate classification value to a variable carried by the token.

The system will then classify the current user as being "on-campus" or "off-campus". After the user is classified, a token is in both the *Branch* and *EndClassify* places, so the token will move to the *WebTravers1* place in *Start_tour:#1*. The system is now ready to invoke the *Gallery_tour:#2* subnet shown in Figure 7. Up to this point, since all of the transitions have the time values (0,0), they are executed by the system without interaction with the user.

The net in the left window of Figure 7, *Gallery_tour:#2*, specifies a path for traversing three image pages in sequential order. The subnet on the right of the figure displays a graphical version of an image to the “on-campus” user and a text-only version to the “off-campus” user. Either version includes help information (irrelevant subnets are iconized). In this case, as the window name suggests, the image is that of the lion, and the subnet is associated with the *show_lion* transition in the main net.

In the figure, the subnet’s *input* place is mapped to *lion*, while its *output* place is mapped to *finish*. Note that the token in *lion* has been replicated in *input*. There are three subnet mappings defined in Figure 7. In caT, the parent net can pass presentation-related values such as the contents (i.e., file names) of the *image* or *text* places in *show_lion:#3* to child subnets. Using the same subnet, all three subnet transitions in *Gallery_tour:#2* expand to the same structure, but each subnet transition passes different display information. In the figure, when the *Oncampus* transition in *show_lion:#3* is enabled and invoked, the *image*, *text*, and *template* places get tokens. Then, the Web browser will display the contents of the *image* and *text* places for an “on-campus” user (the use of the *template* node for Web presentation will be discussed in a later section). On the other hand, when the *Offcampus* transition is enabled and invoked, the *text* and *template* places get tokens. The browser will then display the contents of the *text* places for an “off-campus” user. As a result, the online art gallery provides a complete set of resources to its “on-campus” patrons and a limited one to its “off-campus” patrons.

3.2.3 Fuzzy knowledge handling

In the real world, we need to handle uncertain knowledge everyday. This fact applies also to context-aware applications, where we need to handle uncertain user contexts. Fuzzy tokens have been proposed for Petri nets [Cardoso et al. 1996], but we decided that introducing fuzzy logic into the current model would result in an unnecessarily complex, fuzzy Petri-net-based model. Consequently, caT keeps the Trellis Petri net model, and invokes an external fuzzy logic engine when necessary. caT uses the fuzzy logic engine in evaluating conditional statements used for link adaptation as well as for updating values in local variables on the transition firing.

Matlab Fuzzy Logic Toolbox [The Mathworks, Inc. 1999] is used as caT's fuzzy logic engine. The rulebase and membership functions are authored by using Matlab's GUI-based toolbox; i.e., FIS (Fuzzy Inference System) Editor, Rule Editor, Membership Function Editor, Rule Viewer, and Surface Viewer.

In caT, the following function for invoking an external fuzzy logic engine is provided: *fis(rulebase name, argument1, …, argumentN).* The first argument is a rulebase name, and the following arguments are either local token values or user profile values. In order to provide an example, we return to our earlier art gallery. In the art gallery example, the user's access right (*accessRight*) value is inferred by invoking the following *"fis()"* function: *fis(./libcat/accessright.fis,r.w.currentTime,r.u.disToCampus).* The first argument is a rulebase name for inferring the access right value (*accessright.fis*), and the following arguments are additional argument values: the user's

access time (*currentTime*) and the current location (*disToCampus*). The *accessright.fis* rulebase is as follows:

```
Rule1: If (time is day) and (distance is close)
       then (accessRight is high) (1)
Rule2: If (time is day) and (distance is middle)
       then (accessRight is middle) (1)
Rule3: If (time is day) and (distance is far)
       then (accessRight is low) (1)
Rule4: If (time is not day)
       then (accessRight is high) (1)
```

Generally, the rulebase infers an access right (*accessRight*) value from the current user's access time and distance. When the current time is *day*, users who are *close* in distance get high access rights. However, when the current time is not *day*, all users get high access rights, regardless of their distance. Fuzzy terms such as *day, close, middle,* and *far* are defined by membership functions. A membership function is a curve that defines how each point in the input space is mapped to a membership value (or degree of membership) between 0 and 1.

For example, a membership function for a fuzzy term *day* is shown in Figure 8. In the figure, when *time* (on the horizontal-axis) is between 9:00AM and 5:00PM (17:00), the membership value of *day* (on the vertical-axis) is 1. On the other hand, when *time* is

between 8:00PM and 4:00AM, the membership value is 0. Even further, when *time* is in a fuzzy range, the membership value is between 0 and 1. For example, when *time* is 6:00AM, the membership value is around 0.5. Since the membership function defines the meaning of a fuzzy term, the user can specify his (or her) meaning for the fuzzy term (*day*) by adjusting the membership function. In a similar fashion, the other fuzzy terms such as *close, middle, far, high, middle,* and *low* are defined by membership functions.

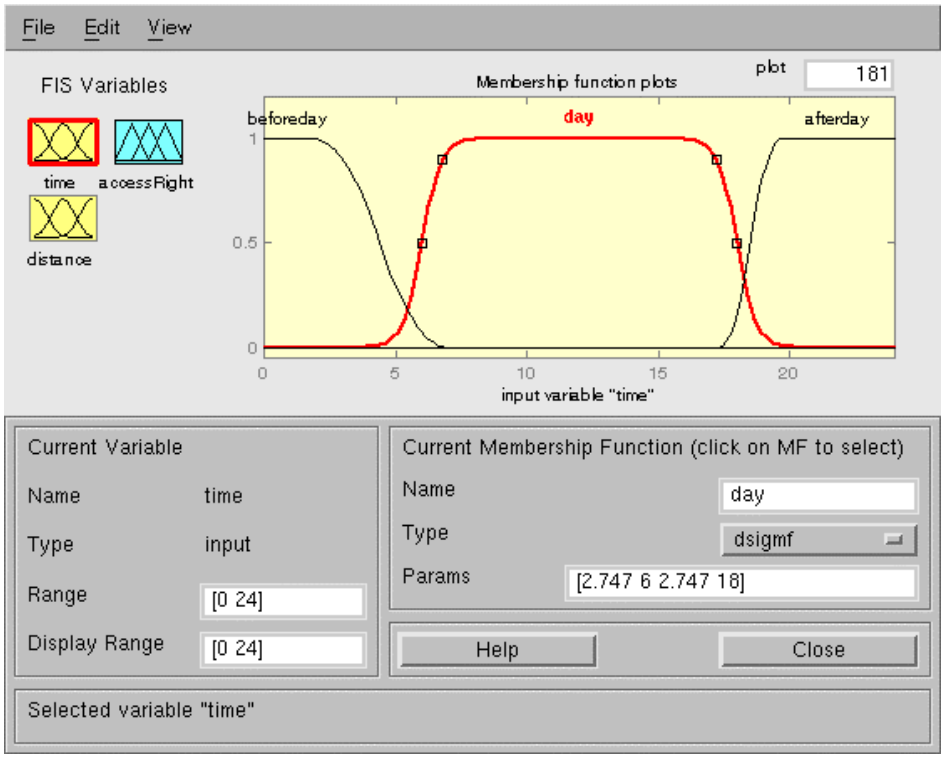


Figure 8: A membership function for day

Figure 9 shows an output window of Matlab's Rule Viewer for the *accessright.fis* rulebase. The Rule Viewer shows a general overview of the current rulebase with its membership functions, and it shows a result value when the user simulates input values. According to the figure, the access right value is 0.903 when *time* is 2:00AM and *distance* is 80 miles. The detailed algorithms for inferring a result value from the fuzzy rulebase are described in [The Mathworks, Inc. 1999; Yen and Langari 1999].

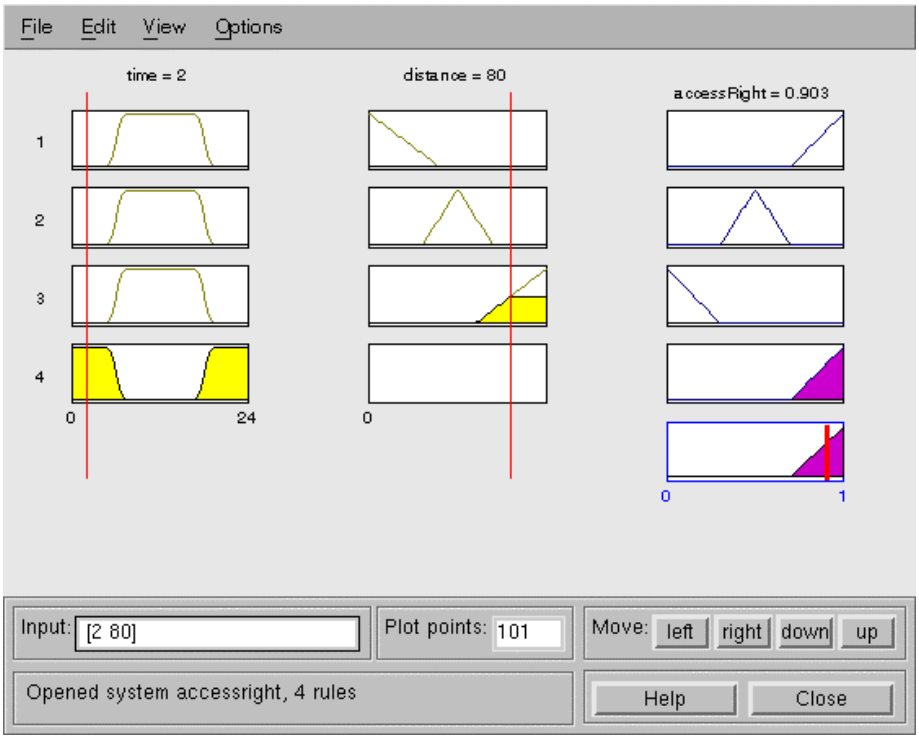


Figure 9: Matlab's Rule Viewer for the sample rulebase

3.2.4 Flexible information presentation

caT (and Trellis) supports good separation between document specification and presentation. This separation allows multiple presentations of a particular document's specification. A reader can choose the appropriate presentation for a document by selecting a browser, or he (or she) can use multiple browsers simultaneously. Moreover, when read by multiple readers simultaneously, the behavior of documents can be specified to range between shared and separate. Consequently, distributed readers can be specified to be unaware of each other, sharing the document's specification, but each with an independent view of the document. Alternately, one reader's action may be specified to affect another reader's view of documents. This specification allows for sharing in cooperative environments.

In Trellis, a browser called `xtb` displays the content associated with each active place in a separate window. In caT, we developed additional browsers for multiple presentations of a particular document's specification: an image browser displays only the image content associated with each active place; a text browser displays the text content; and the authoring tool, which is another client browser, shows the multiple subnets' status changes when it is in a simulation mode (a mode that is especially useful to the author).

The reader also can browse the caT model using Web browsers such as Netscape Navigator. The Web-based browsing feature of caT allows the user to define sophisticated hypertext applications using the caT model and to browse the resulting

information using the familiar user interface of WWW (World-Wide Web) browsers. For example, Web pages generated by the caT model easily may inherit timing features (e.g., time-based access control) from caT. The support for browsing with Web browsers provides a connection between the caT hypertext model and the WWW to derive synergy from the two different models. caT (and Trellis) provides flexible hypertext modeling features, but the WWW has been more widely accepted due to the simplicity and availability of its browsers. Web-based flexible information presentation will increase the usability of caT.

The design of the earlier Trellis browsers assumed that the display environment would be a workstation-based windowing environment. Consequently, the browsers have made free use of multiple windows. For example, χTrellis's `xtb` displays the content associated with each active place in a separate window. When the content is textual, the window also allows for the selection of the associated links. However, when the content requires invocation of an external rendering program (e.g., when images are displayed by invoking the `xv` program), the available links appear in a separate control window.

In the World-Wide Web, information presentation is browser-centered rather than screen-centered, and scroll-based rather than card-based. While it is possible to support multiple-window displays in the Web context (see, for example, Chung [1997] and Shipman et al. [2000]), the commonplace information presentation is in a single window. After evaluating the different possible implementations of a Web-based presentation, we

decided that the most natural representation would be as a composite node—with the active content elements and links displayed as a single Web page. The composite node has been used in other hypertext models or systems: for example, the Dexter model [Halasz and Schwartz 1990] defines composite components that are constructed out of other components; the document in the Augment system [Englebart 1984] is a tree-structured composition of atomic components called statements; and the filebox concept in NoteCards [Halasz 1988] provides hierarchical organizational structures using a composition mechanism.

In caT, a template file (Meta file) specifies how active content elements are to be displayed and how links are to be embedded. The template file is simply another Trellis content type associated with a place. It takes control when its associated place is marked; consequently, different template files can be active at different times during a browsing session. The resulting Web page is a virtual composite node constructed from several atomic nodes. Only active caT content elements are used in generating the Web presentation of the composite node. Thus, based on the current net state, the Web display changes dynamically.

The template file is, essentially, an HTML page with place-holders indicating where information is to be embedded when the corresponding places are marked. The Template file has the following format (the default value for each option attribute is underlined):

<*Template* [[*multiple_frame* = "yes_3frame" | "yes_2frame" | "no"]
[*linkbullet_show* = "yes" | "no"]]>
<*Transition name* = "[placeName^]transitionName"
[[*activelink_name* = "activeTransitionName"]
[*inactivelink_name* = "inactiveTransitionName"]]>
<*Place name* = "placeName"
[[*link_display* = "all" | "none" | "show: 'transName1',…,'transName*N*'"]
[*link_location* = "below" | "above"] [*link_sort* = "yes" | "no"]
[*link_duplicate* = "yes" | "no"]]>

Each template file has a <*Template*> construct placed in front of the other constructs. Any number of <*Transition*> and <*Place*> constructs follow the <*Template*> construct, while descriptive text can be placed in between constructs. In general, <*Template*> specifies global settings for current page generation. <*Transition*> specifies an embedded link that corresponds to a link in a current subnet. <*Place*> specifies a target place and display options for the output links of the place.

A detailed description of each construct in the template file is provided below.

a) The <*Template*> construct:

- <*Template*> specifies global settings for current page generation. Each template file has one <*Template*> construct placed in front of the other constructs (<*Transition*> and <*Place*>).
- *multiple_frame* sets for frame-based or frameless display. The frame-based display has three frames. The left control frame replicates the transition-associated links (called "caT links" in order to distinguish them from WWW

links). The controls in the top frame are provided for debugging purposes (selection of color tokens and applications). The main contents are shown in the main display frame that is located to the right of the control frame and below the top frame. When *multiple_frame* is set to “yes_2frame”, the frame-based display has two frames: the left and main frames.

- *linkbullet_show* specifies whether or not caT links should be displayed with a preceding bullet. If it is set to “yes”, this feature allows the user to differentiate caT links from normal Web links.

b) The <*Transition*> construct:

- <*Transition*> specifies an embedded link. This embedded link corresponds to a link in the current net within the template’s content. We also can use the <*Transition*> construct within the place’s content documents.

- *name* specifies a target transition in the current net. When there are several same-named transitions, a place name, in addition to a transition name, can be used together for identifying the target transition. Additionally, when *name* is a variable name (variable names have a ‘$’ in front of the “name” to produce “$name”), the value of the variable is used for display. Each subnet can have global variable/value pairs.

- *activelink_name* is used for specifying the link name for display when the target transition is active (or enabled). If the target transition is active and *activelink_name* is not specified, the *name* value is used instead.

- *inactivelink_name* is used for specifying the link name for display when the target transition is inactive. If the target transition is inactive and *inactivelink_name* is not specified, the link name will not be shown to the user. This allows dynamic display of links based on current net state.

c) The <*Place*> construct:

- <*Place*> specifies a target place and display options for its output links. The <*Place*> construct is replaced with the content of the target place only when the corresponding node is active. We expect that the user mainly uses the <*Place*> construct rather than the <*Transition*> construct unless he (or she) needs to control display layouts.

- *link_display* specifies an option for generation of a place's output links. When *link_display* is set to "all", all enabled output links are displayed. When it is set to "none", the links are not displayed. For selective display of the output links, explicit links can be specified using "show:'transName1',…,'transName*N*'", where *N* is an integer.

- *link_location* specifies the display location of the links. When *link_location* is set to "below", the output links are displayed below the content of the place. When it is set to "above", links are displayed above the content.

- *link_sort* specifies a sorting option for controlling the order of enabled links. When *link_sort* is set to "yes", the output links are displayed in alphabetical order.

- *link_duplicate* specifies whether or not output links are allowed to be duplicated. Since an enabled transition can be adjacent to more than one marked place—in other words, since a link may lead from multiple content elements—this *link_duplicate* specification is necessary. Additionally, some of the output links might already be generated using the <*Transition*> construct above the <*Place*> construct. When *link_duplicate* is set to "yes", the output links are displayed without considering link duplications.

As an example of using the template file, reconsider the *show_lion:#3* subnet in Figure 7. The content associated with the subnet's place, *template*, is a simple template file that directs the display of a full version of the page to an "on-campus" user and a limited one to an "off-campus" user. The template specification is as follows:

```
<Template multiple_frame="no">
<Place name="help" link_location="above">
<Place name="text" link_display="none">
<Transition name="$linkname">
<Place name="image">
```

The first line specifies that the output page will be frameless. The second line specifies that the output links attached to the *help* place will be displayed automatically before the content of the *help* place. The third line specifies that the output links attached to the *text* place will not be displayed automatically. Instead, the *text* place's content document has <*Transition*> constructs specified within it as follows:

This is the text file enclosed with the lion picture.
Here is where some explanations about the lion picture must be located. If you need help, click this <*Transition name=“show_help” activelink_name=“Show Help”*> link. You can hide the help if you click this <*Transition name= “hide_help” activelink_name=“Hide Help” inactivelink_name=“Hide Help (inactive)”*> link.

The above <*Transition*> constructs specify embedded links, which correspond to the *show_help* and *hide_help* transitions in the *show_lion:#3* subnet. In the constructs, *activelink_name* and *inactivelink_name* are used for specifying the link names for display when the target transitions are active and inactive, respectively.

In the fourth line of the template specification, the <*Transition*> construct is used for specifying an embedded link, which corresponds to one of the *$linkname* transitions in the current subnet. Since *$linkname* is a variable name, the value of the variable will be used for display. When no variable value exists, the default value, *Next*, is used for the link display. The last line specifies that output links attached to the *image* place will be displayed automatically after the content of the *image* place. Since *$linkname* is only an output link of *image*, and because it has been already specified in the fourth line, in the case of this example, no output link will be generated with this construct.

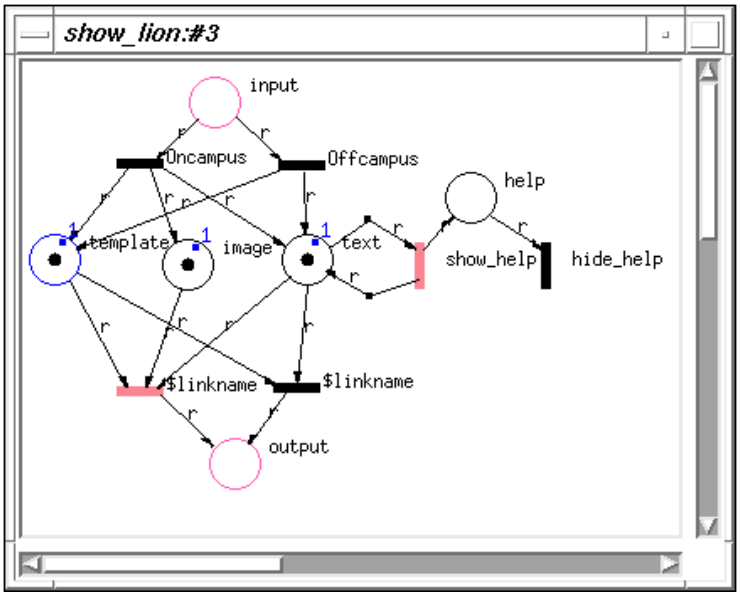
In Figure 7, when the transition *Oncampus* is fired for an “on-campus” user, the places *image, text,* and *template* are marked. The content associated with all three places is therefore used for generating the current Web page (i.e., the image and text both are shown, as structured by the template). The output Web document produced when the transition *Oncampus* is fired is shown in Figure 10. In the figure, the caT links that are

mapped to active transitions in the net are distinguished with a preceding circle bullet, while normal WWW links are shown without the preceding bullet. In the Web document, selecting a caT link fires the associated transition. This changes the state of the net and consequently the Web page display.

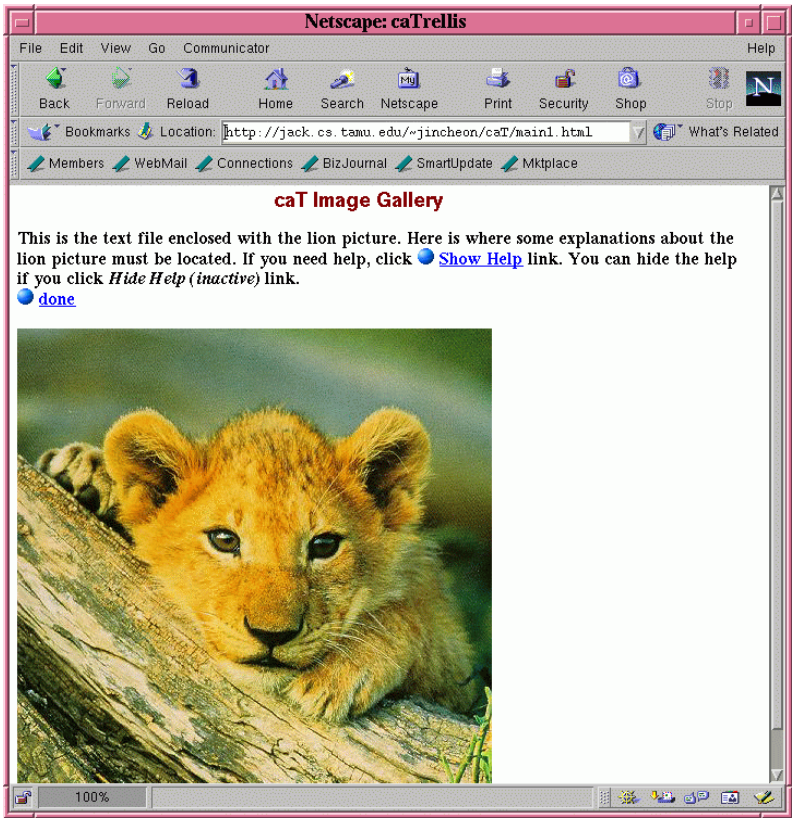
However, when the transition *Offcampus* is fired for an "off-campus" user, only places *text* and *template* are marked, so only their associated content is used in generating the Web page (i.e., no image data is shown to "off-campus" users). In both cases, when the *help* place is marked by firing the *show_help* transition, the content associated with the *help* place will be added to the displayed Web page. Figure 11 shows the output Web document when *show_help* is fired by an "on-campus" user. Note that the plain text string *"Hide Help (inactive)"* in Figure 10(b) is changed to an active link, *"Hide Help"*, in Figure 11(b).

3.2.5 Analysis of extended Petri nets

In the context of hypertext, one benefit of Petri net-based hypertext systems is that authors can use Petri net analysis techniques to verify and validate the behavioral characteristics of developed hypertext systems before they are handed over to users. In Trellis, Stotts and Furuta [1989a] explored these techniques, illustrating that it is possible to verify that all nodes in a hypertext can be reached via some path and that certain nodes cannot be reached from particular initial markings, giving the basis for access control. In addition, in the context of Trellis, Stotts et al. [1998] explored the analysis of basic Petri nets for various browsing characteristics.

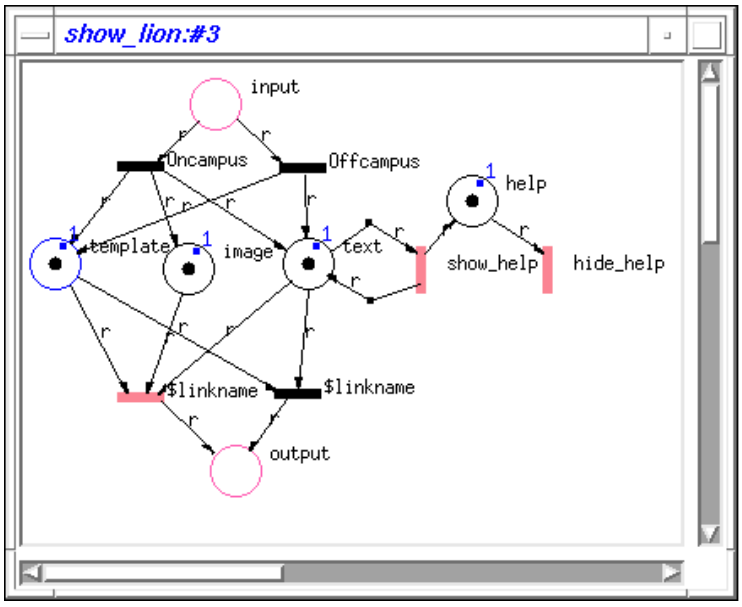


(a) when *Oncampus* is fired

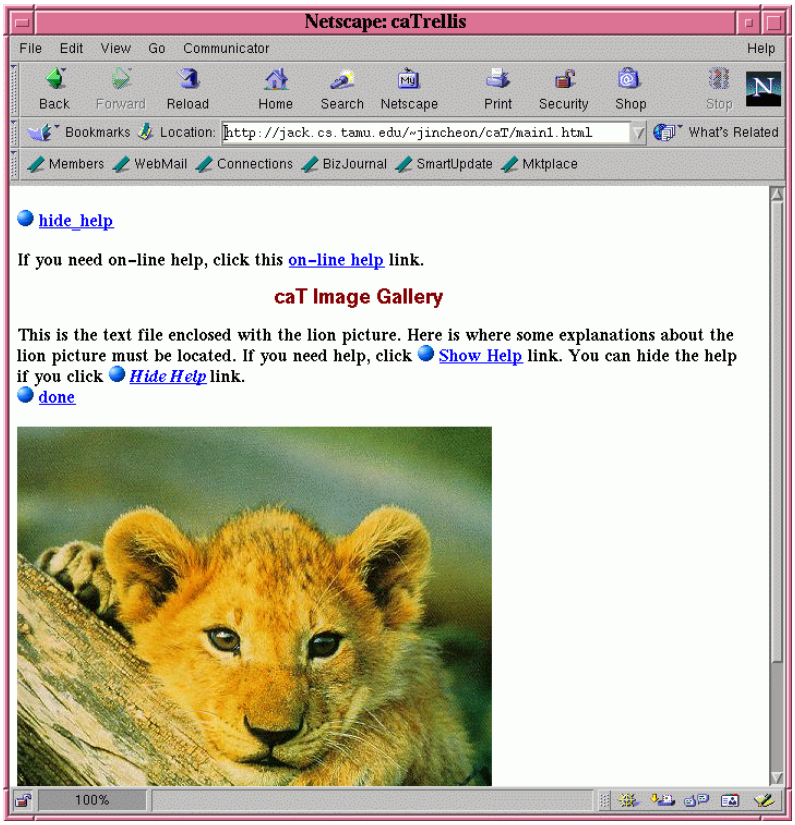


(b) Web display

Figure 10: Show lion for the on-campus user



(a) when *show_help* is fired



(b) Web display

Figure 11: Show lion for the on-campus user with help information

caT's extensions, such as hierarchical nets, structured tokens, and conditional predicates, have been designed to be consistent with Petri net theory. In caT, we have built two kinds of analysis tools, which are integrated into the authoring tool to support the author for net analysis. The first one is an interactive debugging tool and the second one is an analysis tool with which, by building a Reachability Tree (RT), the author can analyze the hierarchical Petri nets. The following characteristics can be verified by the analysis tools: liveness (i.e., if a state *m* exists in which no transitions are enabled), boundedness (i.e., if there is a place that has unbounded number of tokens), and safeness (i.e., if every place has only one token). In Petri net analysis, in addition to dead markings, boundedness is important since the tokens may represent limited system resources. For example, in caT, if a place has unbounded number of tokens, the content associated with the place may be undesirably displayed for all time. The place used for access control may also malfunction if it has unbounded number of tokens.

The debugging tool allows interactive analysis of the net while the author simulates the nets. After the debugging tool is activated, if a dead link or an overflow place occurs while the user fires transitions, warning message windows pop up. The system uses the current analysis option value for boundedness (i.e., a maximum token number in a place) to check overflow places. Additionally, the author can move backward from the current marking since each transition fire is saved when the debugging tool is active. After finishing the traversal, the author can replay the traversal path using a play button, which presents each saved marking sequentially, from an initial marking to an end marking,

with a specified time interval (e.g., two seconds). The author can stop the playing at any time to check the net status.

After building Petri nets, the author can construct a RT using the analysis tool. Before constructing a RT, the author sets analysis options: the maximum token number in a place, the maximum analysis time, the maximum marking number, and the maximum dead marking number. In general, the time complexity of a RT construction is very expensive (it can take even more than a day for large and complex Petri nets). Therefore, the author would likely start with moderate option values such as 3-minute analysis time rather than unlimited time. Similarly, if the maximum token number is not set, a RT may have infinite markings (especially when we analyze an unbounded net). Therefore, authors are likely initially to set the overflow limit number to a small number such as 5. The options modify the analysis tool's behavior in generating a RT. For example, if the number of tokens in a place reaches the limit number, adding tokens does not increase the token count and is flagged as an overflow for that place. The maximum marking number option allows specifying that the construction should finish when a certain number of marking nodes have been constructed. The maximum dead marking number allows specifying that the marking construction should stop when certain numbers of dead markings have been found.

When a RT is infinite or too large to be constructed by the available computing power, it may still be of interest to construct a partial RT; i.e., a subtree of the RT. As explained above, some analysis options allow the author to build a partial RT. However,

the analysis of a partial RT cannot give a total proof of the system correctness since there are those reachable markings and occurring steps that are not present in the RT. Nevertheless, a partial RT often allows the author to identify undesired properties of the modeled system.

In addition, since caT's net states can be affected by external, dynamic data sources (i.e., dynamic environmental data and the user model profile properties), we may need to simulate the external data's characteristics when building a RT. For example, in the caT specification to get a real time value we use *"send(time1.currTime())"*. However, if this function is used when we build a RT, it will take 24 hours of clock time in order to get all markings with different time values. Therefore, as a substitute, we use an integer-generation function that returns an integer in a specified sequence: *send(simuInt.getNext(instanceNum, startNum, endNum, increaseStep))*. For instance, the *"send(simuInt.getNext(0,0,24,12))"* function returns an integer in the following sequence: 0, 12, 24, 0, 12, 24, … . If the current net considers only rough time ranges (e.g., day and night), the above function may be enough. A float-generation function is also available: *send(simuFloat.getNext(instanceNum, startNum, endNum, increaseStep))*.

After building a RT, the author can see the analysis results, and return to the markings that cause problems. At this point the author can check the net status, including token values. In addition, the author can see marking lists (transition invocation sequences), and trace the marking history to find problems. In the next chapter, the

analysis tools will be discussed in more detail. To provide examples for these analysis tools, the gallery tour nets are analyzed.

CHAPTER IV

PROTOTYPE IMPLEMENTATION

4.1 Implementation details

The Trellis prototypes were implemented on the computer hardware and operating system that were most common in Computer Science departments at that time. Consequently, αTrellis was implemented in the late 1980's on the Sun workstation that runs Sun's operating system and proprietary windowing package. χTrellis, implemented in the first half of the 1990's, was implemented on a Sun, but under X windows. The specific widget packages used for χTrellis varied as to the time of implementation. χTed, the Petri net editor, was implemented with the Athena Toolkit, while `xtb`, the client that displays content, was implemented with the Motif widget set.

The history of Trellis' implementation, particularly the use of different widget sets in χTrellis, leads to user interface incompatibilities. Its client-server architecture makes invocation and configuration of the system more complicated than that of standalone applications. However, the separation between client and server, and especially the separation in χTrellis between Trellis engine and user interface, has the nice characteristic of providing a degree of resiliency to the server. In particular, buggy behavior by newly-developed clients in caT crashes the client, but generally does not result in loss or corruption of data since the server is unaffected.

caT is implemented as an extension to χTrellis and adopts that system's distributed client/server architecture model. Communication between client and server is through Remote Procedure Call (RPC). The server has no visible user interface, but it does have an API that allows other remote processes to invoke its functions for building, editing, annotating, and executing Petri nets. caT's clients are separate processes that generally have visual user interfaces and communicate with one or more engines via RPC. Clients collectively provide the necessary views, interactions, and analyses of a net for some specific application domain. The general architecture of the caT system is shown in Figure 12.

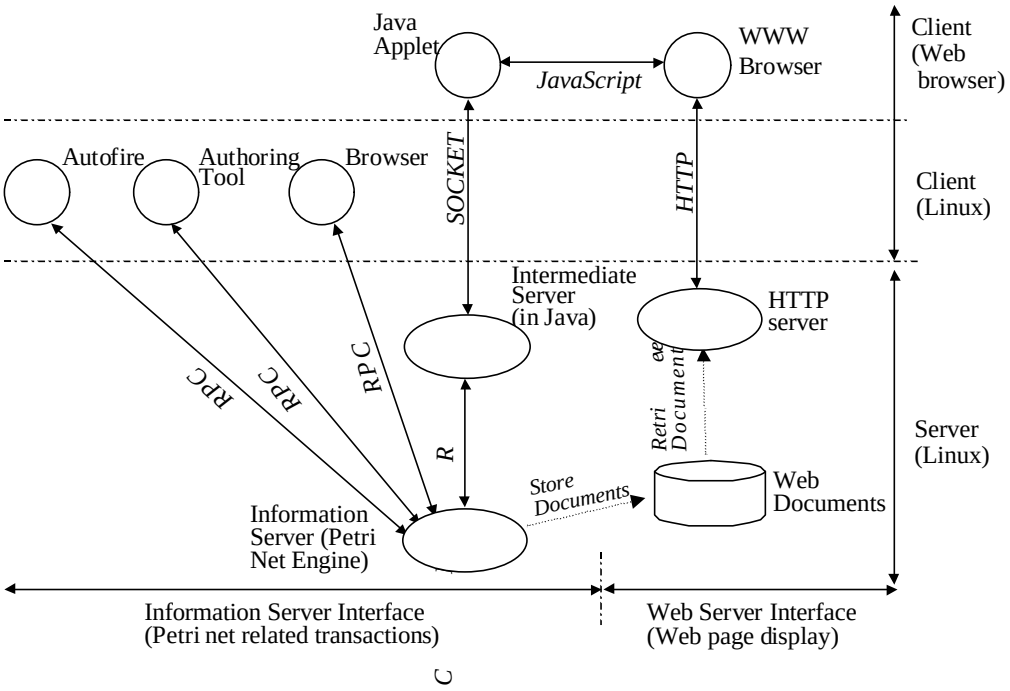


Figure 12: caT system architecture

Incorporation of the extended features, including hierarchical nets, required modification both to the Petri net engine and also to χTed. The incorporation of hierarchical nets, however, was mostly transparent to the `xtb` browser. Support for timed transitions had dropped from the χTrellis implementation, so an additional client with no visual user interface called `Autofire` was added to support automatic firing of transitions that have timing predicates. The detailed description of the system components (i.e., the server and client modules) will be described in the following subsections.

4.1.1 Information server

In Trellis, the information server (or the Petri net engine), which has no visible user interface, implements the Petri net-based model (i.e., Petri nets that have places, transitions, and arcs with annotation information) and provides the needed services that allow other remote processes to invoke its functions for building, editing, annotating, and executing Petri nets. The Trellis engine, called `netserver`, is written in C++. The server process runs in the background and provides services to zero or more client applications. Every `netserver` process runs with a specific net number that makes it possible to have more than one server running on a given machine. The server also allows clients to specify the net (document) with which they want to interact. Internally, the net number is used as the version number for RPC calls. In caT, the information server has been extended to handle its extended features such as structured tokens,

hierarchical nets, and flexible net description functions (e.g., condition statements for transitions, assignment statements for arcs, and fuzzy logic tool invocation).

In the server, a list of attribute/value (A/V) pairs is kept for each place, each transition, each arc, and for the net as a whole. Some attributes are intrinsic to the object to which they belong, whereas other attributes may be used by some specific application or the server itself. The attributes for place, arc, transition, and net are listed and briefly described in Table 4, Table 5, Table 6, and Table 7, respectively. Some attributes are marked with a specific client application name, enclosed in square brackets, when they are used only for the application—for example, [Web] indicates the attributes used only for Web browsing. In addition, the attributes that are generated and managed only by `netserver` are indicated in bold face and the attributes that are provided by caT but are not by χTrellis are indicated with the "*" character. Most of the attributes in the tables are easy to understand and, even though explicit attribute names have not been mentioned, most of them have been already described in the previous chapters. Some new attributes will be discussed in the later sections of this chapter.

Trellis provides a simple language for external representation of the Petri net structure, which has been designed to be parsed in one pass. Thus, using the net description file, the net structure can be loaded into the server process as the initial net structure. Also, the current net structure and its marking status can be stored in the net description file. In caT, the simple language has been extended to represent structured tokens. Each subnet is represented in a separate net description file. Therefore,

Table 4: Place attributes

Attribute	Description
Contents File Name*	Name of the file for the contents of the place URL (http) can be specified also for the contents of the place especially for Web browsing.
External Viewer [`xtb`]	Name of the application called to render the file specified as contents (if null, it assumes simple text content)
Label	Name of the place
Location (X)	Position of the place in the drawing area (x-coordinate)
Location (Y)	Position of the place in the drawing area (y-coordinate)
Template File Name [Web]*	Name of the template file
Note: the following notes are applied to Table 4, Table 5, Table 6, and Table 7. *: indicates an attribute that is provided by caT but not χTrellis ('*' also indicates an attribute that is extended by caT) [`client name`]: indicates an attribute that is used only for the named client	

Table 5: Arc attributes

Attribute	Description
Expression (assignment)*	Assignment expression attached to the output arc for updating current token values when the transition is fired
Token Expression*	Variable expression used by the fire procedure
Hide Button [`xtb`]	List of colors for which the arc is closed. The value "all" is used to hide the arc for all colors. This is used when one wants a button to be visible only from some of the places enabling the transition for a particular color.
Label	Name of the arc

Table 6: Transition attributes

Attribute	Description
Condition*	Conditional expression that specifies an additional constraint (threshold) that must be fulfilled before the transition is enabled
Label	Name of the transition
Location (X)	Position of the transition in the drawing area (x-coordinate)
Location (Y)	Position of the transition in the drawing area (y-coordinate)
Orientation	Transition orientation (VERT, HORIZ)
Time*	Pair of time values termed release time and maximum latency respectively When a time value is not declared, "(0,∞)" is the default value.
Subnet Display Values*	List of variable/value pairs (especially, "Contents File Name" and "Label" values) that are passed to the subnet to have different behavior for instances of the same net definition
Subnet File Name*	Name of the subnet file for the hierarchical net specification When a transition has a subnet file name (called substitution transition), it will be expanded until there are no substitution transitions specified in the subsequent subpages.
Subnet ID*	Subnet identification number
Subnet Input/Output Map*	List of variable/value pairs used for mapping input/output places of the substitution transition in the current page to the places in the target page
Substitution	List of one or more sets of values for the variables in case that the transition can be fired

Table 7: Net attributes

Attribute	Description
Net File Name*	Name of the current net description file
Net Window Title Name*	Name of the current canvas window title
Token Structure*	Token type value (a list of token variable/value pairs) used for token instantiation (or creation)
__Annotation*	Annotation for the current net

hierarchical nets may be represented using several net description files, which are hierarchically connected.

Table 8 shows the syntax for the net description file. Nonterminals are in <BRACKETS>, and terminals are in ‘quotes’. The term *integer* represents a digit sequence, and all <UID> are assumed to be completely unique. The format used to store the net file is ASCII. This ASCII format is easy to read, so it can be modified using any text editor. However, the net file is usually not edited manually since the authoring tool takes care of generating (or updating) it. Additionally, the user profile, used to support user modeling, is represented as a plain text file. Each profile for a user has a list of attribute/value pairs.

4.1.2 Authoring tool

Although graphical editing is only one of many possible ways to generate Petri nets, Trellis prototypes have used a graphical net editor for specification of the hypertext. An early investigation in the Trellis project defined a textual C-like language, called Alphα [Stotts and Furuta 1989b; Stotts and Furuta 1989c], which could be used to define Trellis hyperprograms. This specification mechanism was not included in Trellis prototypes because of practical issues—for example, determining how to present properly formatted graphical representations of the net.

Table 8: Net description file

<FILE>	::= <VERSION> <NET> <DESCRIPTION> <MARKING>
<VERSION>	::= '#V' *integer* anything_else <EOL> *# version is the version number of the description language*
<EOL>	::= CR | NL | TAB
<NET>	::= 'n' <UID> <ATTR_VAL_LIST> ';' *# <UID> identifies a net*
<UID>	::= *integer*
<ATTR_VAL_LIST>	::= <ATTR_VAL_LIST> <ATTR_VAL> | empty
<ATTR_VAL>	::= '(' <ATTR> ',' <A_VAL> ')'
<ATTR>	::= a sequence of chars formed from numbers, letters, and a underline character
<A_VAL>	::= '"' a sequence of chars '"'
<DESCRIPTION>	::= <ITEM> <DESCRIPTION> | empty
<ITEM>	::= <PLACE> | <TRANSITION> | <ARC> | <COMMENT>
<PLACE>	::= 'p' <UID> <ATTR_VAL_LIST> ';'
<TRANSITION>	::= 't' <UID> <ATTR_VAL_LIST> ';'
<ARC>	::= 'a' <UID> <ARC_INFO> <ATTR_VAL_LIST> ';'
<ARC_INFO>	::= '[' <UID> ',' <UID> ',' <UID> ']' *# the first <UID> identifies a net; the second is the source of the arc; and the third is the destination of the arc*
<COMMENT>	::= '#' arbitrary characters <EOL>
<MARKING>	::= 'm' <UID> '[' <UID> ']' <MARK_INF_LIST> ';' *# the second <UID> identifies a net*
<MARK_INF_LIST>	::= <MARK_INF_LIST> <MARKING_INF> | empty
<MARKING_INF>	::= '<' <UID> <TC_LIST> '>' *# <UID> is for a place*
<TC_LIST>	::= <TC> <TC_LIST> | <TC>
<TC>	::= ',' *integer* ':' *integer* | ',' *integer* ':' *integer* '{' <ATTR_VAL_LIST> '}' *# the first is the number of token; the second is token color number*

The Trellis Petri net specification is at a single level. The structure, as a whole, is displayed in a single window, which can be scrolled if necessary. Even with scrolling, the physical limitation of the computer display size imposes a practical limit on the number of elements that can be shown simultaneously. Additionally, the required arcs connecting graph elements create significant visual clutter. Effectively, these factors mean that only relatively small-sized specifications can be created and maintained in αTrellis and χTrellis. caT's authoring tool is designed to manage the problem by displaying hierarchical Petri subnets in multiple windows.

The authoring tool has been developed as a modified version of the χTrellis χTed editor. We have converted the editor to use the same Motif widget set as `xtb`. Actually, we use LessTif [Lesstif.org]. LessTif is free software that is source compatible with OSF/Motif 1.2. However, Motif does not support the MDI (Multi-Document Interface) user interface that is essential for multiple windows handling of hierarchical structure. Therefore, in addition to Motif, we use a MDI package for Motif [Sadler 1996] with slight modifications. The MDI package for Motif is a collection of C++ classes that emulates the behavior of the Multi-Document Interface.

4.1.3 Flexible information presentation tools

caT's Web client required the inclusion of a new intermediate message-handling server to translate between Web-browser-based applets and information servers. When a reader clicks a generated link on a Web page, the server's net state must be updated (i.e., the corresponding transition should be fired). To accomplish this, the Java applet first

communicates with the message-handling server. This server then passes RPC requests using Java JNI (Java Native Interface) [Sun Microsystems, Inc. 2001] to the information servers. Also, when other clients change the server's net state, the changes are reported to the corresponding applets via the message-handling server.

The incorporation of hierarchical nets also required slight changes to the original χTrellis `xtb` browser, written in C++, to allow communication with a caT engine that has multiple subnets (the caT `xtb` browser invokes one RPC function for updating each subnet state). `xtb` would need no changes if we had implemented the incorporation of hierarchical nets entirely on the server side. In addition, the image browser, which displays only the image contents associated with each active place, has been developed as a modified version of the χTrellis `xtb` browser.

4.1.4 Analysis tool

In caT, two kinds of tools have been developed and integrated into the current authoring tool to support the author's net analysis. The first is an interactive debugging tool, and the second is an analysis tool that can construct a reachability tree (RT) of an arbitrary hierarchical Petri net. The basic idea behind the reachability tree (also called the occurrence graph) is to construct a graph that has a node for each reachable marking and an arc for each occurring transition. Therefore, authors can use the analysis tool to construct a RT, and verify behavioral characteristics of developed hierarchical Petri nets with the constructed RT.

In a RT construction, the practical limitation is not only determined by the time complexity (i.e., the RT construction time), but also by the space complexity (i.e., the memory for storing the constructed RT). A RT grows very fast when the number of colored tokens increases. Thus, it is wise to consider rather small number of colored tokens in order to verify the logical correctness of a given Petri net [Jensen 1995]. A RT may become very large, even for small Petri nets. Many high-level Petri nets, in fact, have millions of markings [Jensen 1995]. However, manual testing and debugging is often much slower than the RT analysis, and less reliable.

For reachability tree construction, we generally follow the algorithm introduced in Jensen [1995]. We do, however, add an additional feature for handling the unbounded net. Jensen's algorithm shown below halts if and only if the RT is finite. Otherwise, the algorithm continues forever, producing an ever larger RT.

```
WaitingList := ∅
CreateNode(M0)
Repeat
   Select a node M1 ∈ WaitingList
   For all (b,M2)∈ NextNode(M1) do
   Begin
        CreateNode(M2)
        CreateArc(M1,b,M2)
   End
   WaitingList := WaitingList – {M1}
Until WaitingList = ∅
```

In the algorithm, *WaitingList* is a set of nodes. It contains those nodes for which we have not yet found the successors. *CreateNode(M)* is a procedure that creates a new node

M, and adds *M* to *WaitingList*. If *M* is already a node that exists in the generated RT, the procedure has no effect. Analogously, *CreateArc(M_1,b,M_2)* is a procedure that creates a new arc *(M_1,b,M_2)* with source M_1 and destination M_2. If *(M_1,b,M_2)* is already an arc, the procedure has no effect. For a marking M_1, we use *NextNode(M_1)* to denote the set of all possible "next moves", such as *(b, M_2)*. To avoid infinite markings, especially when we analyze an unbounded net (i.e., some places have infinite number of tokens), the option value *a maximum token number* can be set to a reasonably small number such as 5. In this case, if the number of tokens in a place reaches the limit, adding tokens does not increase the token count, and an overflow for that place is flagged. Therefore, our implementation of *NextNode(M_1)* actually returns markings that have at most the specified maximum number of tokens in places. This feature prevents infinite marking generation of the unbounded net. In the analysis of basic Petri net [Zurawski and Zhou 1994], the symbol ù, which can be thought of as infinity, is used for handling the unbounded net. *"a maximum token number"* in caT is used in similar way as the symbol ù. However, *a maximum token number* specifies explicitly when the place reaches the token limit, and the symbol ù is managed automatically by the system. We use *a maximum token number* because it is easier to implement and provides more control on the user side (an option for the specification of token limit).

In the constructed RT, a lot of markings will be nearly identical. Hence, we may save a significant amount of memory if we avoid duplication of identical parts. To do this, we represent each marking as a set of pointers, as depicted in Figure 13, where we follow the analysis tool design techniques described in Jensen [1995]. Marking records have a

set of pointers to page records, the page records have a set of pointers to place records, and the place records store actual token information (see Figure 13). In other words, a marking record has pointers to its page records that have page (subnets) marking information; a page record has pointers to its place records that have place marking information. In Jensen [1995], the marking of a page is represented by two records for each page instance—one for input/output mapping places that depend on other places in other pages and one for the other places in the current page. However, for the purpose of simplification, in caT, the marking of a page is represented by a single page record.

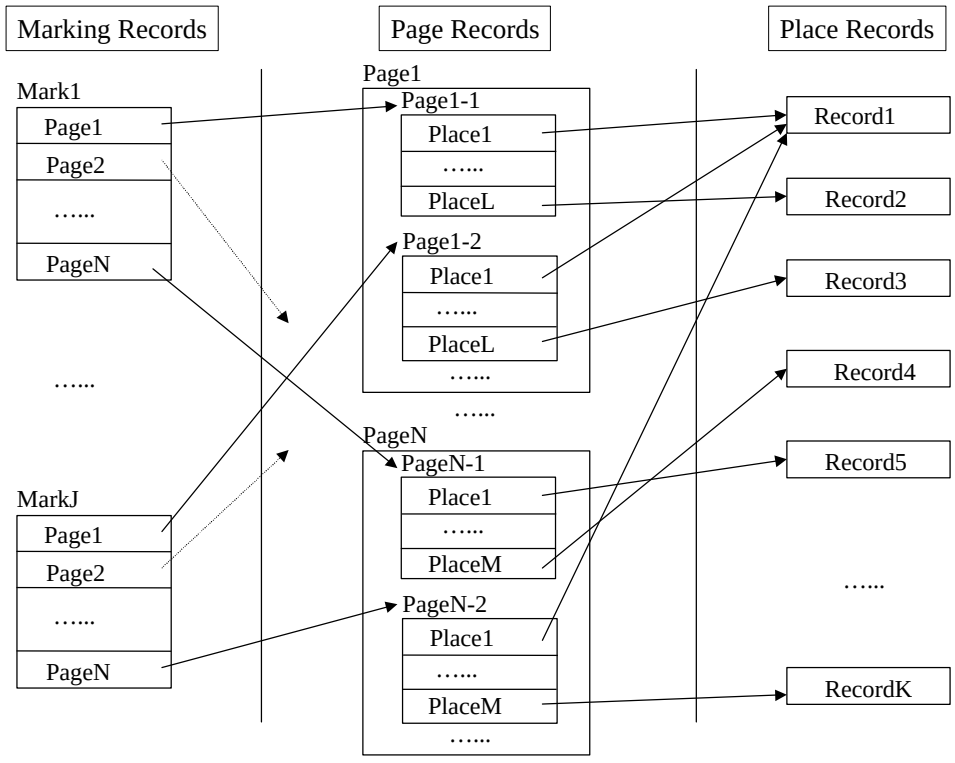


Figure 13: Marking structure

Each place record represents a unique (possible) token marking for a place. This means the unique token marking for a place (i.e., a place record) is created once, even though it may appear in many different places of page records. When we construct a new marking from an existing marking, normally we need to create a small number of new page records. This means that we can reuse most of the page and place records, and, finally, save a lot of memory space—Jensen [1995] says that a RT for a sample net only needs 0.35% of the space it would need without this scheme. For example, *MarkJ* could be a marking obtained from *MarkI* by the firing of a transition that belongs to *Page1*. If the transition does not have any input/output places that are related (or mapped) to its parent or child places, it is obvious that only the record for *Page1* needs to be changed, while all the other page records can be reused.

According to the RT construction algorithm in the *CreateNode(M)* procedure, whenever a new marking has been constructed, we must check whether the marking is identical to any of the existing markings. To do this efficiently, we keep all records (marking, page, and place records) in binary search trees. There are separate binary trees for marking and place records, and for page records there are separate binary trees for each subnet (or page). To be positioned in a search tree, each record is given an integer key calculated from the contents of the corresponding marking, page, or place records.

The current key function of a place instance record is the summation of each color value, between 0(*black*) and 31(*purple4*), multiplied by the number of its tokens. The key function of a page record is calculated from the keys of the involved place records.

Analogously, the key function of a marking record is calculated from the keys of the involved page instance records.

For the implementation of the interactive debugging tool, the marking structure developed for the RT construction is used along with the addition of a marking list structure that stores only ordering information of marking records. In the debugging mode, when a transition is fired, the current marking is saved to the marking records tree, and, in addition, the address of the marking record is added to the end of the marking list structure.

4.1.5 Porting to Linux

Recent years have seen a broadening of the computer architectures in common use in Computer Science. Most visibly has been the increase in PC-category computers running a variety of operating systems. Thus, one factor limiting wide-spread dissemination of Trellis is its increasingly uncommon run-time environment.

The caT system has been ported to Linux (Red Hat Linux 6.2) in order to make it available to a wider group of users. The Trellis system uses ONC RPC, as part of the UNIX System V Release 4 (SVR4), in Sun workstations—there are several RPC standards such as ONC RPC (also known as Sun RPC), OSF DCE (based on Apollo's NCS system), Xerox Courier, and Netwise. In Red Hat Linux, ONC RPC is also supported; therefore, it was straightforward to port RPC related modules to the Linux system.

However, there are some differences between `rpcgen` versions in SunOS and Red Hat Linux. Generally, the `rpcgen` tool generates remote program interface modules such as stub programs for the server and client, and it significantly reduces the development time that would otherwise be spent on low-level routines. The `rpcgen` in SunOS can generate MT(Multi Thread)-safe code for use in a threaded environment, and it can generate MT-safe stubs that enable RPC servers to automatically use Solaris threads (the predecessor to POSIX threads) to process client requests concurrently in releases since Solaris 2.4 (Sun's operating system and its window environment). However, the `rpcgen` in Red Hat Linux does not support the generation of MT-safe stubs for multithreaded PRC servers. Additionally, the `rpcgen` tools in both systems support new features, such as TI (Transport-Independent)-RPC and C-style mode. TI-RPC makes RPC applications transport-independent by allowing a single binary version of a distributed program to run on multiple transports. C-style mode lets arguments be passed by value instead of as pointers to a structure. It also supports passing multiple arguments.

4.2 Interactions with the authoring tool

For structured authoring support, the χTrellis χTed editor has been extended to allow for the building and browsing of hierarchical subnets in multiple windows. In the new authoring tool, the editing functionality of Petri net objects is provided by the original χTrellis χTed editor implementation. In caT, the author, using the multiple-window-

based authoring tool, first builds a Petri-net-based hierarchical document structure, and then associates net segments with the corresponding content segments.

The χTed interface consists of menus in the top area, a series of buttons located in the left area of the window, and multiple graphic canvases where the nets are graphically displayed (see Figure 14). In addition to the main menu items, a menu pops up when the mouse is right-clicked on each drawing (or canvas) area (see Table 9). The functionality of the menu items is described briefly in Table 10. The descriptions for the operations of icons in the left area of the window are in Table 11 and Table 12. Table 11 describes the meaning of icons that are used for editing Petri net objects (place, transition, arc, and token) and how to add objects to or delete them from the canvas. For the functions represented by the pictorial editing icons, when selected, the cursor changes shape to become a mnemonic aid. Table 12 describes the meaning of icons for net simulation and recording (or debugging) while detailing how to simulate and record the nets.

Table 9: Popup menu of cTed canvas window

Menu Item	**Description**
Annotation	Show the annotation for the current net; when an annotation is defined, the character "A" is shown at the top left corner of canvas.
Net Attributes…	Show current net attributes
Save	Save the current active document as the current description file name; the active document has a blue-colored window title.
Save As…	Save the current active document as a new description file name

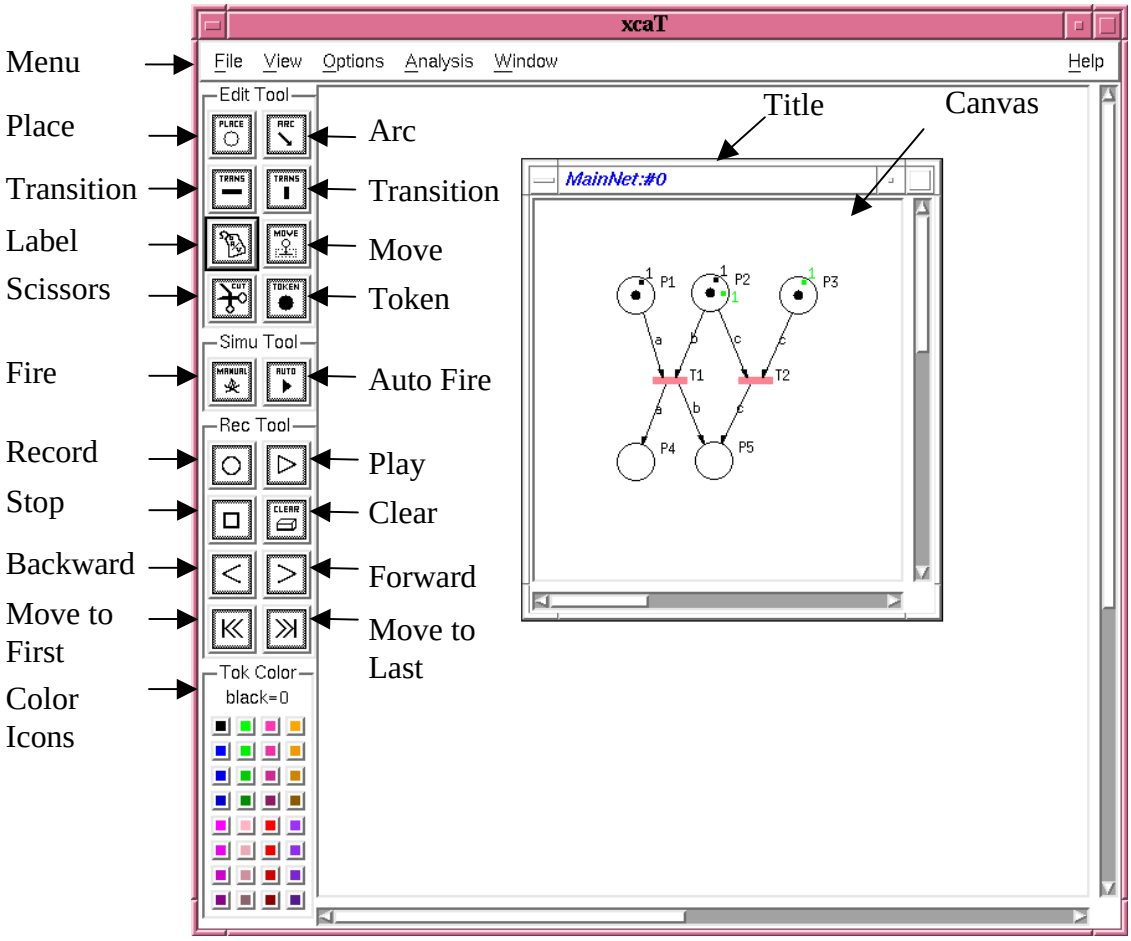


Figure 14: cTed main window

Table 10: Menus of cTed main window

Top Menu	Sub Menu	Description
File	*New*	Discard the current document structure and make an empty net
	Open…	Load new document structures into the engine from a net description file (the extension of the net file is "net")
	Save All Pages	Save the current running documents as description files
	Exit	Exit the editor
View	*Hierarchical Net Refresh*	Reload all current nets from the engine for refreshing all current windows
	Transition Refresh	Refresh all transitions from the engine (enabled or disabled)
Options	*Analysis…*	Set the options for the construction of the reachability tree
	Appearance…	Set coloring schemes for displaying graphical objects (place, transition, background, etc)
	Autofire…	Set the polling interval for automatic invoking of enabled, timed transitions
	Play…	Set the polling interval for automatic play of recorded markings
	Poll…	Set the polling interval for updating current nets from the engine
	Text Editor…	Set a text editor for viewing/editing text contents associated with places. The default editor is "/usr/bin/ gedit" (the editor is specified in a configuration file).
Analysis	*Construct Reachability Tree*	Construct a reachability tree
	Show Net Properties…	Show the analysis results after the reachability tree construction
	Show Mark lists…	Show the mark lists of the reachability tree
	Destruct Reachability Tree	Destruct the reachability tree
Window	*Cascade Windows*	Cascade the windows; each window's title turns into a red color if the window contains active transitions in it.
Help	*Help*	Show a short summary of commands
	About caT	Show the current version information

Table 11: Icons for editing

Icon	Description
Place	Add new places to the canvas; to add new places to the net, click the *Place* icon to go into adding places mode. The cursor will change to a circle (place symbol) to reflect this. Then, move the cursor to the place where you want to put a new place, and click the first button. Repeat as many times as you want.
Arc	Add new arcs between places and transitions; to add arcs, select the *Arc* icon, and click first over the transition (place) the arc is coming from, and then click again over the place (transition) the arc is going. It is possible to include knuckles in the arcs by clicking in between the two object involved.
Transi-tion	Add new transitions to the canvas; to add new transitions to the net, click the *Transition* icon to go into adding transitions mode. The cursor will change to a bar (transition symbol) to reflect this. Then, move the cursor to the place you want to put a new transition, and click the first button. Repeat as many times as you want. There are two transition icons available. One of them produces a horizontally oriented transition, whereas the other generates those that are vertically oriented.
Label	Add/delete/change attribute variable/value pairs (except net attributes); in order to check/add/delete/change the attributes of the objects of the net, the *Label* icon must be selected first. The cursor will change to a label-like form. After clicking on the object, a dialog window is displayed. Especially, the first mouse button click on the large dot in the center of the place pops up a dialog window for showing the token contents of the place.
Move	Rearrange the nets; in order to rearrange objects, the *Move* icon must be selected first. The objects can be dragged with the first button of the mouse. It is also possible to drag the canvas itself. Just click the mouse over a place where there is no place or transition, and move the mouse while holding the first button pressed.
Scissors	Delete objects from the net; to delete places, transitions, or arcs, select the *Scissor* icon, and click over the corresponding object. When a place (or transition) is deleted, so are all the arcs starting or ending on it.
Token	Add/delete tokens; to add tokens to a place, first select the desired color using the *Color* icons (palette), then click over the token icon. The cursor will change to reflect token mode. Then, clicking inside a place will add one token of the current color to it. Tokens can be removed in the same way by clicking the third mouse button instead.

Table 12: Icons for simulation and recording

Tool Group	Icon Name	Description
Simulation Tool	Fire	Fire transitions To fire a transition, select the *Fire* icon, and click over the active transition; in case of more than one possible token color combination, a menu is presented to the user to select one of the appropriate variable instantiations. Substitution (or subnet) transitions cannot be fired; clicking the third mouse button instead can raise the target subnet window of the substitution transition.
	Auto Fire	Fire enabled, time-based transitions automatically with the specified interval
Record Tool	Record	Record current transition fire sequences To start recording of simulation (recording mode), select the record icon, then click over the active transitions for simulation. In recording mode, objects (place, transition, arc) cannot be added or deleted.
	Play	Play recorded transition sequences with a specified polling interval for automatic play
	Stop	Stop current play or recording
	Clear	Clear current stored sequence of markings
	Backward	Move backward The semantics of the backward (forward) icon are similar to the semantics of the backward (forward) button in Web browsers. In recording mode, when you move back to a previous marking and click an active transition, the markings from one step ahead from the current point to the right most point where you came from by doing backward icon clicks are removed from the stored markings. The new marking is then added to the end of the stored markings.
	Forward	Move forward
	Move to First	Move to the first net marking
	Move to Last	Move to the last net marking
Token Color	Color icons	Select the current token color 32 different colored tokens are available.

In the authoring tool, to add/change/delete the attribute values of existing objects (the place, transition, and arc), the *Label* icon must be selected first. Then, for example, a left click on a transition pops up an attribute dialogue for the transition (see Figure 15). In the dialog, we can see the values of attributes by clicking items in *Attribute List*. We can then add/change/delete attribute values if it is necessary (the description of each transition attribute is provided in Table 6 in section 4.1.1).

The several *Value buttons*, shown in Figure 15, are used for handling the attribute value in *Value Field*. The “Get File” and “View/Edit File” buttons, respectively, allow getting a new file name with a file dialog interface and viewing (modifying if possible) the current file content with an external text editor. The “Input/Output Map” button is for specifying the input/output mapping pair values of the substitution transition. The “Display Values” button allows seeing (or adding) the currently defined variable/value pair values, which will be passed to the related subnet, thereby enabling different behaviors of the same net. Finally, the “Cut”, “Copy”, and “Paste” buttons are used for copying of the current attribute value in *Value Field* to other attribute values.

To add/change/delete the token attribute values, the *Label* icon must be clicked first. Subsequently, a left click on the large dot in the center of the place that has tokens opens a token attribute dialog (see Figure 16). In the dialog, at first, we need to select a token color in Token *Color List*. Then, when an attribute in Token *Attribute List* is selected, the attribute value will be shown in *Value Field*, and we can add/change/delete the attribute values if necessary.

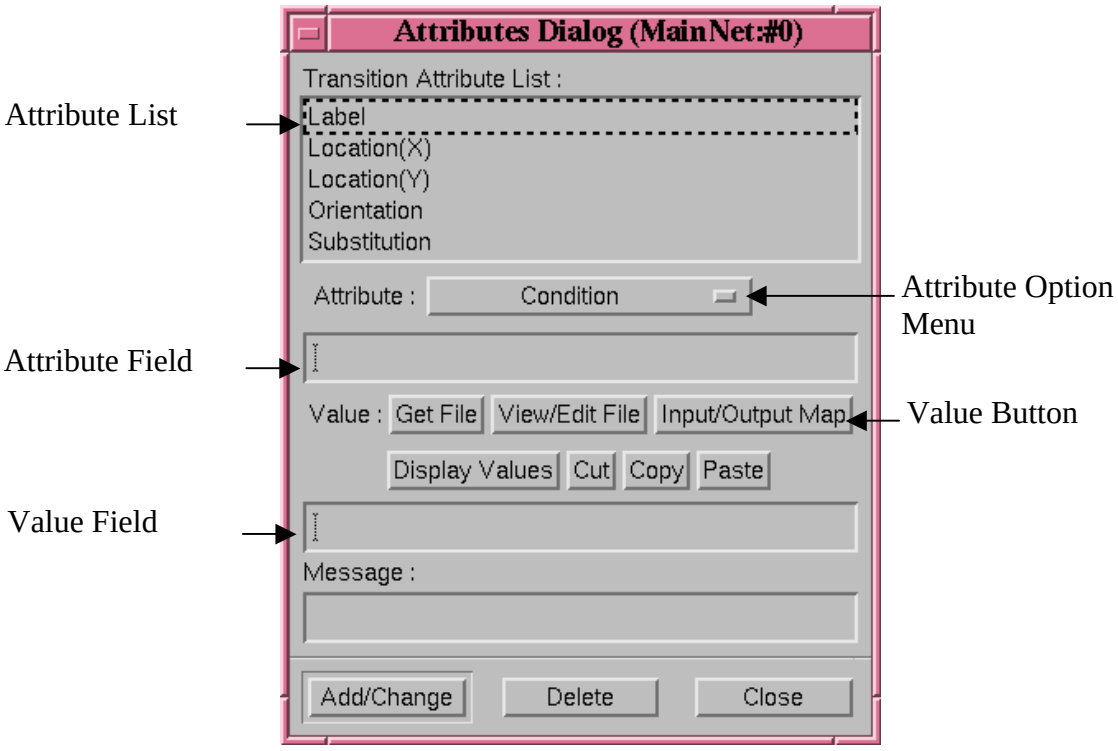


Figure 15: Transition attribute dialog

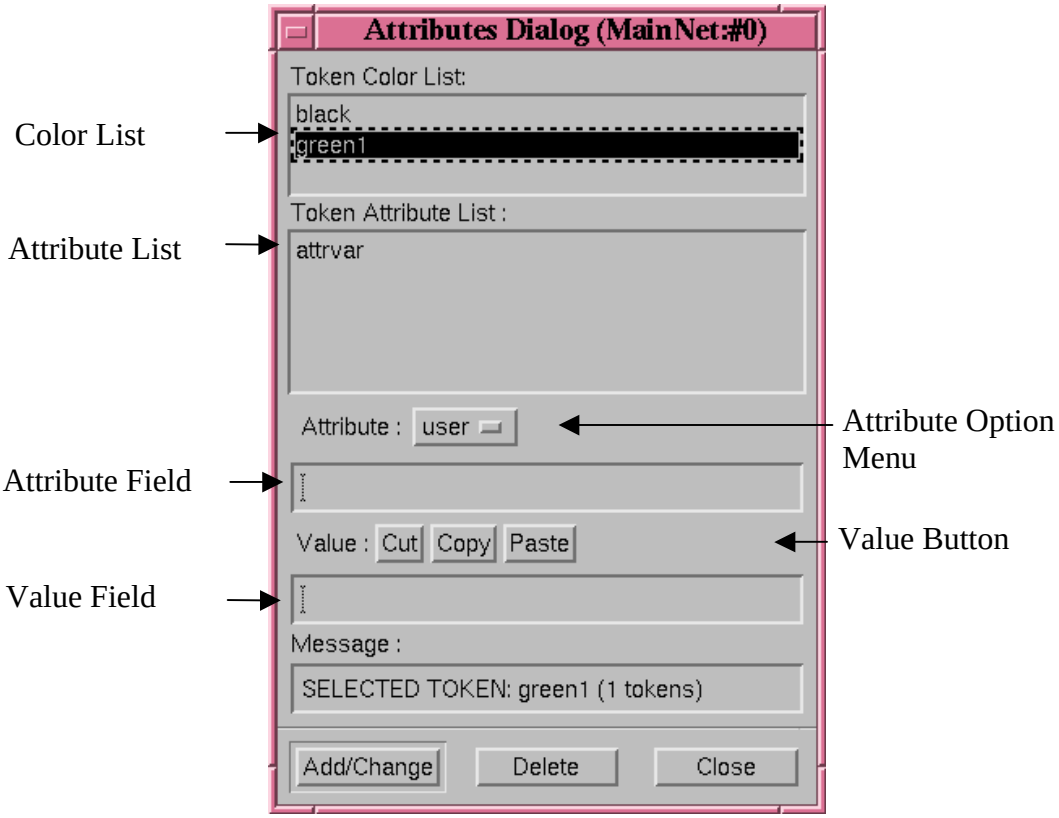


Figure 16: Token attribute dialog

To create hierarchical nets (i.e., to associate a normal transition with a lower-level subnet), additional attribute values are added to the transition. To provide an example, we will again revisit the gallery tour example introduced in Chapter III. To expand the *Classify_user* transition to the "classify_user.net", which is shown in Figure 6 (in section 3.2.2.4, Chapter III), we must select the *Label* icon first. Then, when we click on the *Classify_user* transition, the transition attribute dialog (see Figure 15) pops up. In the attribute dialog, we need to select the "Subnet File Name" attribute from *Attribute Option Menu*, and use the "Get File" button located above *Value Field* to get the "classify_user.net" file name from the Get File Dialog. When *Value Field* has the target subnet description file name and the "Add/Change" button is pressed, the subnet *Classify_User:#6* is opened (or created).

For the next step, we need to specify the input/output place mappings. In the previous transition attribute dialog, we first need to select the "Subnet Input/Output Map" attribute from *Attribute Option Menu*. Then, the "Input/Output Map" button located above *Value Field* is used for opening the "Subnet Input/Output Map Dialog" window. This window helps specify the input/output mapping pair values (see Figure 17). In the popup dialog window, we first select an origin input place name, *ClassifyUser*, from *Origin Page Place Option Menu*. Then, we select a target input place name, *CalcTime*, from *Target Page Place Option Menu*. To add the current value pair, the "Add/Change" button is used. We repeat this process with an origin output place name, *EndClassify*, and a target output place name, *Branch*. After closing the window, the defined value pairs will appear in *Value Field*. Finally, the "Add/Change" button is

used to add the current value to the current transition. The color of mapped places will be turned into "*maroon*" in order to distinguish them from the other normal places.

In the authoring tool, the user can select one of the system-provided color schemes, or he (or she) can define the colors using the color selection toolbox. The tool provides the following color settings: foreground, background, active transition, active subnet transition, input/output subnet node, template node, and active window title. For example, each canvas window's title turns into a color that is specified by "active window title" if the window contains active transitions.

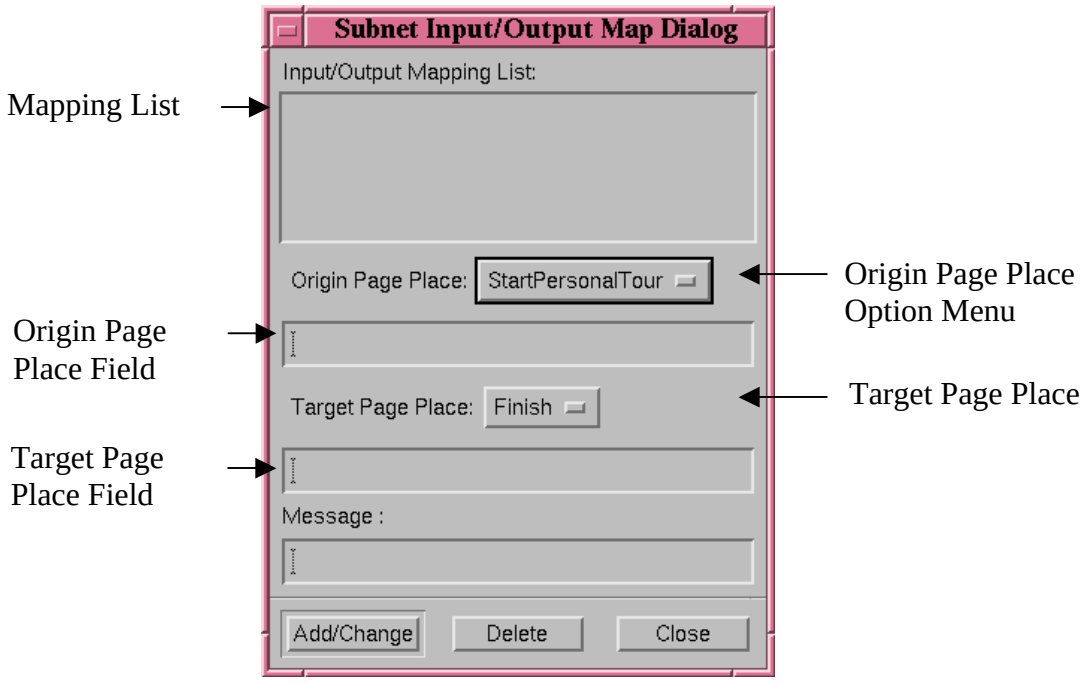


Figure 17: Subnet input/output map dialog

4.3 Interactions with the analysis tool

After building a RT using the analysis tool, the author can see the result of the analysis. The author can set the current net to the marking that causes the problem, and check the net status, including the token values. In addition, the author can see marking lists, and trace the marking history to find problems.

As an example, we will consider the analysis of the gallery tour example nets. The nets consist of six subnets, and one "*blue1*" token is in the *ReaderPool* place of the *Start_tour:#1* subnet (see Figure 6 in section 3.2.2.4, Chapter III). The token has the following initial (attribute, value) pairs: (accessRight, 0.0), (class, unknown), (currentTime, 0.0), (network, 128.0.0), and (user, unknown). Before constructing a RT, the author should set analysis options: the maximum token number in a place, the maximum analysis time, the maximum marking number, and the maximum dead marking number. Figure 18 shows the current option values for the RT generation. The maximum token number in a place is 5 tokens, the maximum analysis time is 5 minutes, and the other options are not set. The maximum token number option is especially important because the example nets are unbounded nets—in the unbounded nets, some transitions generate an infinite number of tokens in their output places.

To construct a RT, the "Construct Reachability Tree" submenu item in the "Analysis" menu (see Figure 14) is used. After the construction, the "Show Net Properties" submenu item shows the analysis result. Figure 19 shows the results of analysis of the gallery tour nets with the current analysis option values.

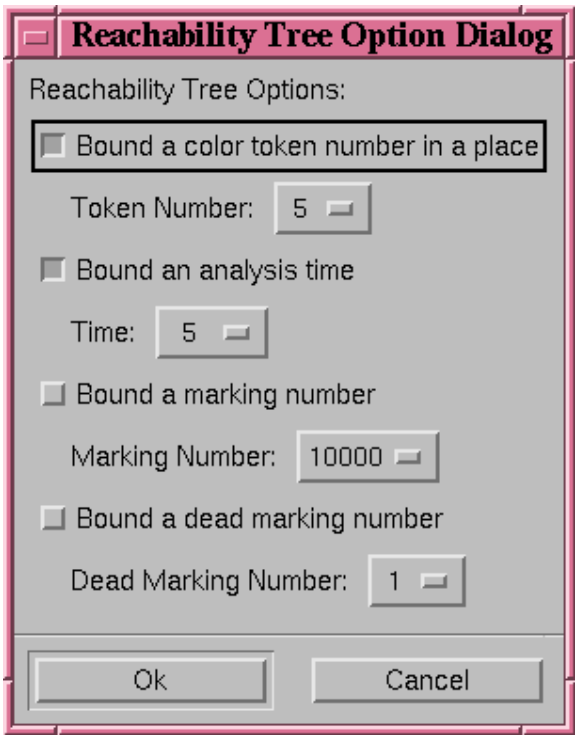


Figure 18: Analysis options

Figure 19 shows the three result properties of the analysis: liveness, boundedness, and safeness. It says that there is one dead marking out of total 521 markings. The user can set the current net to the dead marking by selecting a mark (*"Mark36"*) in the liveness list and clicking the "Go to Marking" button. Actually, the dead marking occurs when the reader finishes a reading session; i.e., when the *FinishTraverse* place in the *Start_tour:#1* subnet has a token. Additionally, it says that the net is not bounded and that the maximum token value encountered is 5 (the limit value was chosen earlier).

Note that the “help” places in pages (or subnets) 3, 4, and 5 are flagged as overflow places. In the authoring tool, the window name (or the subnet name) located on each window’s title bar copies the source transition’s name and has an additional unique number after the name. In the figure, for example, “help(page3)” indicates the *help* place in subnet 3—the “page3” value does not contain the source transition’s name for simplicity. By clicking list items below the “Overflow places:” label, the user can move to the corresponding subnet and the marking to check the overflow place. The net is not safe since safeness requires all places to have at most one token.

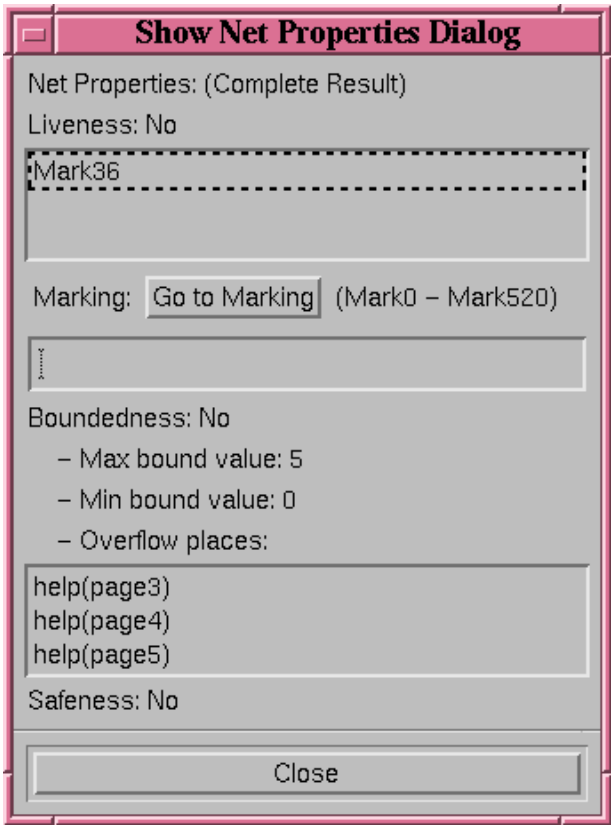


Figure 19: Net properties

If the author wants to trace a RT, he can use the Mark List dialog (see Figure 20). In the dialog, the author can trace how the dead marking occurs. In the figure, the author searches under *"Dead Mark"*, finds a dead marking, *Mark36*, and from there can trace back to the marking that leads to the dead marking, *Mark53*. In the figure, "*M53: (hide_help@P3->M36)*" means that when the *hide_help* transition in subnet 3 is fired from *Mark53*, the current marking (i.e., *Mark53*) moves to *Mark36*. In addition, the author can search under *"->M36"* to find the other markings that lead to *Mark36*. Here, *Mark23*, *Mark48*, *Mark53*, and *Mark55* lead to the dead marking. Moreover, the author can check a sequence of transitions by tracing the mark lists in the Mark List dialog. For example, the author can check the existence of the following transition sequences: Calc_currentTime(subnet1), Calc_accessRight(subnet2), and On_campus(subnet2).

Another analysis tool, the debugging tool, allows interactive analysis of the net while the author simulates the nets. To start a debugging session (debugging or recording mode), the author selects the *Record* icon. Then the author clicks over the enabled transition for simulation. In the debugging mode, every transition fire is recorded. If a dead link or an overflow place occurs while the user fires transitions, a warning message window pops up. Additionally, the author can move backward/forward from the current marking using the *Backward/Forward* icons. After finishing the traversal, the author can replay the traversal path using the *Play* icon. This icon plays the saved markings from an initial marking to an end marking with a specified time interval. The author can stop the playing at any time using the *Stop* icon to check the net status. The *Clear* icon is used for clearing the current stored sequence of markings. The *Move to First* and *Move to Last*

icons are used for moving the current marking pointer to the first and last net markings in the sequence, respectively.

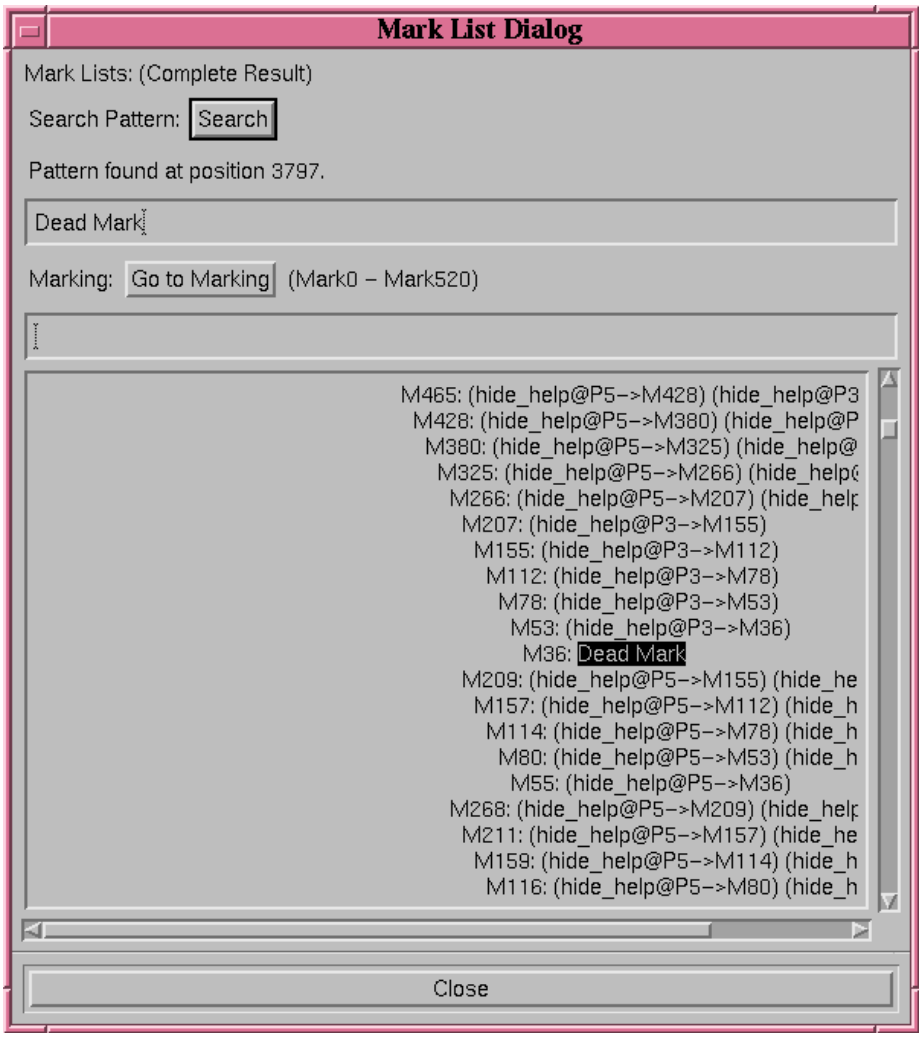


Figure 20: Mark lists

CHAPTER V

EXAMPLE APPLICATIONS

Several example applications have been built by using caT in different domains. In the following sections, some of these applications are introduced in order to show the usability of the caT specification and system, and also, if applicable, to provide a comparison to Trellis specifications. The first one is a context-aware application, called a personal document path, which provides contextualized information to the reader based on his (or her) current environmental information. The personal path example demonstrates another potential context-aware application that can be described in caT, in addition to the gallery tour example introduced in the previous chapters. The second application is a formal specification of various software requirements. In the software engineering field, Petri nets are one of the mechanisms that have been examined for their usefulness in providing formal specifications for software requirements [Ghezzi et al. 1991]. In addition, hypertext has been suggested as an important component of systems that support the software engineering process [Bigelow 1988]. Therefore, in Trellis, the requirements specification of describing the actions of a building's elevator was described [Furuta and Stotts 1994b]. In the second application, a new elevator protocol specification described in caT is introduced to show how efficiently caT can represent formal specifications as compared to Trellis. The last example is a specification of CSCW protocol specifications. Since Petri nets inherently provide a good environment for the synchronization of parallel actions, the caT model is also useful for expressing

group interactions within a document structure. In Trellis, a conference protocol was specified for directing a meeting among several participants [Furuta and Stotts 1994a]. In this example, we restructure the conference protocol by using caT's flexible net description features to show the power and flexibility of the caT specifications.

5.1 Context-aware applications

Context-aware applications provide an opportunity for customizing the provision of information in a way that reflects the user's current needs. In this research, we introduce a new context-aware hypertext model that supports the development of context-aware applications. In caT, we can specify what kinds of hypertextual information will be shown when the user enters certain contexts. The adaptive net behaviors of the caT model are especially capable of allowing contextualized information delivery in a dynamically-changing environment.

In this section, a context-aware application called a personal document path is introduced in order to illustrate a potential context-aware application that could be described in caT. The personal document path allows a reader to identify a favorite collection of Web documents and associate them with contextual information. When browsing using a Web browser, it provides contextualized information to the reader based on his (or her) current environment.

The personal path example considers the following simple scenario of personal space. When the reader first accesses the initial page of a personal path during office

hours, he (or she) will be asked if he (or she) is busy or not (this context may possibly be inferred by using external devices that sense the surrounding environment of the current user). If not busy, he (or she) will see a brief path of Web pages such as the news, weather, technical information, the local Web server usage report page, etc., before seeing a research-related search engine page; otherwise, the research-related page will be displayed directly. When the reader accesses the path during non-office hours, the favorite general purpose Web search engine page will appear after a brief path of Web pages.

Figure 21 shows the output model with *start_tour:#1* and *daily_tour:#2* subnets on top, and the other subnets iconized. In the *start_tour:#1* subnet, a token initiates from the *Begin* place and receives current access time. Based on this access time, the token moves either to the *office_hour* or *not_office_hour* transition. The *office_hour* transition has a condition statement, "*r.w.currentTime >= 9.0 && r.w.currentTime <= 17.00"*; and the *not_office_hour* transition has "*r.w.currentTime < 9.0 || r.w.currentTime > 17.00"* as its condition statement. Therefore, when the reader accesses the net during regular office hours, *office_hour* will be enabled. Otherwise, *not_office_hour* is enabled. Up to this point, transitions are automatically fired by the system without the user's interaction, since *time* attributes of the transitions are defined as "(0,0)".

When the token passes through the *office_hour* transition, a query document for the reader's feedback is generated. The template file attached to the *Template* node in the *start_tour:#1* net is as follows:

```
<Template multiple_frame="no">
<Place name="Question" link_display="none">
<Transition name="busy" activelink_name="(a) busy day">
<Transition name="not_busy" activelink_name="(b) not busy day">
```

The first two lines specify that the output page will be frameless, and output links attached to the *Question* place will not automatically be displayed after the content of the *Question* place. Instead, <*Transition*> constructs (the last two lines) are used for specifying the active link names for the output links. The output Web document from the current net status is shown in Figure 22.

Since the *busy* and *not_busy* transitions in the *start_tour:#1* net have a time value of "(0,∞)", the system will wait for the reader's input (i.e., a mouse click). When one of these transitions is fired, the reader's feedback information (i.e., schedule) is then stored within the current token, since the output arcs of each transition have specific assignment expressions (the output arc of the *busy* transition has an expression "*r.w.schedule = 'busy'*"; the output arc of *not_busy* has "*r.w.schedule = 'not_busy'*").

After receiving the reader's input, the token moves to the *StartTraverse* place. The updated token value is used for matching with the conditional statements attached to subsequent transitions. On the other hand, when the token passes through the *not_office_hour* transition, it directly reaches the *StartTraverse* place without the query document generation. The output arc of the *not_office_hour* transition has an expression "*r.w.schedule = 'not_busy'*".

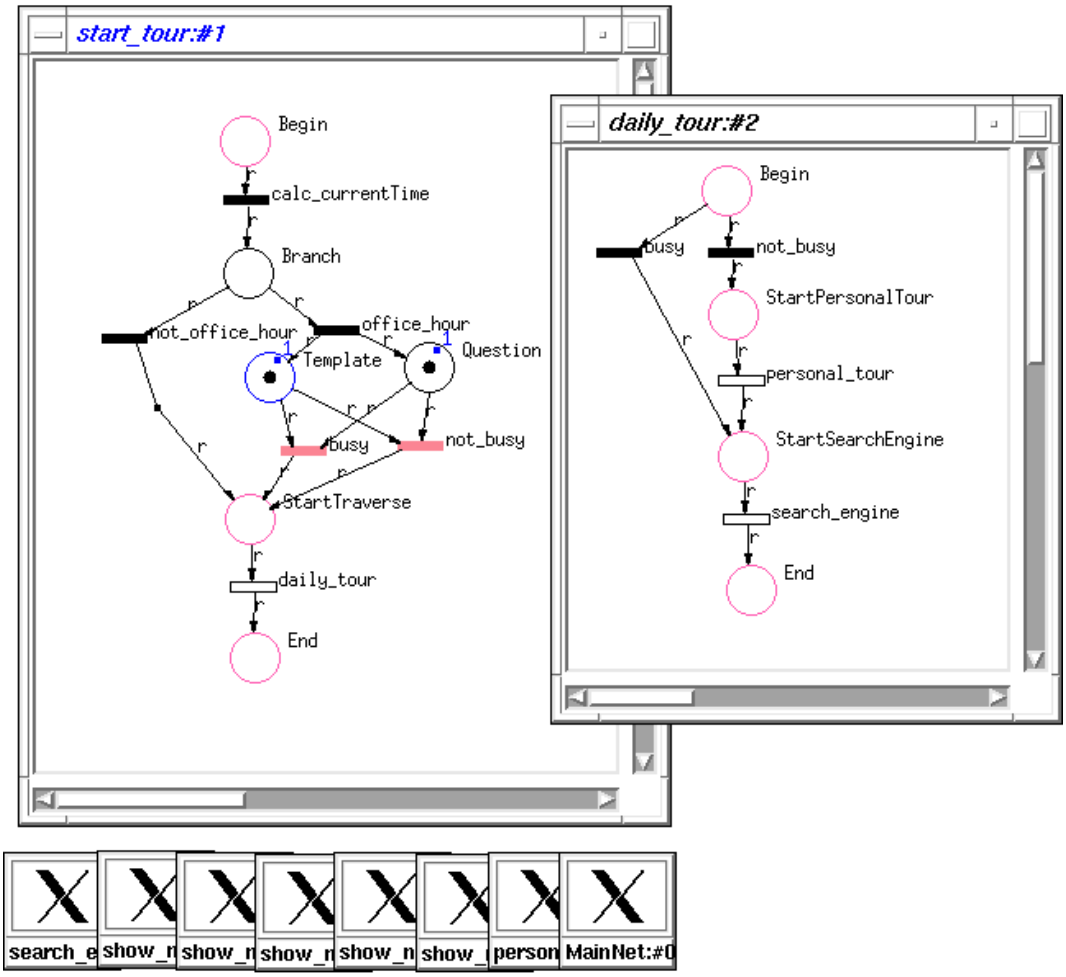


Figure 21: start_tour net: nets when accessed during office hours

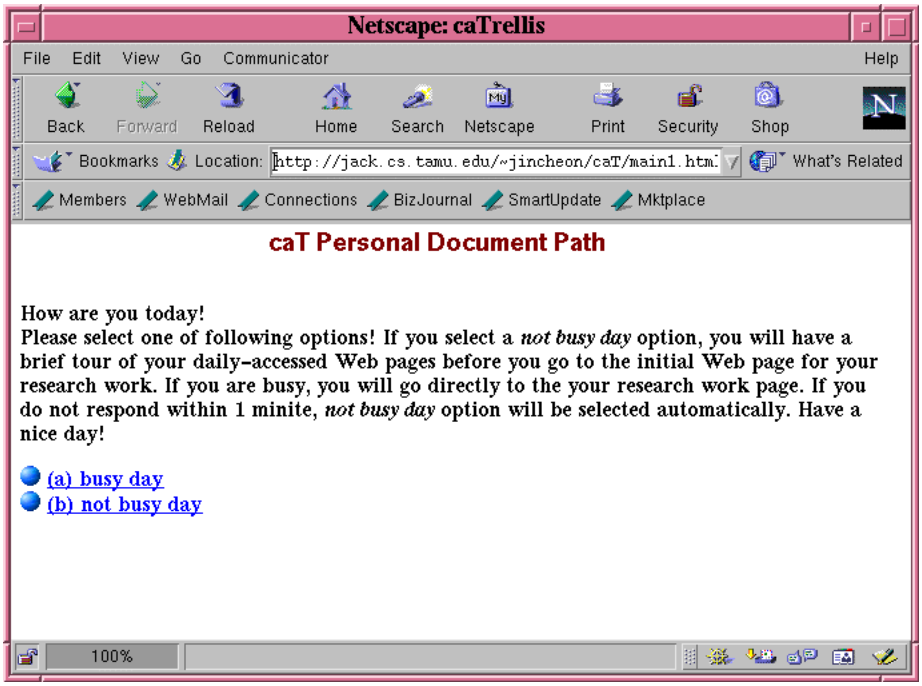


Figure 22: start_tour net: Web browser-based display

In the *daily_tour* substitution (or subnet) transition, *StartTraverse* and *End* in *start_tour:#1* are mapped to *Begin* and *End* in the *daily_tour:#2* subnet, respectively. Therefore, when a token arrives at *StartTraverse*, *Begin* in *daily_tour:#2* receives an identical token value. In the *daily_tour:#2* subnet, the *busy* transition has a condition statement "*r.w.schedule == 'busy'*"; the *not_busy* transition has the statement "*r.w.schedule == 'not_busy'*". Thus, the token moves to the *StartPersonalTour* place before it goes to *StartSearchEngine*, if the reader is classified as "not busy"; otherwise, it moves directly to the *StartSearchEngine* place. Therefore, a busy reader can skip the

brief tour of the Web pages (i.e., the *personal_tour* subnet), and instead view the search engine page directly.

Figure 23 shows the subnets and their marking status when the reader browses the third node of the *personal_tour:#3* subnet with annotation information (only related subnets are displayed). In the *personal_tour:#3* subnet, all five transitions (*show_node1, show_node2, show_node3, show_node4,* and *show_node5*) expand to the same structure by using the same subnet specification. However, each subnet (or substitution) transition passes different display information, such as a different file name (or Web URL) of the *Content* place, to each subnet instance.

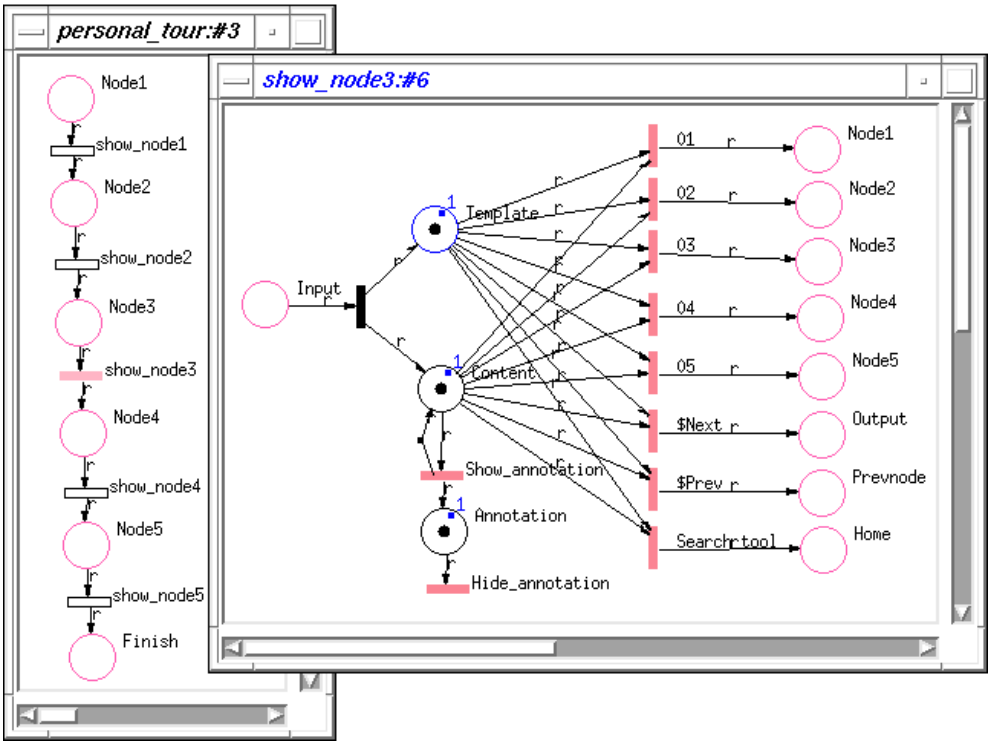


Figure 23: personal_tour net: nets when show_node3 is accessed

In Figure 23, the nodes in *personal_tour:#3* have input/output mappings with nodes in child nets. For example, in the *show_node3* transition of *personal_tour:#3*, *Node3* is mapped to both *Input* and *Node3* in the *show_node3:#6* net; *Node4* is mapped to both *Output* and *Node4*; additionally, the other nodes have mappings that allow access of one subnet to the other subnets through a parent net.

The template file for the *show_node3:#6* net is as follows:

```
<Template multiple_frame="yes_2frame">
<Place name="Annotation" link_display="none">
<Place name="Content" link_display="none">
```

According to this template specification, the output page will have two frames. Output links from the *Annotation* and *Content* places will not be displayed in the main frame, and the links on the left control frame will instead be used for path browsing. This system replicates the transition-associated links found on the left control frame. If normal Web links are followed, changing the information display from that generated by caT, the links on the left control frame will allow the user to continue to traverse the net specification. Figure 24 shows an output page that corresponds to Figure 23. Currently, the reader sees the third page with the annotation after clicking on the *Show_annotation* link, and could hide the annotation by clicking on the *Hide_annotation* link. In the current specification, a click on the *Show_annotation* (or *Hide_annotation*) link would only affect its current subnet. Thus, to provide a persistent view (i.e., a view with or without annotation information) for all subnets, the token status of the *Annotation* place

in the current subnet may be shared with related subnets by using the input/output mapping mechanism—i.e., specifying the *Annotation* place as a global place within the related subnets.

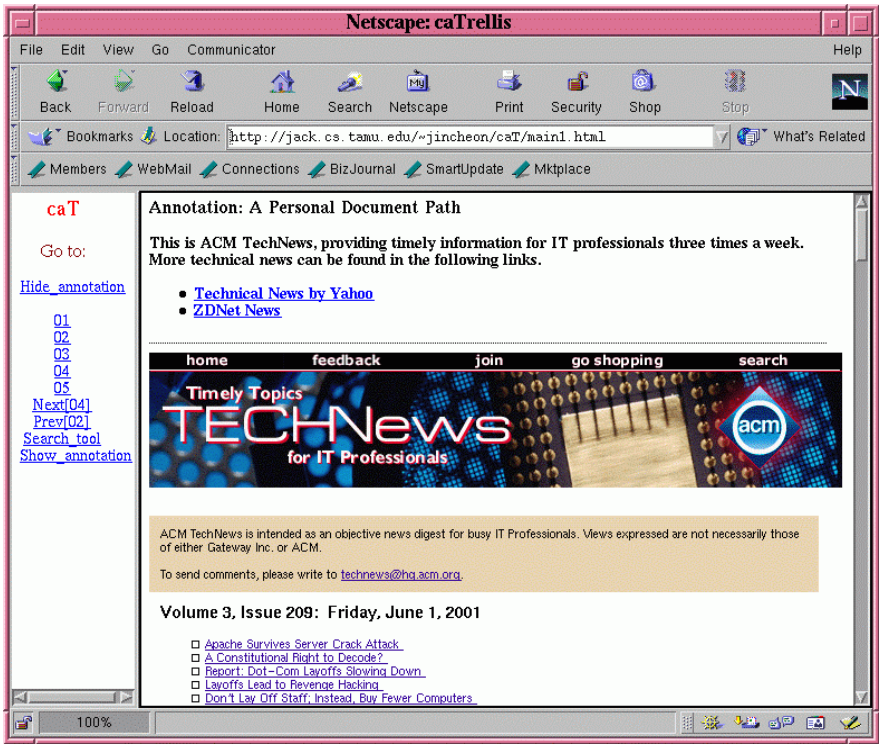


Figure 24: personal_tour net: Web browser-based display

Figure 25 shows a subnet and its marking status when a reader browses the *search_engine:#9* subnet during office hours. A reader browsing during office hours would see a research-related search page; otherwise a general Web search engine page would be displayed. In this subnet, the *office_hour* transition has a condition statement

that is satisfied only when the token has an access time within an office hour range; the *not_office_hour* transition is enabled when accessed outside of an office hour range.

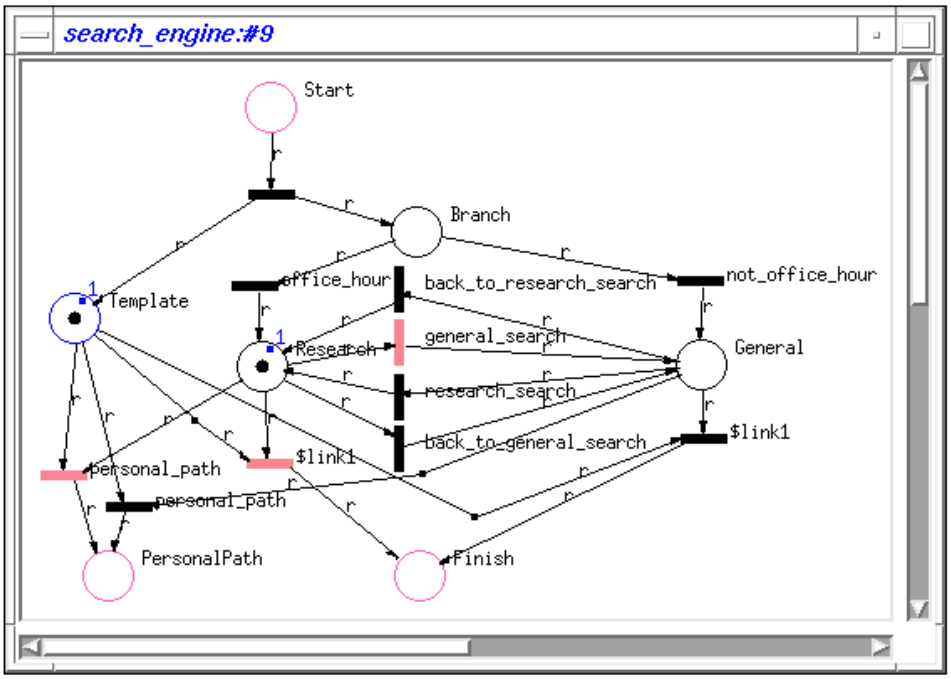


Figure 25: search_engine net: net when accessed during office hours

The template file for the *search_engine:#9* net is the same as that used in the *show_node3:#6* subnet, except those used for place names (i.e., *Research* and *General*) in the <Place> constructs. Figure 26 shows an output page that corresponds to Figure 25. A reader seeing a research search page could move to a general search page or back to a personal path page. Also, after moving to the general search page, the reader could return to the original research search page.

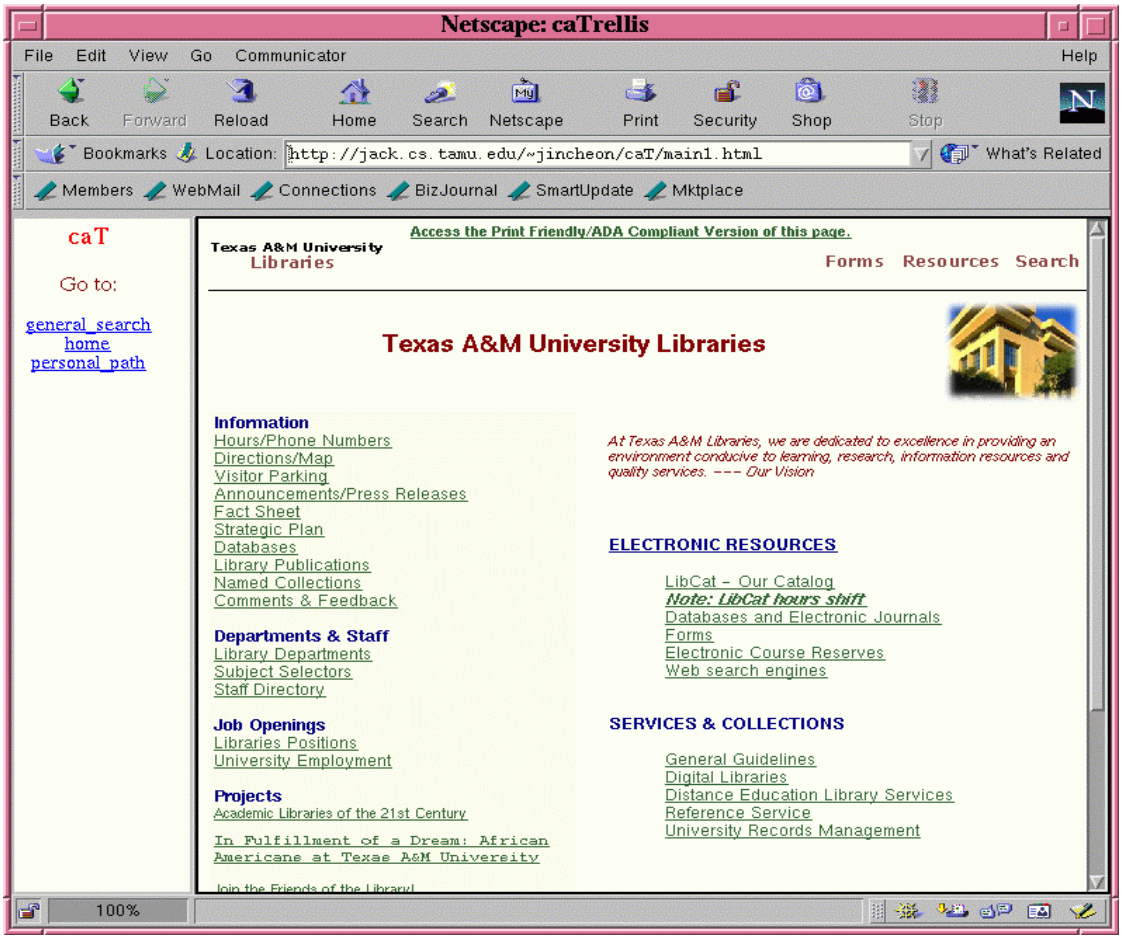


Figure 26: search_engine net: Web browser-based display

This personal document path example shows how the caT system supports both the authoring and the browsing of contextualized Web documents. The author can describe various dynamic documents by using the caT system, and the reader can browse dynamic documents by using the Web browsers. In this example, context information such as access time is provided by the system, and context information such as the

reader's current schedule is acquired from the reader's input or inferred from other contextual information.

Other kinds of environmental aspects that it might make sense to include relate to the context. For example, particular changes to the context of the Web pages would trigger (or refresh) their display—e.g., severe weather warnings, etc. Eventually, the addition of external sensors that perceive the current reader's contextual information would make context-aware applications described in caT both more interesting and powerful.

5.2 Software requirement specifications

In Trellis, a requirements specification describing an elevator protocol was discussed [Furuta and Stotts 1994b]. It illustrated the duality of document specifications and software requirements specifications. The concepts of rapid prototyping and incremental development, supported in Trellis, are useful in the contexts of both requirements specifications and dynamic document writing. After the individual specification is defined in Trellis, the reader can execute (or simulate) the output model and see their interested hypertextual view of the simulation process by using different browsers.

However, in Trellis, even though only one elevator is considered, the specification of the net suffers from graphical complexity due to the fact that it is represented with single-level (i.e., non-hierarchical) specification and simple (or non-structured) tokens. The single-level specification of the net requires that related components be multiply repeated in the description.

In this section, a new elevator protocol specification is introduced to show how compactly caT can represent the formal specifications of software requirements, especially as compared to Trellis. caT's elevator specification can be simplified because modular subnets, which separate distinct functions into separate descriptions, can be specified and interrelated, and also because information about the elevator's state can be encoded into the structured token.

In this example, we represent elevators for a three-floor building. Each elevator has three buttons inside of it to indicate to which floor it must travel. There are also buttons on the wall of each of the floors to call the elevator. There are two buttons on floor two that direct the elevator either upward or downward, and one on floors one and three to call the elevator for an upward or downward trip, respectively.

Figure 27 shows a caT elevator specification that consists of three subnets. *Floor3-2* and *Floor2-1* transitions in the main net, *MainNet:#0*, are associated with the *Floor3-2:#2* and *Floor2-1:#1* subnets, respectively. In the *Floor3-2:#2* subnet, the subnet's *Floor3_close* place is mapped to *Floor3_close* in *MainNet:#0,* its *Floor2_close* place to *Floor2_close,* and its *Floor_Buttons* to *Floor_Buttons*; the *Floor2-1:#1* subnet has similar mappings. Note that the *Floor_Buttons* place in each subnet shares the same token through the input/output mappings. In this way, the same tokens in the main net are shared with its subnets—similar to a global variable. In *MainNet:#0,* the token in the *Floor_Buttons* place stores the status of the button pressed such as *#3Down, #2Up, #2Down,* and *#1Up*, and the token information provided is used in other subnets. In

actuality, *#3Down*, *#2Up*, *#2Down*, and *#1Up* are defined as attributes of the token in the *Floor_Buttons* place, and receive the value "on" when the corresponding transitions (i.e., *#3Down*, *#2Up*, *#2Down*, and *#1Up* transitions in *MainNet:#0*) are invoked.

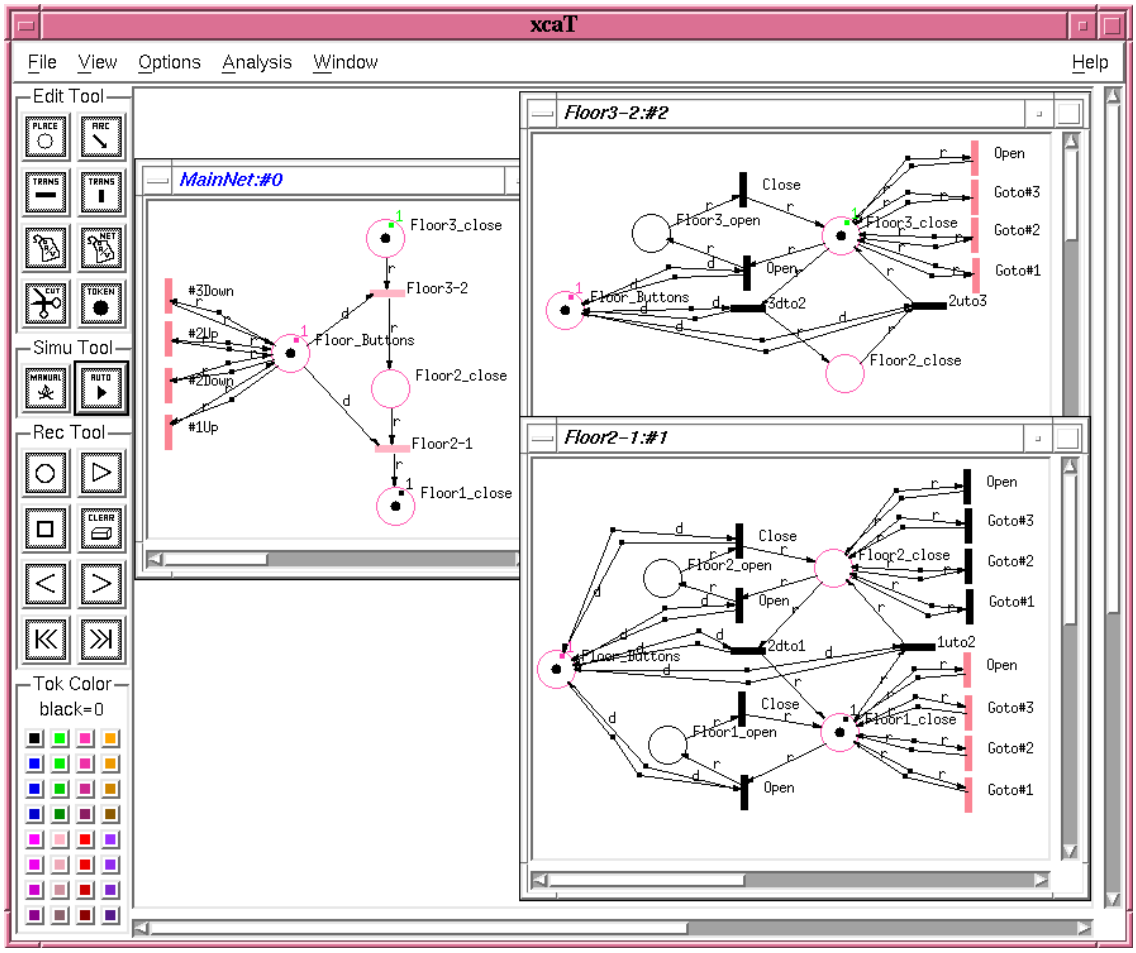


Figure 27: Elevator specification

In caT, the colored token is discernible, so each differently colored token can be used for different purposes. In *MainNet:#0,* the *maroon* token in the *Floor_Buttons* place represents the status of the buttons on the wall; the *black* token in *Floor1_close* and the *green* token in *Floor3_close* represent the first and second elevators, respectively. The tokens for the elevator store the status of the button pressed such as *open, goto#3, goto#2,* and *goto#1*. Actually, *open, floor3, floor2,* and *floor1* are defined as attributes of the elevator token and receive the value "on" when the corresponding transitions (i.e., *open, goto#3, goto#2,* and *goto#1* transitions in both *Floor3-2:#2* and *Floor2-1:#1*) are invoked.

In addition, conditional statements are specified for controlling token (elevator tokens) flows. For example, the *2uto3* transition in *Floor3-2:#2* has the condition statement "*r.w.floor3 == 'on' || d.w.#3Down == 'on'*". In this statement, the color variable *r* is instantiated to a token in *Floor2_close*, and *d* is instantiated to a token in *Floor_Buttons*. This statement indicates that when the *goto#3* button inside its elevator or the *#3Down* button on floor three is pressed, the tokens in the *Floor2_close* place can move through the *2uto3* transition.

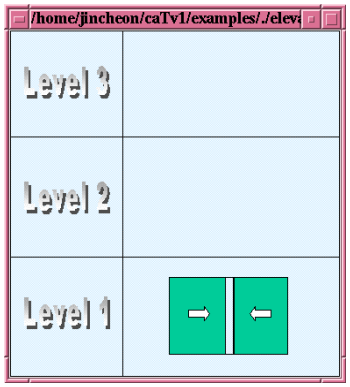
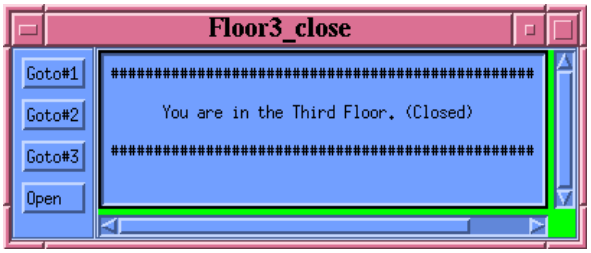
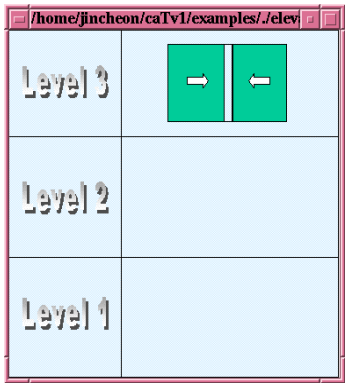
Expression statements associated with output arcs are specified to update the status of button presses in local token variables. For example, the output arc of the *#3Down* transition in *MainNet:#0* has an expression statement of *"r.w.#3Down = 'on'"*. On the other hand, one of output arcs of the *Open* transition in *Floor3-2:#2* has an expression statement of *"d.w.#3Down = 'off'"*. In this way, these expressions set/unset the status of

the buttons presses. In addition, timing attribute values attached to the transitions cause the simulation to react dynamically to the rider's action (i.e., the button selection).

Figure 28 displays the presentations of two browsers, the standard χTrellis browser (`xtb`) and an image browser, when the elevator specification is simulated. In caT, each place may contain several different types of contents for selective presentation, such as text, image, and sound. The image browser displays only the image contents of the active places.

Figure 28(a) shows a window that presents the available buttons on the wall of each of the three floors; the buttons are grouped together for convenience. Figure 28(a) is the presentation of `xtb` for the token color *maroon*. In the browser, only those places with tokens of a given color can generate a new window. Figure 28(b) shows the status of the two elevators—the first elevator (the *black* token) idle on the first floor, and the second (the *green* token) idle on the third. The windows in the "Text Presentation" column show the text-based presentations of `xtb`. The selection of links corresponds to the pushing of buttons inside an elevator. Following the links (i.e., the execution of the net) produces a display that corresponds to the states of operation in the elevator. On the other hand, the windows in the "Image Presentation" column show the graphic presentations of the image browser. In the image browser, the available links appear in a separate control window. The elevator protocol specification example shows how separate browsers could provide different views of a document's state which, when viewed together, would increase the understanding of the document's content.

(a) Presentation of floor buttons

	Text Presentation	Image Presentation
Elevator 1 (idle on first floor)		
Elevator 2 (idle on third floor)		

(b) Presentation of two elevators

Figure 28: Executing the elevator specification

In the elevator protocol specification, we can add another elevator using a differently colored token. However, if we are required to add another floor, we need another subnet similar to the *Floor3-2:#2* subnet. In addition, the current subnets including the token variables in the *Floor_Buttons* place, need some modifications to take into account the added floor.

Since caT implementations interpret their own specifications, caT (and Trellis) provides an environment for the rapid prototyping and incremental development of various protocols. In addition, the analysis tool which is integrated with the authoring tool, helps to investigate the output specification by identifying conditions such as dead links (i.e., deadlock in the software context). To explore the potential usability of caT, it will be interesting to build a variety of further software processes (or protocols) in the caT model. There is an on-line process support tool called the “process handbook” [Malone et al. 1999] that helps people to redesign existing organizational processes, invent new organizational process, and share ideas about organizational practices in general. It may be interesting to build some of these interesting processes in caT.

5.3 CSCW specification

When read by multiple readers simultaneously, the behavior of caT documents can be specified to range between shared and separate. Distributed readers can be specified to be unaware of each other, sharing the document’s specification but with an independent view of the document. Alternately, one reader’s actions may be specified to affect

another reader's view, thus allowing the sharing of documents in cooperative environments. This section discusses documents shared by multiple users.

In an early Trellis paper [Furuta and Stotts 1994a], a conference protocol is specified for directing a meeting of several participants. A moderator leads this meeting, and its participants can be in either listening or speaking modes. In the listening mode, a participant can either take the floor for speaking, or leave the meeting. The moderator and each participant are represented by different colored tokens. This paper also illustrates the flexibility with which the specifications can be changed in order to implement new policies.

In caT, we restructure the conference protocol by using caT's flexible net description features, including the hierarchical net specification, in order to show the power and flexibility of the caT specifications. We extend this meeting protocol to implement new policies such as setting the maximum times a listener can get the floor, as well as providing for the possibility of a listener with a high privilege to swap places with the moderator, and a participant who releases the floor to receive a lower priority for the next taking of the floor. In addition, we add a voting function to the protocol—after the agenda is fully discussed, participants can be asked to vote for topics on the agenda. The protocol is designed to be displayed on a Web browser. In this section, only the extended features are discussed; the details of the basic conference protocol can be found in [Furuta and Stotts 1994a].

This specification consists of three subnets: *MainNet:#0*, *getFloor:#1*, and *vote_subnet:#2*. The *MainNet:#0* net (see Figure 29) specifies most of the conference protocol features and associates some of its transitions with *getFloor:#1* (see Figure 30) and *vote_subnet:#2* (see Figure 31 and Figure 32) for specifying floor-taking and voting actions, respectively. In the *MainNet:#0* net, there is one token (*magenta1*) in the *moderate* place and one token (*orange1*) in *listen*. We can interpret from the current marking that there is one moderator and one listener. With the current marking, the moderator can delete or suspend the current listener, or begin voting. He also can swap the role of moderator with the current listener. In the *getFloor:#1* subnet, there is one token in the *listen* place, and the *getFloor* and *finish* transitions are enabled. With the current marking, the listener can take the floor or leave the current conference session. To restrict a listener from taking a floor more than 5 times, the *getFloor* transition has the conditional statement *"d.w.numvisit <= 5"*, and one of the output arcs of *getFloor* has an assignment expression *"d.w.numvisit = d.w.numvisit + 1"* to increase the number of visits. In the net, when a listener takes the floor, other listeners who had the floor previously move to the *listen_pool* place, thereby letting new listeners who have not yet had the floor have a higher priority for the next opportunity. In addition, a listener just returning from speaking remains in the *wait* place, thereby letting other listeners have a higher priority for the floor. The time attribute is used to specify the length of each wait. In addition, it may be possible for a listener to take the floor in any situation, if no one else wants it. For this purpose, we may need another transition that is the same as

getFloor, but does not have a conditional statement and has a certain time delay for allowing the current *getFloor* transition to first be applied.

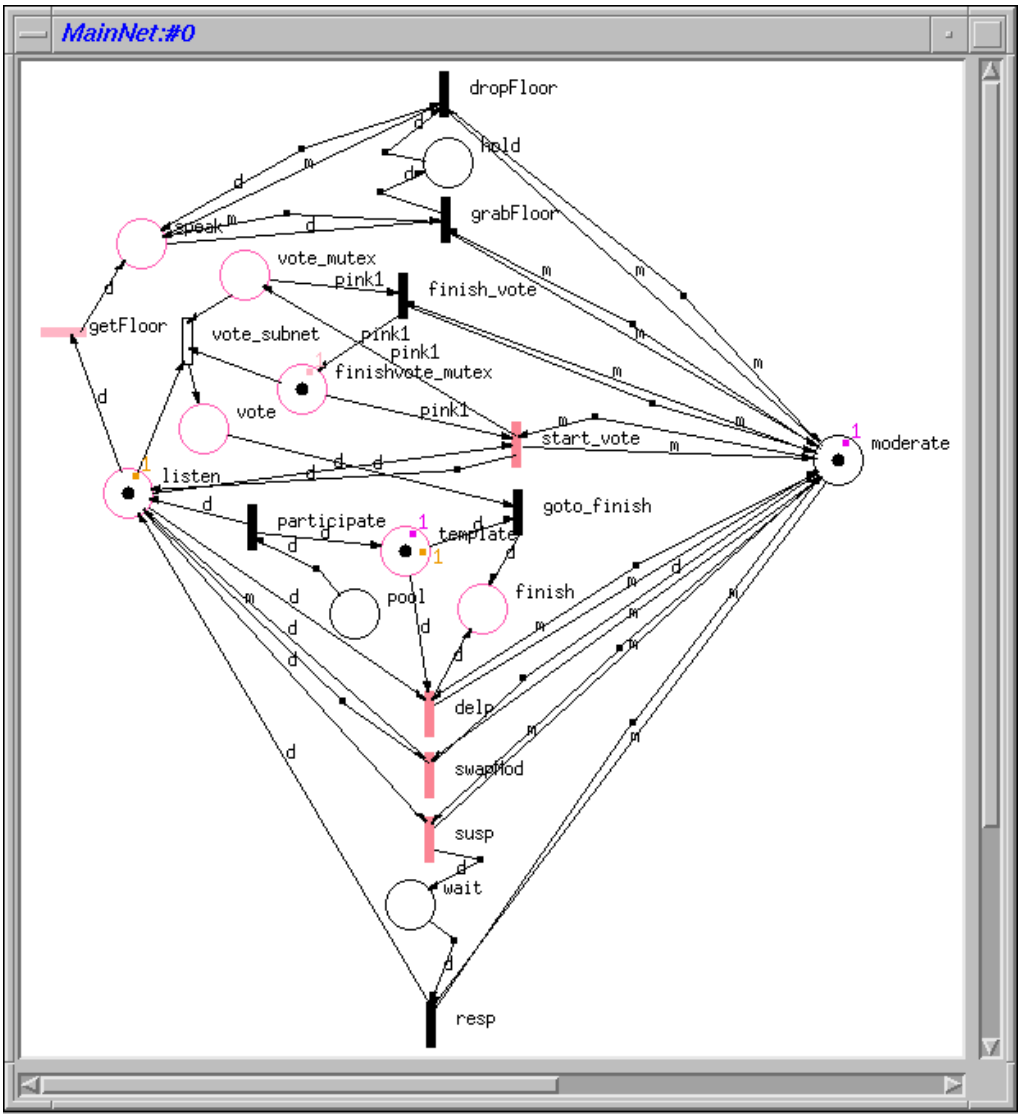


Figure 29: Conference protocol: MainNet net

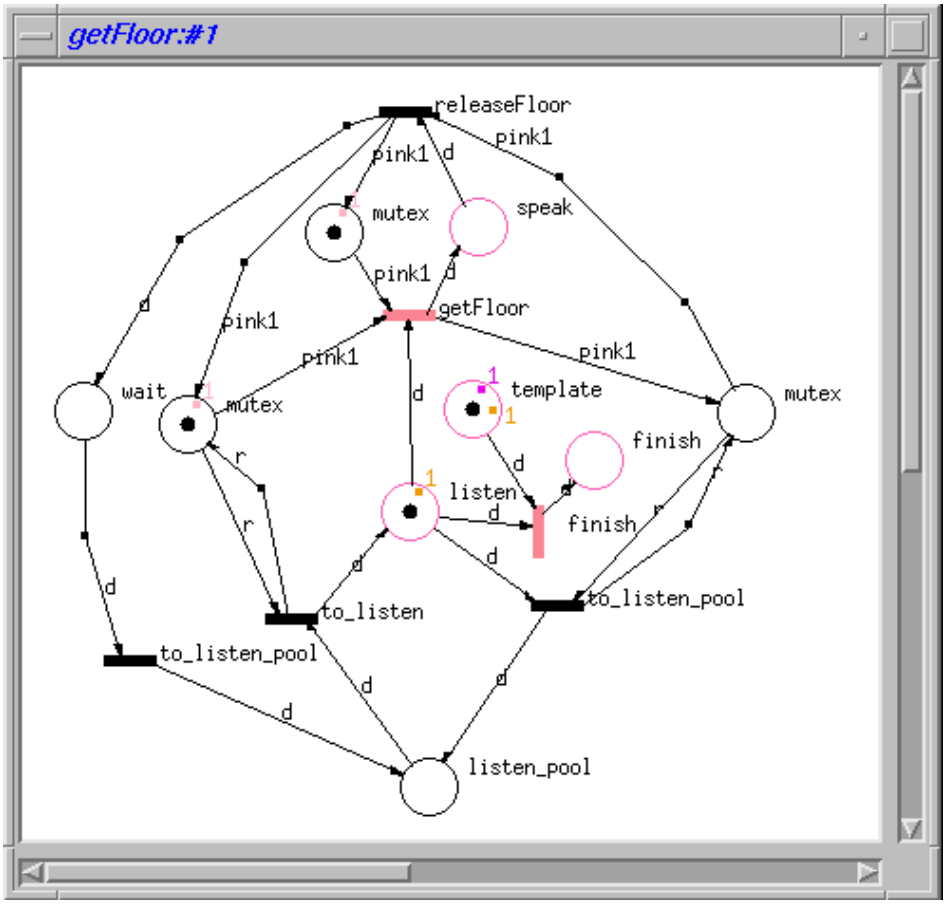


Figure 30: Conference protocol: getFloor net

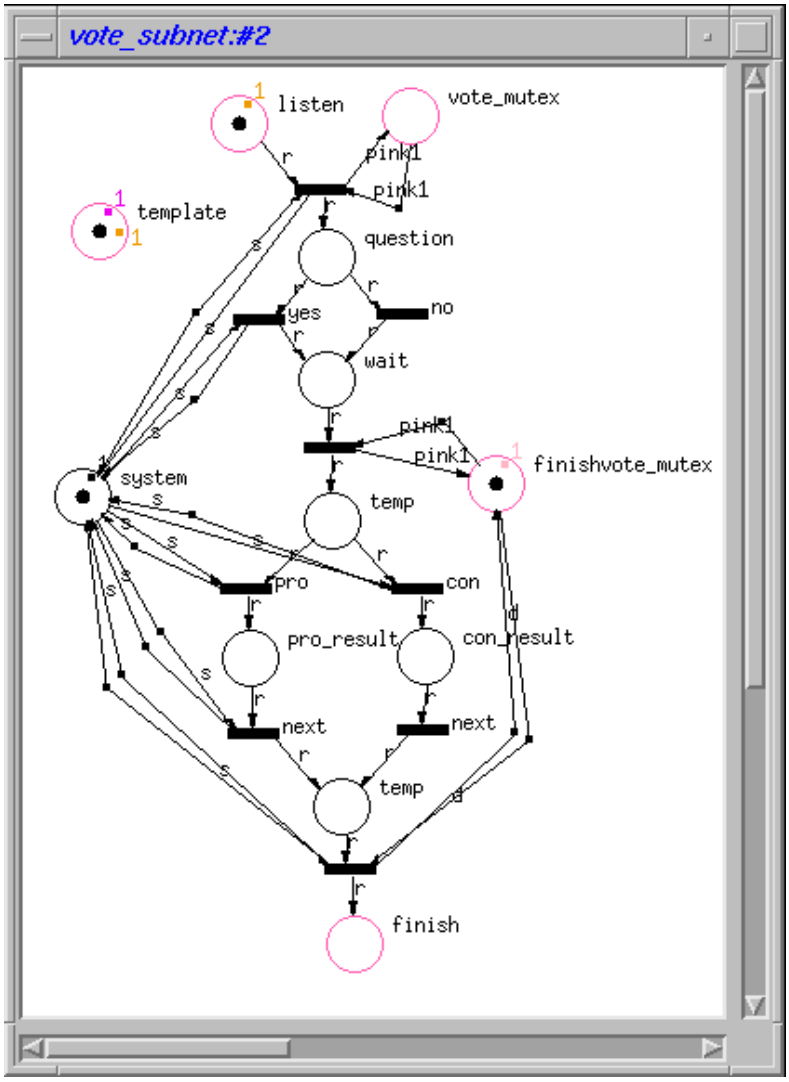


Figure 31: vote_subnet net: listening state

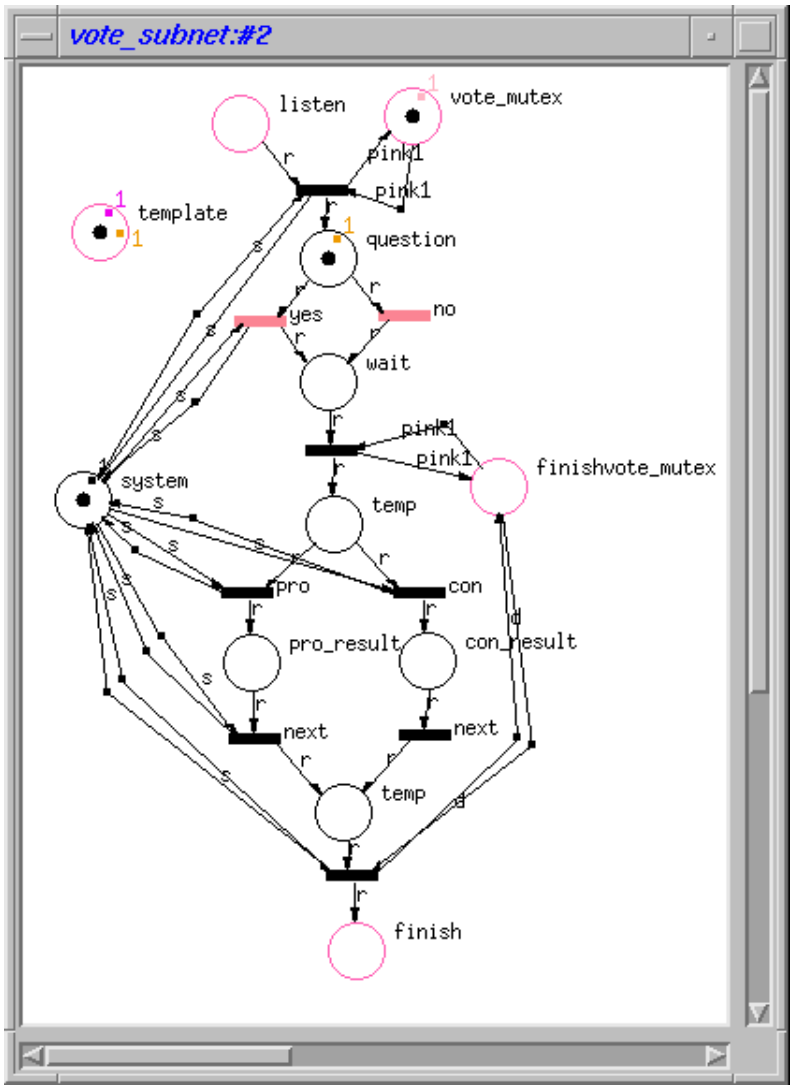


Figure 32: vote_subnet net: voting state

Figure 31 represents the *vote_subnet:#3* subnet and its marking status when a participant is in the listening mode, and Figure 32 represents the subnet and its marking status when a moderator forces all participants to the voting mode. A general description of the subnet is as follows. When the moderator clicks the *start_vote* link in *MainNet:#0* (see Figure 29) in his (or her) page, the *vote_mutex* place (in both *MainNet:#0* and *vote_subnet:#3*) receives a token, and all tokens in the *listen* place move to the *question* place (see Figure 31 and Figure 32). After finishing the vote (clicking the *yes* or *no* links), the participant waits in the *wait* place until the moderator clicks on the *finish_vote* link in *MainNet:#0* in order to put a token in *finishvote_mutex*. Then, the participant can view the results of the vote and move back to the listening mode or leave the meeting.

For example, the condition statement for the *pro* transition is as follows: *s.w.numpro > (2/3 * s.w.numparticipant).* This statement enables the *pro* transition when two thirds of the current participants vote “Yes”. The token in the *system* place stores the number of participants and the number of pros and cons using its output arc expressions.

The template file for the *vote_subnet:#2* is as follows:

```
<Template multiple_frame=“yes_3frame”>
<Place name=“question” >
<Place name=“wait”>
<Place name=“pro_result”>
<Place name=“con_result”>
```

According to this template specification, the output page will have three frames. Note that the *listen* place is not specified because it is instead specified in the template

file of the *getFloor:#1* subnet. Figure 33 and Figure 34 show the output result pages for a listener in the net state, shown in Figure 31 and Figure 32, respectively. Since Petri nets provide an inherently good environment for the synchronization of parallel actions, it will be interesting to develop further cooperative documents in the caT model.

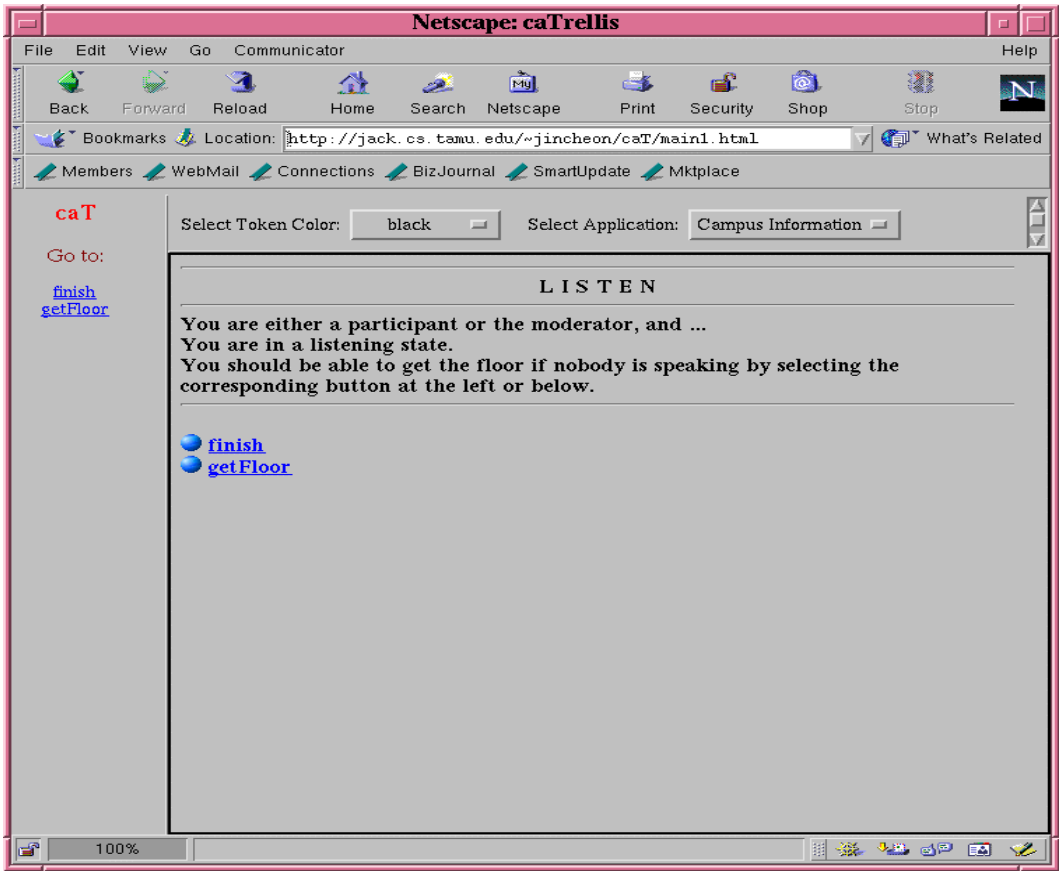


Figure 33: Output page in the listening state

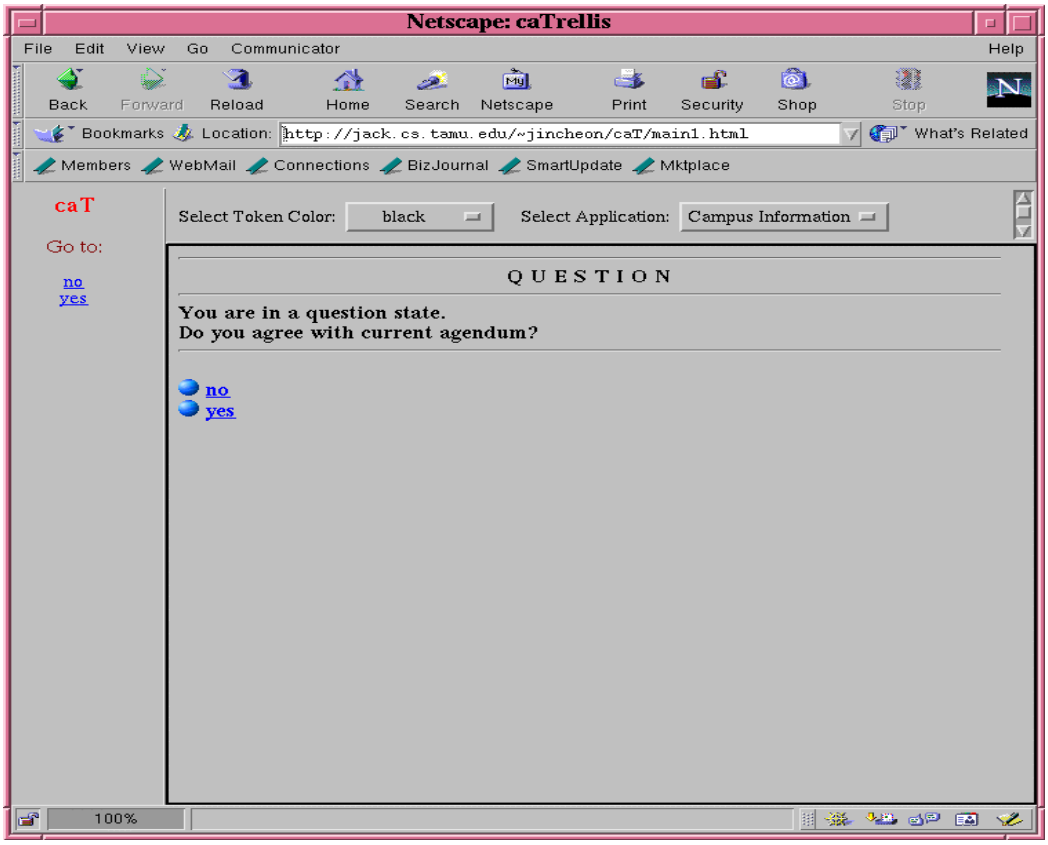


Figure 34: Output page in the voting state

CHAPTER VI

EVALUATION AND DISCUSSION

To acquire information about the usability of the prototype system and to guide the direction of future related research, a system evaluation was performed. The evaluation approach used for this prototype system is a general-usability analysis rather than a statistically significant study. The main target objects for this evaluation were a hypertext document authoring tool and a browsing tool that allowed for Web-based browsing of the resulting documents.

In this chapter, the objectives and methods of the evaluation, as well as the subjects who participated in this evaluation process, are described first. An analysis and discussion of the resulting data that was collected from the subjects follows. Lastly, our evaluation of the specification mechanism is detailed. The general procedures and methods used for this evaluation partly follow the experimental methodology described in Nielsen [1993] and Shneiderman [1992].

6.1 Evaluation of the caT system

6.1.1 Objectives

This evaluation took the form of a usability study with the goal of gaining information regarding current research and that of identifying future research opportunities in the area of hypertext document authoring and browsing with a Petri net-based hypertext

system, i.e., caT. The usability evaluation experiments investigated whether or not subjects could effectively use the authoring tool for creating hypertext documents, and could browse the resulting documents using the browsing tool.

6.1.2 Subjects

Since human subjects were involved in the evaluation of this prototype software system, the evaluation experiment required that three documents (the IRB Application, the Protocol Format for Use of Human Subjects in Research, and the Informed Consent Document) be submitted to the Institutional Review Board (IRB) at Texas A&M University—these three documents can be found in Appendices A, B, and C. After the approval from the IRB, the human subjects of this evaluation experiment were recruited via E-mail or person-to-person contact.

Five subjects were recruited to evaluate the caT system. All of them were graduate students (four doctoral and one master's student) at Texas A&M University. All subjects were in the Department of Computer Science and worked with the Center for the Study of Digital Libraries (CSDL). The subjects who had at least some experience with hypertext or Web document development were selected because they could best compare the caT system to other hypertext authoring systems or techniques. However, it was not easy to find subjects who had some experience with Petri nets and their tools. Two of the subjects knew about Petri nets before the evaluation and had some familiarity with Trellis concepts. However, none of the subjects had actually used Petri net tools. Profiles of the subjects who participated in this evaluation are shown in Table 13.

Table 13: Profiles of subjects

Characteristics	Subject				
	User 1	User 2	User 3	User 4	User 5
Major Field	Computer Science	Computer Science	Computer Science	Computer Science	Computer Science
Years in authoring Web documents or developing Web-based software	More than 2 years	More than 2 years	More than 2 years	More than 2 years	1-2 years
Knew Petri nets	No	Yes	No	Yes	No
Used Petri net tools	No	No	No	No	No
Could read Web-based customizable documents	Yes	Yes	No	No	Yes

6.1.3 Methods

This experiment conducted a usability evaluation of the prototype system. To understand how well the target users would be able to interact with the prototype system, the subjects were instructed to perform a sequence of pre-planned tasks with the software system. Then, in order to collect evaluation information, the subjects were required to answer questionnaires.

At the beginning of each evaluation, a brief overview of the system was demonstrated for each subject. In the demonstration, Web presentation and current analysis features were introduced by several applications described in the caT model. In addition, the subjects were introduced to Petri nets and had an opportunity to actually build high-level Petri nets using `xted`.

Next, each subject began the evaluation with a tutorial that was specially designed to give the subject step-by-step instructions for the caT system. Each subject actually used the system while following the tutorial. Several simple tasks were given for the system usability evaluation. The features covered in the tutorial were:

- An introduction to caT and high-level Petri nets
- The creation of hierarchical, colored timed Petri nets using `xted`
- The building of a caT model for a simple context-aware application, followed by Web browsing of the resulting model

Since the entire evaluation experiment was designed to take no more than two hours, and some features required background knowledge for the task, the following features were not evaluated by the subjects even though some of these were demonstrated at the beginning of the evaluation:

- Petri net analysis and debugging tools in `xted`
- Fuzzy rule base creation and interface
- User model profiles and the `xtb` browser

After completing the assigned tasks in the tutorial, the subjects were requested to answer the questionnaire shown in Appendix D. The first part of the questionnaire asked several questions about the subject. The second part of the questionnaire was comprised of evaluation questions about the caT system. Both qualitative and quantitative evaluations were included in the questionnaires.

6.1.4 Results analysis and discussion

The evaluation results collected from the subjects through the questionnaire are summarized in Table 14. In this table, the scale range for each item is between 1 and 9 with the value of 1 being *"Strongly Disagree"* and 9 being *"Strongly Agree."* The results indicate that the subjects were generally satisfied with the caT system even though the results were not statistically significant. Most of the subjects especially believed that caT could be useful for adaptive Web document publishing. They would use caT if it was available to them. However, they thought that caT needed further enhancements to provide an intuitive interface.

Table 14: Evaluation data

Evaluation Item		Subjects' Assessment
		Average
Interface Design	The tool provides an intuitive interface.	6.6
	Authors can easily and independently use the program.	7.4
	The program runs properly (bug-free).	7.6
Overall Evaluation	Rate your experience with the tool.	7.6
	The tool is useful for adaptive Web document publishing.	8
	The length of time to use the program is not excessive; learning justifies amount of time spent.	7.8
	Will you use the tool if the tool is available to you?	Yes (all subjects)

Before performing the evaluation, a number of usability improvements were made to χTed (caT's authoring tool), identified in part by informal reviews of the user interface.

For example, attribute names displayed to the user were modified to reflect their function, rather than their internal name. Fireable transitions in χTed's display were flagged by coloring them red (color mappings used in the authoring tool can be specified by the user; colors identified here are the default colors). With the addition of hierarchical nets, we also found it useful to graphically distinguish those transitions that represented subnets (with an open box) and further to indicate whether transitions in the subnet were active (the open box is colored pink when the subnet contains enabled transitions). Places mapped to inputs and outputs of subnets are also distinguished by coloring. We are considering the use of color to reflect the content type associated with places as well. In the present implementation of caT's authoring tool, places associated with template content (for the WWW client) are colored blue. Additionally, windows that contain active transitions are flagged by coloring those window titles red when the *Cascade Windows* menu item is invoked. This allows for easy finding of the currently active transitions among many subnets.

Common difficulties found during the evaluation and comments from subjects were mainly related to the user interface, and they are described as follows (the first and second items have been incorporated into a new version of the caT system, while the other items provide valuable perspective needed to enhance the current system):

- Attribute dialog (see Figure 15 in Chapter IV): Some of the subjects had difficulties with the attribute dialog. The most commonly recurring problem was that they did not press the "Add/Change" button before they closed the dialog

with the “OK” button. In that case, the current attribute value was not transferred to the information server. To solve the problem, a warning dialog was added. Currently, whenever the current attribute dialog is closed with the “OK” button, the current value in the Attribute *Value field* is compared to the one in the server. If the values are different, a warning dialog window pops up to inform the user that these values don’t match. Another confusion regarded the *Attribute List* and the *Attribute Option Menu* in the attribute dialog. The *Attribute List* shows the current object’s attribute variables stored in the server, and the *Attribute Option Menu* is a combo box (or control menu) that provides an easy way of adding a new attribute variable to the *Attribute Value* field. The meaning and usage of these features were not intuitive to some subjects. However, the confusion was resolved after the evaluation investigator explained their function to the subjects.

- Three mouse button operation: The system uses three mouse buttons, but the evaluation pointed out that there were some inconsistencies among the uses of these buttons. For example, in the attribute mode (when the *Label* icon is pressed, see Figure 14 in Chapter IV), the first mouse button is used for opening an attribute dialog window. However, in the token mode (when the *Token* icon is pressed), the third mouse button popped up an attribute dialog window for showing the token contents of the current place. Currently, a first button click on a place is defined to open either a place or a token attribute dialog, depending on the clicked location. The large dot region in the center of the place is defined to pop up a token attribute dialog window when there is a first mouse button click

on it. The region outside of the large dot but inside the place circle is defined to pop up a place attribute dialog window. As another example, the second mouse button was used in different modes for both object dragging and token removal. To simplify this, the second button is now reserved for object dragging and the third mouse button is instead used for token removal in the token mode. In addition, a *Move* icon has been added to the *Edit Tool* box to allow object dragging with the first mouse button, and the *Label Net* icon that was duplicated by a popup window submenu ("Net Attributes…") was deleted in order to make room for the addition of the *Move* icon. As a result, the user now usually uses the first button except for token deletion with the third button (object rearrangement with the second button is still available). Since the token deletion operation deletes one colored token at a time, we may need another operation for one-click deleting of all tokens in a particular place. A cut operation, provided by the *Scissors* icon, may be used for this purpose.

- Direct manipulation operation: In the current user interface, most of the object's attribute values are added/deleted/updated through the attribute window. However, for the specification of some attribute values, more direct user interface may help the user to efficiently use the system, such as specifying external files (template, content files, etc.) of places through a drag and drop interface and mapping input/output places of subnets by direct mouse clicks. In addition, a more direct way to clone places, transitions, and arcs will be useful in prohibiting the repeated opening and closing of attribute windows to copy and

paste an object's attribute values. Furthermore, in the authoring tool, drawing objects such as places and transitions are not defined as selectable objects—i.e., individual objects cannot be selected with a mouse click. They are drawn directly on the canvas with their location and size information. If the drawing objects could be made selectable, it would be easier to manipulate grouped objects such as move operations of grouped objects.

- Help function: The current authoring tool does not provide functionally useful help. Context-sensitive help and/or interactive help sessions with online examples would be helpful. Additionally, a library of example nets that could be easily modified to meet the author's design requirements would also be useful.

- Link display decision: For Web-based browsing of the caT model, a template file specifies how links are to be embedded. On the other hand, `xtb` has link display decisions enabled into the attribute/value pairs of arcs on the net. The *Hide Button* attribute is used when the author wants a link to be visible only from some of the places enabling the transition for a particular color. It is a little inconsistent to have different mechanisms for link display decisions in different browsers. Thus, these mechanisms should be unified to provide consistent control of link displays among multiple browsers. One easy solution for the current problem is that the *Hide Button* attribute is consistently used in all browsers for the link display control. In the Web-based browsing, in addition to the template file, the *Hide Button* attribute will be used.

- Miscellaneous functions: If many subnets are displayed in the authoring tool, it may be easy to get lost. Thus, a hierarchical summary viewing tool that shows a summarized hierarchical subnet tree may reduce disorientation and provide for the easy navigation of multiple subnets. As another issue, when in simulation mode, we can see the token movement around the nets. However, the tokens' local values are not visible. Thus, the visualization of token attribute values while the net is running will help the user to monitor the tokens' local value changes.

6.1.5 Evaluation of specification mechanism

The caT model provides a good abstraction and separation of the structure from the content of hypertext, using it helps to discipline the authoring activity by encouraging development in a structured fashion, requiring that the structure be designed before the actual contents are associated with the nodes of the structure. When the document structure is well organized from the initial phase, it will provide more manageability and expandability. On the other hand, in the Web, authors create fragments of content, including within them pointers to other pieces of content. In a sense, the caT specification mechanism is top-down while the Web mechanism is bottom-up.

As one of the evaluation tasks, subjects were asked to build a simple caT model (similar to Figure 25 in Chapter V) to evaluate the caT's specification mechanism. In the task, subjects who had prior knowledge of Petri nets performed the task a little better (i.e., with less help from the evaluation investigator) than those subjects who had not known Petri nets. The overhead for familiarizing themselves with Petri nets, especially

for those subjects who were new to the Petri nets concept, might cause this difference. It did not seem straightforward to design document structures in Petri nets, especially for first-time users (i.e., the subjects). It may also be true that, in the programming discipline, the structured design process requires some experience. The template node, i.e., a composite node, especially made the design process more complicated since the subjects had to consider token flow between the templates and among related places in order to achieve dynamic composition from the contents of the places. In addition, subjects learning the semantics and syntax of caT's additions to the basic Petri net's formalism, such as the transition's conditional and the arc's assignment statements, caused overhead. However, the work of associating the actual contents to the nodes in the structure was not difficult when compared to the authoring of document structures.

After the subjects were accustomed to the design process in caT's Petri nets, they could generate document structures and successfully finish the task. However, we expect that a certain amount of experience with using Petri nets will be necessary for authoring well-designed documents in caT. In addition, a more intuitive user interface for structured authoring will be necessary to reduce the complexity of the design processes. An easy way of creating/deleting subnets and a user-friendly navigation among subnets will be especially helpful.

CHAPTER VII

DISCUSSION AND CONCLUSION

In this chapter, future research opportunities for the caT system are discussed first, and the overall description of this research work and conclusions follow.

7.1 Discussion

7.1.1 Different client interpretations of content specification

The Trellis models and their prototype implementations, including caT, primarily focus on representation and distribution of hypertextual browsing semantics—another way to say this is that they specify the hypertext's structure and manage its traversal. The mapping of the content to the structure is managed in the prototypes by the clients. It is the role of the prototype to determine how this mapping is to take place. The clients determine both how to interpret the different data types (perhaps even choosing not to present the type to the reader) and also how to acquire the data (generally, by using the default action of looking in a known location in the file system).

The earlier Trellis project investigated the flexibility obtainable by adopting different client interpretations of content specification. For example, in a collaborative environment, it is possible that the participants may benefit from individual interpretations. One example of this would be different participants simultaneously viewing information in their own different native languages. Another might be

participants in remote locations cooperatively traversing a structure via network, but obtaining content from CD-ROMs that had been mailed to them previously rather than over the network (the bandwidth necessary to support control flow will be significantly less than that needed to transfer data).

The above scenarios become possible when the browsers have an option for how to acquire the data; i.e., file location information. For example, a local file system may be specified instead of the default file system in the server. In this case, the local file system may have a file directory structure that is similar to the one in the server. However, the local file system has different contents with the same named files. For example, a Korean user has a Korean version of files in his local file system, and a Chinese user has a Chinese version in his file system. The server may store an English version for the user who does not have his own favorite version in his local file system, or who is interested in the English version.

To support multiple readers who may use their own data in distributed environments client browsers need to be runnable on multiple platform environments. In the caT implementation, the current browsers, especially `xtb` and the image browsers, only run in the Linux environment. Thus, the distribution of client browsers is rather restricted. The current programs may be ported for multiple platforms. In this case, the managing of multiple versions for multiple platforms becomes burdensome, and system-dependent components such as X-Window system make it difficult to port some of the programs (such as χTed) to other Window systems. To resolve this problem, the current browsers

may need to be reimplemented in Java. Eventually, the server (i.e., the information server) may also need to be reimplemented in Java.

For different client interpretations of content specification, an interesting extension is to map multiple kinds of contents to an individual place. In the Trellis implementations, places only hold one level of content. The flexible specification of the relationships among different representations of a node's content (as to both type and location) will be useful to provide customized documents for each reader. We can think of the case of different browsers choosing to render a particular type in different ways. In addition, the modeling of relationships between device and data representation selection are investigations that should be relevant to future applications of caT. As the characteristics of the devices used will vary greatly, the flexibility gained from describing and managing these relationships should be especially applicable to context-aware applications.

In caT, to show the potential for the above features, each place is allowed to contain, for the selective presentations, several different types of contents, such as text, image, and sound. Thus, browsers for specific data types that display only their own specific type contents of the active places are necessary. Here, we can think of the following prospective browsers: image browser, text browser, and sound browser. In caT, an image browser that shows only the image contents of the active places has been developed. This image browser has been developed to provide a common framework that allows easy building of additional client browsers.

The current `xtb` is a kind of general browser, which shows various different kinds of data type contents with different viewers in multiple windows. It also allows invocation of an executable program such as an E-mail program. `xtb` assumes that each place has only one content element, and uses the following attributes: *Contents File Name* and *External Viewer*. The *Contents File Name* attribute specifies the name of the file that holds the contents of the place, and the *External Viewer* attribute specifies the name of the application called to render the file specified as contents (if null, it assumes the content is simple text).

The early Trellis implementations [Stotts and Furuta 1993] show how separate browsers (`xtb` and an image browser) can provide different views of a document's state, which, when viewed together, increase the understanding of the document's content. In caT, this approach is also taken in the elevator protocol specification example introduced in Chapter V. In comparison to the Trellis image browser, the caT image browser allows the user to browse the nets (i.e., fire transitions), including rendering images of active places.

In caT, the image browser uses an additional attribute for the image data specification, *Image File Name*. In Trellis, this is done only by the file name stored in the *Contents File Name* attribute, which is less flexible (the browser only sees a specific file extension, such as bit-mapped images). Thus, in caT, if we need to specify sound data, we may need an additional attribute such as *Sound File Name*. However, this approach is not general enough for incorporating many different types of contents. Thus,

it needs to be improved. One possible approach is having another attribute, *Secondary Content File Name*, which contains a list of content file names. Each browser will use implicit MIME (Multipurpose Internet Mail Extensions) type information for selecting one of files from both the *Content File Name* and the *Secondary Content File Name* attribute values. When multiple same-typed files are defined in a place, we may need to specify additional information (such as priority information among the same-typed files) for selecting one of the files.

7.1.2 The analysis tool

The current analysis tool is mainly focused on the analysis of a high-level Petri net itself. It needs to be extended to provide more useful analyses of hypertext document properties. In the context of Trellis, Stotts et al. [1992; 1998] explored the analysis of basic Petri nets for various browsing characteristics. In Trellis, the reader specifies dynamic document properties, such as “in some browsing path, Node X and Y must be concurrently visible”, with a branching temporal logic [Galton 1987]. The temporal logic reasons about the ordering of events in time, and a branching temporal logic supports special logic symbols that allow for the formulation of assertions involving relative ordering as well as the quantification over paths in a tree-like model of time. Trellis uses CTL* [Emerson and Halpern 1986], one of the general languages for branching time logics, as the basis for the hyperdocument notation called HTL* (Hypertext Temporal Logic).

The reader can analyze the document for the presence or absence of dynamic properties described in HTL (a restriction of HTL*) with a program verification technique called model checking [Clarke et al. 1986]. The HTL formulae are actually translated into equivalent CTL (a restriction of CTL*) formulae for applying an algorithm developed by Clarke. The coverability graph (i.e., a finite-state machine) generated from the Petri net is used as the input for Clarke's model checker. Taking a similar approach, Atlee and Gannon [1993] use a similar technique for the verification of software requirements specifications. They present a technique for transforming software requirements into state-based structures, which can then be analyzed using a state-based model checker.

However, the Trellis implementation does not provide an integrated environment for document property analysis, and there are several separate steps involved in completing the analysis. For example, the analysis and checking tools are separate programs, so the coverability graph, computed by the analysis tool, needs to be translated into the input format required by the model checker. In addition, the colored Petri net must be converted to a basic Petri net for model checking.

Therefore, for handy document property analysis, the Trellis implementation needs to be improved, especially concerning integrating individual steps together to improve quality and efficiency. Eventually, the authoring tool in caT may support the query of desired document properties in HTL, and it may present the analysis results within the tool. A translator that converts the desired document properties described in natural

language-like syntax to the temporal logic syntax may be necessary for easy specification of the desired document properties.

In the current caT implementation, the analysis tool has been integrated into the authoring tool. This allows interactive interfaces for the net analysis, while, in Trellis, the support of separate authoring and analysis tools allows batch mode interfaces. Converting Trellis's analysis method from a batch mode to an interactive mode raises many research (and implementation) issues. First of all, in the interactive mode, incremental analysis support will be required while the net is being developed. For example, the analysis tool may allow specifying analysis regions of the current nets (such as a subset of all subnets, a specific subnet, or some nodes within a subnet) for the analysis of partial nets. The partial net analysis feature will help the author to analyze only interested parts of nets before he (or she) finally performs the analysis of complete nets, which may take lengthy computation time. In addition, the current client-server architecture needs to consider slow response time of some interactive analysis transactions such as a RT construction. For instance, a client may wish to be interrupted when the analysis returns results rather than just waiting for the response. For improving the performance (i.e., time and size) of a RT construction, more complex analysis techniques such as a RT with equivalent classes and a RT with symmetries may be applied to the current analysis implementation [Jensen 1995].

7.1.3 Miscellaneous issues

In the authoring tool, i.e., χTed, the author can simulate Petri nets to test dynamic behavior of the nets. However, if the author wants to see resulting dynamic documents from the current net status, additional browsers such as `xtb` or a Web browser must be involved. Therefore, in the authoring tool, it would be convenient to have a document layout window that provides resulting document layouts (or at least resulting document file name lists). In this case, the author may not need to invoke additional clients to examine resulting documents. Furthermore, a simple version of the current χTed browser called Net Displayer may be implemented for providing a current document structure (i.e. a Petri net) view to the reader. The reader will mainly see resulting documents using regular browsers, and he (or she) will see the net status (such as marked places and active transitions) as additional information with Net Displayer.

caT supports a simple form of user modeling. Thus, it will be interesting to extend the current user modeling feature. For instance, to support educational hypertext applications, a reader's current knowledge values (e.g., known or not known) of the concepts in the current documents can be added (and updated) in the user profile. The system then provides different link annotations (e.g., different link colors) or link removal based on the current knowledge values. To support this feature, the system must support additional functions that set initial knowledge values for the current reader, and it must update the values based on the reader's traversal of the documents. In addition, the current net description statements (such as the conditional statements of transitions

and the assignment statements of arcs) may need to be extended for the link adaptation to the knowledge value information. Moreover, there will be many open, interesting problems related to automatic user modeling. For example, the system may infer the current user's knowledge value from his (or her) several browsing patterns, and it may set initial knowledge values for the current documents.

In caT, synchronization is rather coarse-grained, especially for multimedia presentation. For example, media can only be synchronized at their beginning or end. The maximum latency value associated with the *Time* attribute of each transition specifies the following condition: when all input tokens of a transition have stayed in their places for more than the specified maximum latency time, the transition is automatically fired. In fact, the maximum latency time indicates the maximum presentation time for the contents of the marked input places of the current transition. caT may need to support fine-grained synchronization (or presentation) controls. For example, when we have sound and video media objects and a text document as the contents of the current transition's input places, we may want to specify the following conditions: when either the sound or video object finishes presentation, finish all the other places' presentations, start the sound presentation five seconds after the video presentation starts, and start the video object from a specific clip point. It is possible that some of these scenarios may be described in the current caT model. However, since caT does not consider the delay of data transfer especially when the data are loaded from remote machines, it may be possible that it moves to the next state without presenting the sound and video objects at all. To support these features more obviously, the current

time-based Petri nets may need to be enhanced. In addition, browsers may need additional communications with the server for presentation controls, such as control of their presentation (e.g., start from a specific frame) and report of their presentation states (e.g., load, start, play, finish, etc.)—SMIL supports some of these features.

In caT, every caT model (i.e., a document application) is an instance of an information server (i.e., a Petri net engine). Clients, which are separate processes, communicate with the server via RPC. In the current caT implementation, clients poll the server to check net status changes. When there are a limited number of clients, this is acceptable. However, when there are many clients working together, this places a heavy load on the server. A different possible approach is broadcasting the net status changes from the server to the clients that ask for reports of net changes in advance (this was the mechanism in αTrellis). The server also breaks down with larger numbers of clients if it is point to point. Thus, it will be interesting to improve the current communication scheme using other approaches like multicasting [Stevens 1994]. Multicast-capable networks implement one-to-many (or many-to-one) communication at the physical level, using addressing schemes that allow a single transmitted message to be routed to many receivers.

In caT, there is no communication support among servers. When it is supported, each server located in each user's computer can communicate with other servers to support cooperative applications. For example, we can think of a scheduling tool in which each reader has his (or her) own scheduling server and the server communicates with other

servers to support cooperative scheduling works. First of all, to support this feature, the server must be easily portable to multiple platforms. In the current implementation, the colored tokens are used in cooperative document applications. With this extension (i.e., the communication support among servers), it will be interesting to compare the usability of two approaches. It will also be possible to use the server communication feature together with the colored tokens. For example, we may build large-scale cooperative applications, where each group has one server that communicates with other servers. As another example, each user in each server has a unique colored token for his (or her) identification, and he (or she) interacts with other participants in other servers using the unique colored token. For example, a user in a server sends, by firing a transition, his (or her) unique colored tokens to other participants (i.e., input places of transitions in other nets) to allow them to see his (or her) schedule data—in this case, the received tokens are used for access control.

7.2 Conclusions

In this research, we have developed an extended hypertext model called caT (for Context-Aware Trellis) in order to support flexible adaptation in a dynamically-changing environment. In summary, caT is an extension of the earlier Trellis model that is designed for supporting the authoring, browsing, and analysis of fairly complex, dynamic documents. It supports these functions in the following ways:

- Introduces a new context-aware hypertext model for the development of context-aware hypertext documents;

- Presents mechanisms for reducing complexity of specification representation in the high-level Petri net-based hypertext system;
- Provides a mechanism to create a dynamic composite node from the active content elements and to embed the link anchors for flexible Web-based information presentation of the caT model;
- Develops structured authoring and analysis tools for the high-level Petri net-based hypertext system; and
- Improves the Trellis implementation, especially aspects of the Trellis prototypes and their environment that restricted their general application.

In caT, the author who is accustomed to the caT model and its underlying formalism can specify various dynamic documents using the structured GUI (Graphical User Interface) authoring tool. Afterward, the reader who is not required to know the model can browse the dynamic documents using various browsers to have his selective document views in different environments. Especially, the Web-based browsing of the caT model allows the reader to define sophisticated hypertext applications using the caT model and to browse result information using the familiar user interface of WWW (World-Wide Web) browsers.

Unlike many hypertext systems, caT (and Trellis) specifications are centered around the structure of the hypertext rather than the nodes of the hypertext. This is particularly noticeable when comparing the process of caT specifications with the process used in authoring World-Wide Web collections. In χTed, authors initiate their design (i.e.,

document structures) by specifying a Petri net. In the Web, authors create fragments of content, including within them pointers to other pieces of content. In a sense, the χTed specification mechanism is top-down, while the Web mechanism is bottom-up. In another interpretation, χTed specifications are link-centric, while Web specifications are node-centric. As the number of authors using caT increases with the distribution of the caT system to potential authors, it will be interesting to see whether the differences in specification genres are reflected in the resulting hypertexts.

Potential target applications of the caT model are context-aware hypermedia applications running on mobile systems with attached sensors to capture their environment. In caT, several prototype examples have been developed to show its usability for specifying and presenting various dynamic documents: context-aware documents, executable formal specifications of software requirements, cooperative documents, and customized Web documents. The further investigation and development of more appealing documents that are described in caT will be interesting.

In caT, the extensions of the Trellis model, such as hierarchical nets, structured tokens, and conditional predicates, have been designed to be consistent with Petri net theory in order to apply analysis techniques to investigate the document specification. Consequently, we have built two kinds of analysis tools, which are integrated into the authoring tool. The first one is an interactive debugging tool that allows interactive analysis of the net while the author simulates the nets. The second one is an analysis tool with which the author can analyze the hierarchical Petri nets by building a Reachability

Tree. However, the current analysis tools are mainly focused on the analysis of a high-level Petri net itself. Further research work to support more useful analysis of hypertext document properties will be interesting.

Finally, to acquire information about the usability of the prototype system and to guide the direction of future related research, a system evaluation was performed. The successful result of the evaluation showed the potential usability of the model as a context-ware hypertext system. This research work will provide some background techniques and experiences for further context-aware research in the hypertext field. With the increased availability of computing devices equipped with environmental sensors, further research work will enhance the usability of the caT system.

REFERENCES

ABOWD, G. D., ATKESON, C. G., HONG, J., LONG, S., KOOPER, R., AND PINKERTON, M. 1997. Cyberguide: a mobile context-aware tour guide. *ACM Wireless Networks, 3*, 5 (Oct.), 421-433.

AKSCYN, R. M., McCRACKEN, D. L., AND YODER, E. A. 1988. KMS: a distributed hypermedia system for managing knowledge in organizations. *Communications of the ACM, 31*, 7 (July), 820-835.

ATLEE, J. M. AND GANNON, J. D. 1993. State-based model checking of event-driven system requirements. *IEEE Transactions on Software Engineering, 19*, 1 (Jan.), 24-40.

BENNETT, F., RICHARDSON, T., AND HARTER, A. 1994. Teleporting – making applications portable. In *Proceedings of the Workshop on Mobile Computing Systems and Applications* (Santa Cruz, CA, Dec.), 82-84.

BESLMULLER, E. 1988. Office modeling based on Petri nets. In *Proceedings of the 5th Annual ESPRIT Conference* (Brussels, Belgium), 977-987.

BIGELOW, J. 1988. Hypertext and CASE. *IEEE Software*, *5*, 2 (Mar.), 23-27.

BOTAFOGO, R. AND MOSSÉ, D. 1995. The MORENA model for hypermedia authoring and browsing. In *Proceedings of the International Conference on Multimedia Computing and Systems* (Los Alamitos, CA, May), 42-49.

BROWN, P. J., BOVEY, J. D., AND CHEN, X. 1997. Context-aware applications: from the laboratory to the marketplace. *IEEE Personal Communications, 4,* 5 (Oct.), 58-64.

BROWN, P. J. 1998a. Some lessons for location-aware applications. In *Proceedings of First Workshop on HCI for Mobile Devices* (Glasgow University, Scotland, UK, May), 58-63.

BROWN, P. J. 1998b. Triggering information by context. *Personal Technologies, 2,* 1 (Sept.), 1-9.

BRUSILOVSKY, P. 1996. Methods and techniques of adaptive hypermedia. *User Modeling and User Adapted Interaction, 6,* 2-3, 87-129.

BUSH, V. 1945. As we may think. *The Atlantic Monthly, 176,* 1 (July), 101-108.

BULTERMAN, D. C. A., HARDMAN, L., JANSEN, J., MULLENDER, K. S., AND RUTLEDGE, L. 1998. G*ri*NS: a graphical interface for creating and playing SMIL documents. In *Proceedings of the Seventh International World Wide Web Conference* (Brisbane, Australia, April), 519-529.

CAMPBELL, B. AND GOODMAN, J. M. 1988. HAM: a general purpose hypertext abstract machine. *Communications of the ACM, 31,* 7 (July), 856-861.

CAPPS, M., LADD, B., STOTTS, D., AND NYLAND, L. 1996. Educational applications of multi-client synchronization through improved web graph semantics. In *Proceedings of 5th Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises* (Stanford, CA, June), 21-26.

CARDOSO, J., VALETTE, R., AND DUBOIS, D. 1990. Petri nets with uncertain markings. In *Lecture Notes in Computer Science, Advances in Petri Nets 1990, 483,* G. Rozenberg, Ed., Springer-Verlag, Berlin, Germany, 64-78.

CARDOSO, J., VALETTE, R., AND DUBOIS, D. 1996. Fuzzy Petri nets: an overview. In *Proceedings of 13th IFAC World Congress* (San Francisco, CA, June 30 – July 5), 443-448.

CARDOSO, J. AND PRADIN-CHÉZALVIEL, B. 1997. Logic and fuzzy Petri nets. In *Proceedings of Workshop on Manufacturing and Petri Nets within International Conference on Applications and Theory of Petri Nets* (Toulouse, France, June), 17-34.

CHEN, S. M., KE, J. S., AND CHANG, J. F. 1990. Knowledge representation using fuzzy Petri nets. *IEEE Transactions on Knowledge and Data Engineering, 2,* 3 (Sept.), 311-319.

CHUNG, C.-C. 1997. subWeb: creating a guided path of concurrent and synchronized browsing streams on the World Wide Web. Master Report. Department of Computer Science, Texas A&M University, College Station, TX.

CLARKE, E. M., EMERSON, E. A., AND SISTLA, A. P. 1986. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems, 8,* 2 (April), 244-263.

CONKLIN, J. 1987. Hypertext: an introduction and survey. *IEEE Computer, 20,* 9 (Sept.), 17-41.

COOPER, K. 1995. $TSPN_{ui}$: a Petri net model for specifying user interactions in multimedia presentations. MSC Thesis. University of British Columbia, British Columbia, Canada.

DE BRA, P. D., EKLUND, J., AND KOBSA, A. 1999a. Adaptive hypermedia: purpose, methods, and techniques. In *Proceedings of the 10th ACM Conference on Hypertext and Hypermedia* (Darmstadt, Germany, Feb.), 199-200.

DE BRA, P. D., HOUBEN, G., AND WU, H. 1999b. AHAM: a Dexter-based reference model for adaptive hypermedia. In *Proceedings of the 10th ACM Conference on Hypertext and Hypermedia* (Darmstadt, Germany, Feb.), 147-156.

DELISLE, N. M. AND SCHWARTZ, M. D. 1986. Neptune: a hypertext system for CAD applications, In *Proceedings of ACM SIGMOD for International Conference on Management of Data* (Washington DC, May), 132-143.

DE OLIVEIRA, M. C. F., TURINE, M. A. S., AND MASIERO, P. C. 2001. A statechart-based model for hypermedia applications. *ACM Transactions on Information Systems, 19,* 1 (Jan.), 28-52.

DEY, A. K. AND ABOWD, G. D. 1999a. Toward a better understanding of context and context-awareness. Tech. Rep. GIT-GVU-99-22. Department of Computer Science, Georgia Institute of Technology, Atlanta, GA.

DEY, A. K., SALBER, D., ABOWD, G. D., AND FUTAKAWA, M. 1999b. The conference assistant: combining context-awareness with wearable computing. In *Proceedings of the Third International Symposium on Wearable Computers* (San Francisco, CA, Oct.), 21-28.

DIAZ, M. AND SENAC, P. 1993. Time stream Petri nets: a model for multimedia streams synchronization. In *Proceedings of the International Conference on Multimedia Modeling* (Singapore, Nov.), 257-273.

ELLIS, C. 1979. Information Control Nets: a mathematical model of office information flow. In *Proceedings of the 1979 ACM Conference on Simulation, Measurement and Modeling of Computer Systems*, (Boulder, CO, Aug.), 225-240.

ELLIS, C. AND NAFFAH, N. 1987. *Design of office information systems*. Springer-Verlag, New York, NY.

EMERSON, E. A. AND HALPERN, J. Y. 1986. "Sometimes" and "not never" revisited: on branching vs. linear time. *Journal of ACM, 33*, 1 (Jan.), 151-176.

ENGLEBART, D. C. 1984. Authorship provisions in Augment. In *Proceedings of the IEEE COMPCON* (San Francisco, CA, Feb. 27 - Mar. 1), 465-472.

FURUTA, R. AND STOTTS, D. 1989. Programming browsing semantics in Trellis. In *Proceedings of Hypertext '89* (Pittsburgh, PA, Nov.), 27-42.

FURUTA, R. AND STOTTS, D. 1994a. Interpreted collaboration protocols and their use in groupware prototyping. In *Proceedings of the Conference on Computer Supported Cooperative Work* (Chapel Hill, NC, Oct.), 121-131.

FURUTA, R. AND STOTTS, D. 1994b. A hypermedia basis for the specification, documentation, verification, and prototyping of concurrent protocols. Tech. Rep. TAMU-HRL 94-003. Department of Computer Science, Texas A&M University, College Station, TX.

FURUTA, R. AND STOTTS, D. 2001. *Trellis: a formally-defined hypertextual basis for integrating task and information*. Lawrence Erlbaum Associates, Mahwah, NJ.

GALTON, A. 1987. *Temporal logics and their applications*. Academic Press, New York, NY.

GENRICH, H. J. AND LAUTENBACH, K. 1981. System modeling with high-level Petri nets. *Theoretical Computer Science, 13,* 109-136.

GHEZZI, C., JAZAYERI, M., AND MANDRIOLI, D. 1991. *Fundamentals of software engineering*. Prentice Hall, Englewood Cliffs, NJ.

GUDWIN, R. AND GOMIDE, F. 1998. Object networks – a modeling tool. In *Proceedings of FUZZ-IEEE98, 1998 IEEE World Congress on Computational Intelligence* (Anchorage, AK, May), 77-82.

HALASZ, F. G. 1988. Reflections on NoteCards: seven issues for the next generation of hypermedia systems. *Communications of the ACM, 31,* 7 (July), 836-855.

HALASZ, F. G. 1991. 'Seven issues': revisited. Keynote for Hypertext '91. See transcript at http://www.parc.xerox.com/spl/projects/halasz-keynote.

HALASZ, F. AND SCHWARTZ, M. 1990. The Dexter reference model. In *Proceedings of NIST Hypertext Standardization Workshop* (Gaithersburg, MD, Jan.), 95-133.

HALASZ, F. AND SCHWARTZ, M. 1994. The Dexter hypertext reference model. *Communications of the ACM, 37,* 2 (Feb.), 30-39.

HAREL, D. 1987. Statecharts: a visual formalism for complex systems. *Science of Computer Programming, 8,* 3 (June), 231-274.

HODES, T. D. AND KATZ, R. H. 1999. Composable ad hoc location-based services for heterogeneous mobile clients. *ACM Wireless Networks 5*, 5 (Oct.), 411-427.

HOLT, A. W. 1988. Diplans: a new language for the study and implementation of coordination. *ACM Transactions on Office Information Systems, 6,* 2 (Jan.), 109-125.

ISO. 1997. Information technology – coding of multimedia and hypermedia information – part 1: MHEG object representation, base notation (ASN. 1). ISO 13522-1.

JENSEN, K. 1992. *Coloured Petri nets: basic concepts, analysis methods and practical use volume 1*. Springer-Verlag, New York, NY.

JENSEN, K. 1995. *Coloured Petri nets: basic concepts, analysis methods and practical use volume 2*. Springer-Verlag, New York, NY.

JOYCE, M. 1991. Storyspace as a hypertext system for writers and readers of varying ability. In *Proceeding of Hypertext '91* (San Antonio, TX, Dec.), 381-387.

KAY, J. 1995. The UM toolkit for cooperative user models. *User Models and User Adapted Interaction, 4,* 3, 149-196.

KENDALL, R. 1996. Hypertextual dynamics in A Life Set for Two. In *Proceedings of Hypertext '96* (Washington, DC, Mar.), 74-83.

KIM, T.-H., KIM, K.-I, AND LEE, K.-C. 2000. Simple and consistent SMIL authoring: no more structure editing and no more errors. In *Proceedings of Multimedia Computing on the World Wide Web 2000* (Seattle, WA, Sept.).

LADD, B., CAPPS, M., STOTTS, D., AND FURUTA, R. 1995. Multi-head/Multi-tail Mosaic: adding parallel automata semantics to the Web. In *Proceedings of the 4th WWW Conference* (Boston, MA, Dec.), 433-440.

LESSTIF.ORG. 2001. The LessTif homepage. http://www.lesstif.org.

LITTLE, T. AND GHAFOOR, A. 1990. Synchronization and storage for multimedia objects. *IEEE Journal on Selected Areas in Communications 8*, 3 (Apr.), 413-427.

LOONEY, C. G. 1988. Fuzzy Petri nets for rule-based decisionmaking. *IEEE Transactions on Systems, Man, and Cybernetics*, *18*, 1 (Jan. - Feb.), 178-183.

MALONE, T. W., CROWSTON, K., LEE, J., PENTLAND, B., DELLAROCAS, C., WYNER, G., QUIMBY, J., OSBORNE, C., BERNSTEIN, A., HERMAN, G., KLEIN, M., AND O'DONNELL, E. 1999. Tools for inventing organizations: toward a handbook of organizational processes. *Management Science*, *45*, 3 (Mar.), 425-443.

MASIERO, P. C., OLIVEIRA, M. C. F., GERMANO, F. S. R., AND SANTOS, G. P. B. 1994. Authoring and searching in dynamically growing hypertext databases. *Hypermedia Journal*, *6*, 2, 124-148.

NA, J.-C. AND FURUTA, R. 2000a. Context-aware hypermedia in a dynamically-changing environment, supported by a high-level Petri net. In *Proceedings of Hypertext '2000* (San Antonio, TX, May), 222-223.

NA, J.-C. AND FURUTA, R. 2000b. Context-aware digital documents described in a high-level Petri-net-based hypermedia system. In *Proceedings of International Conference on Digital Documents and Electronic Publishing* (Munich, Germany, Sept.).

NA, J.-C. AND FURUTA, R. 2001. Dynamic documents: authoring, browsing, and analysis using a high-level Petri net-based hypermedia system. In *ACM Symposium on Document Engineering 2001* (Atlanta, GA, Nov.). To appear.

NIELSEN, J. 1993. *Usability engineering*. AP Professional, Cambridge, MA.

PAULO, F. B., TURINE, M. A. S., DE OLIVEIRA, M. C. F., AND MASIERO, P. C. 1998. XHMBS: a formal model to support hypermedia specification. In *Proceedings of Hypertext '98* (Pittsburgh, PA, June), 161-170.

PETERSON, J. L. 1981. *Petri net theory and the modeling of systems*. Prentice-Hall, Englewood Cliffs, NJ.

PRABHAKARAN, B. AND RAGHAVAN, S. V. 1993. Synchronization models for multimedia presentation with user interaction. In *Proceedings of ACM Multimedia '93* (Anaheim, CA, Aug.), 157-166.

REISIG, W. 1985. *Petri nets: an introduction*. Springer-Verlag, New York, NY.

RHODES, B. J. 1997. The wearable remembrance agent. In *Proceedings of 1st International Symposium on Wearable Computers* (Cambridge, MA, Oct.), 123-128.

SADLER, S. W. 1996. Motif multi-document interface version 1.0. ftp://ftp.x.org/contrib/widgets/motif/MDI-1.0.README.

SALBER, D., DEY A. K., AND ABOWD, G. D. 1999. The context toolkit: aiding the development of context-enabled applications. In *Proceedings of ACM SIGCHI Conference on Human Factors in Computing Systems* (Pittsburgh, PA, May), 434-441.

SCHILIT, W. N., ADAMS, N., AND WANT, R. 1994. Context-aware computing applications. In *Proceedings of the Workshop on Mobile Computing Systems and Applications* (Santa Cruz, CA, Dec.), 85-90.

SHIPMAN, F., FURUTA, R., BRENNER, D., CHUNG, C., AND HSIEH, H. 2000. Guided paths through web-based collections: design, experiences, and adaptations. *Journal of the American Society of Information Sciences 51,* 3 (Mar.), 260-272.

SHNEIDERMAN, B. 1992. *Designing the user interface: strategies for effective human-computer interaction, second edition*. Addison-Wesley, Reading, MA.

SIBERTIN-BLANC, C. 1985. High level Petri nets with data structures. In *Proceedings of 6th European Workshop on Application and Theory of Petri Nets* (Espoo, Finland, June), 141-170.

SMIL. 2001. Synchronized Multimedia Integration Language (SMIL 2.0) specification, W3C proposed recommendation (05 June 2001). http://www.w3.org/TR/smil20/.

STEVENS, W. R. 1994. *TCP/IP illustrated, volume1: the protocols*. Addison-Wesley Publishing Company, Reading, MA.

STOTTS, P. D. AND FURUTA, R. 1989a. Petri-net-based hypertext: document structure with browsing semantics. *ACM Transactions on Information Systems, 7,* 1 (Jan.), 3-29.

STOTTS, P. D. AND FURUTA, R. 1989b. Alphα: an authoring language for Petri-net-based hypertext. In *Proceedings of the Hypertext II Conference* (University of York, UK, June).

STOTTS, P. D. AND FURUTA, R. 1989c. αTrellis: a system for writing and browsing Petri-net-based hypertext. In *Proceedings of the Tenth International Conference on Application and Theory of Petri Nets* (Bonn, Germany, June), 312-328.

STOTTS, P. D. AND FURUTA, R. 1990. Hierarchy, composition, scripting languages, and translators for structured hypertext. In *Proceedings of Hypertext '90,* (Inria, France, Nov.), 180-193.

STOTTS, P. D. AND FURUTA, R. 1991. Dynamic adaptation of hypertext structure. In *Proceedings of Hypertext '91* (San Antonio, TX, Dec.), 219-231.

STOTTS, D. AND FURUTA, R. 1993. The Trellis project: process modeling for CSCW. Tech. Rep. TAMU-HRL 93-005. Department of Computer Science, Texas A&M University, College Station, TX.

STOTTS, P. D., FURUTA, R., AND RUIZ, J. C. 1992. Hyperdocuments as automata: trace-based browsing property verification. In *Proceedings of the 1992 European Conference on Hypertext* (Milan, Italy, Nov. 30 - Dec. 4), 272-281.

STOTTS, P. D., FURUTA, R., AND CABARRUS, C. R. 1998. Hyperdocuments as automata: verification of trace-based browsing properties by model checking. *ACM Transaction on Information Systems, 16,* 1 (Jan.), 1-30.

SUCHMAN, L. A. 1987. *Plans and situated actions*. Cambridge University Press, New York, NY.

SUN MICROSYSTEMS, INC. (Palo Alto, CA) 2001. Java homepage. http://java.sun.com/.

THE MATHWORKS, INC. (Natick, MA) 1999. Fuzzy logic toolbox user's guide. The Mathworks Home Page, http://www.mathworks.com/.

TOMPA, F. W. 1989. A data model for flexible hypertext database systems. *ACM Transaction on Information Systems, 7,* 1 (Jan.), 85-100.

TURINE, M. A. S. AND DE OLIVEIRA, M. C. F. 1997. A navigation-oriented hypertext model based on statecharts. In *Proceedings of Hypertext '97* (Southampton, UK, April), 102-111.

VAN BILJON, W. R. 1988. Extending Petri nets for specifying man-machine dialogues. *International Journal of Man-Machine Studies, 28,* 4, 437-455.

WANG, H. K. AND WU, J.-L. C. 1995. Interactive hypermedia applications: a model and its implementation. *Software-Practice and Experience, 25,* 9 (Sept.), 1045-1063.

WANT, R., HOPPER, A., FALCAO, V., AND GIBBONS, J. 1992. The active badge location system. *ACM Transactions on Information Systems, 10,* 1 (Jan.), 91-102.

WANT, R., SCHILIT, B., ADAMS, N., GOLD, R., PETERSON, K., ELLIS, J., GOLDBERG, D., AND WEISER, M. 1995. The PARCTAB ubiquitous computing experiment. Tech. Rep. CSL-95-1. Xerox Palo Alto Research Center, Palo Alto, CA.

WILLRICH, R., SÉNAC, P., DIAZ, M., AND DE SAQUI-SANNES, P. 1996. A formal framework for the specification, analysis and generation of standardized hypermedia documents. In *Proceedings of Multimedia '96* (Boston, MA, Nov.), 399-406.

WOO, M., QAZI, N., AND GHAFOOR, A. 1994. A synchronization framework for communication of pre-orchestrated multimedia information over broadband networks. *IEEE Network, 8,* 1 (Jan.-Feb.), 52-61.

YEN, J. AND LANGARI, R. 1999. *Fuzzy logic: intelligence, control, and information.* Prentice-Hall, Upper Saddle River, NJ.

ZADEH, L. A. 1989. Knowledge representation in fuzzy logic. *IEEE Transaction on Knowledge and Data Engineering, 1,* 1 (March), 89-100.

ZHENG, Y. AND PONG, M.-C. 1992. Using statecharts to model hypertext. In *Proceedings of Hypertext '92* (Milan, Italy, Nov.), 242-250.

ZURAWSKI, R. AND ZHOU, M. 1994. Petri nets and industrial applications: a tutorial. *IEEE Transactions on Industrial Electronics, 41,* 6 (Dec.), 567-583.

APPENDIX A

IRB APPLICATION

Revised: May 1, 2000

TAMU #_____
(for IRB use only)

Texas A&M University
IRB Application
Protocol for Human Subjects in Research

PART 1: Summary Cover Sheet ✓ **If Requesting Exempt Status Check Here**(See Page 2)
Please check off or provide details on the following (enter N/A if not applicable)

Principal Investigator Name Na, Jin-Cheon Faculty/Staff Graduate Student ✓
Department: Engineering / CPSC Mail Stop 3112 Phone 845-4924
Email: jincheon@cs.tamu.edu Fax 847-8578
Name of Graduate Committee Chair/Research Advisor if Graduate Student Richard Furuta
Department Engineering / CPSC Mail Stop 3112 Phone 845-3839
Project Title Hypertext Authoring Tools: A Usability Evaluation

Funding Agency None
Objective Estimate of Risk to Subject: ✓ None _ Low _ Moderate _ High
Existing Documents____ Existing Specimens ____
Gender of subjects: _ Male _ Female Both ✓ Age(s): 20 - 50 Total Participants (est.): 10

Location of Research
Source of Subjects:
_ Subject Pool(_____)
✓ Other TAMU Students
__ Community
__ Prisons
__ School Teacher/Administrator
__ Other Please specify_____

Subject Recruitment:
✓ Direct Person-to person contact
_ Telephone Solicitation (Attach a script)
_ Newspaper Ad (Attach a copy)
_ Posted Notices (Attach a copy)
_ Letter (Attach a copy)
__Other (Please describe) _____

Compensation Yes__ No ✓ (Attach payment schedule)
Deception Yes___No✓ (Attach debriefing form if yes)

Research/Course Credit Yes _No ✓

Invasive or Sensitive Procedures: Yes ___ No ✓
_ Blood Samples
_ Physical Measurements (electrodes, etc.)
__Other (Specify_____
_ Urine Samples
_ Stress Exercise
_ Review of Medical/Pysch. Records
_ rDNA

Sensitive Subject Matter: Yes___ No✓
_ Alcohol, Drugs
_ Depression/Suicide
_ Learning Disability
_ Abortion, AIDS/HIV, Sex
__Psychological Inventory
__Other please specify_____

Use of Video__ or Audio tapes ___
If yes, answer the following:
Retained Yes _ No _
Retained/Length of Time _____
Destroy/Erase Yes _ No _
Other (explain) _____
Use specified in consent form? Yes _ No _
Designate who will use or have access to tapes: _____

Provisions for Confidentiality/Anonymity
___ Replies Coded
___ Secure Storage
○ Anonymous Response OR
⊙ Confidential Response
(Cannot be both anonymous & confidential)

Location Where Signed Consent Forms Will be Filed: ROOM 408D, Department of Computer Science, H. R. Bright Building
(Consent forms must be kept on file for 3 years after the completion of the project).*(It is best to keep the forms in a campus office in a locked file cabinet.)*
Do you have any relationship with any or all of the subjects, other than your investigator role? ____Yes ✓ No.
If "Yes," you must explain in the source of subjects section and explain how you will avoid any type of coercion.
(Doctor-patient, teacher-student, counselor-student, etc.)

Revised May 1, 2000

REQUEST FOR EXEMPTION **from full** IRB review

Some research projects involving human subjects are exempt from **full** review by the IRB. See the attached sheet on research categories exempt from full IRB review. (*Sensitive topics and subjects such as children or minors, pregnant women and prisoners are not considered for exempt research).*

Basis for Exemption [Please refer to attached "Categories Exempt From Full IRB Review."]
(Do not check unless requesting an exemption from full IRB review.)

_____ Established Educational Settings/Normal Educational Practices(a letter of approval from a school official must be obtained and submitted to the IRB before the study can be conducted)(children or minors are not exempt)

_____ Use of educational anonymous tests (cognitive, diagnostic, aptitude, advancement; **attach copy**).

__✓__ Survey or interview procedures, [**unless** identifying subjects places them at legal or personal risk, and unless survey or procedures deal with sensitive matters of personal behavior]

_____ Observations of public behavior [**unless** identifying subjects places them at legal or personal risk, and unless observations deal with sensitive matters of personal behavior]

_____ Anonymous collection or study of existing documents, records, pathological or diagnostic specimens which are without any identifiers or codes.

_____ Evaluation of agencies and programs for administrative purposes where there was no deviation from standard practice.

_____ Taste and food quality evaluation and consumer acceptance studies.

The U.S. population is becoming increasingly culturally, linguistically, economically, and ethnically diverse. The research needs to make a concerted effort to ensure that research subjects reflect the population demographically, including these groups who have been traditionally under represented. However, it is recognized that the available pool of subjects may preclude having a balanced population. If you cannot use a diverse population in your research, you must justify this action in Part II, A, 1.

NOTE: The IRB makes the final decision whether or not a proposal is exempt from full IRB review. Please check with the IRB Secretary (979-458-4067). Exempt proposals require one (1) copy of each instrument, i.e., Part A, Part B (with signatures), consent forms, research instrument, recruitment materials, etc. Full IRB review proposals require an original, with signatures, and 3 full copies, including research instrument, consent forms, recruitment materials, etc.

APPENDIX B

PROTOCOL FORMAT FOR USE OF HUMAN SUBJECTS IN RESEARCH

Protocol Format for Use of Human Subjects in Research

Part A.

I have read the Belmont Report, "Ethical Principles and Guidelines for the Protection of Human Subjects of Research" and subscribe to the principles it contains. In light of this Declaration, I present for the Board's consideration the following information, which will be explained to the subject about the proposed research:

1. SELECTION AND SOURCES OF SUBJECTS
   (a) **Source and number of subjects:** Approximately 10 graduate students from the Department of Computer Science will be recruited.
   (b) **Method of recruitment and selection:** Students will be contacted via E-mail or person-to-person contact.
   (c) **Ages:** range from 20s to 40s.
   (d) **Compensation:** N/A
   (e) **Location and duration of experiment:** the experiment will be conducted in the computer laboratory at the Center for the Study of Digital Libraries (CSDL), Room 406 of H.R. Bright building. The experiments will last one to two hours, approximately.
   (f) **Specific steps to ensure confidentiality or anonymity of responses or results:** the subjects will participate in the experiments and be asked to answer questionnaires [see appendixes]. Questionnaire results will be abstracted for publication without revealing subjects' identifying information.
   (g) **Investigator's relationship to subjects:** N/A
   (h) Special **physical or psychological conditions:** N/A

2. EXPERIMENT PROCEDURE
   (a) **Physical/Behavioral Aspects:** the experiment involves a usability evaluation of the software system designed by the principal investigator. The usability of authoring and browsing tools in the system that interact with users (human subjects in this experiment) will be evaluated. The first tool is an authoring tool for hypertext documents generation; another tool is the hypertext document-browsing tool that allows Web-based browsing. To collect information from groups of potential users, the subjects will be instructed to perform a sequence of pre-planned tasks with the software system, then, the subjects will answer questionnaires, in which both qualitative and quantitative evaluations are included.
   (b) **Deception or Coercion:** no deception and nor coercion involved in experiment.

3. RISKS AND BENEFITS TO SUBJECTS
   (a) **Description of any potential risks or discomforts to the subject:** no potential risks are expected to the subjects.
   (b) **Definition of benefits or alternatives for participation in the experiment:** no benefits are expected to subjects. The principal investigator is a graduate student, and his committee chair is the alternative investigator.

4. SIGNITURE ASSURANCE

Principal Investigator/Graduate Student:
I understand Texas A & M University's policy concerning research involving human subjects and I agree:

1. To accept responsibility for the scientific and ethical conduct of this research study;
2. To obtain prior approval from the Institutional Review Board before amending or altering the research protocol or implementing changes in the approved consent form;
3. To immediately report to the IRB any serious adverse reactions and/or unanticipated effects on subjects which may occur as a result of this study;
4. To complete, on request by the IRB, the Continuation/Final Review Forms.

SIGNATURE: Jin Cheon Na DATE: 4/23/01
Jin-Cheon Na (Principal Investigator/Graduate Student)

Chair, Graduate Committee:
I certify that I have read and agree with this proposal, that the PI has received adequate training to perform this research, and will receive adequate supervision while performing this research.

SIGNATURE: _____ DATE: 4/23/01
Richard Furuta (Chair, Graduate Committee)

Department Head, Computer Science:
This is to certify that I have reviewed this research protocol and agree that the research activity is within the mission of the Department and appropriate for the responsibilities and assigned duties of the principal investigator.

SIGNATURE: _____ DATE: 4/23/01
Wei Zhao (Department Head, Computer Science)

APPENDIX C

INFORMED CONSENT DOCUMENT

Informed Consent Document

Please read the following and then sign at the end of this document.

1. I understand that this is a usability study with the goals of informing current research and identifying future research opportunities in the area of hypertext document authoring and browsing using the Petri nets-based hypertext system.

2. I understand that I will participate in the usability evaluation experiments that investigate whether users can effectively use the authoring tool for creating hypertext documents, and browse the result documents using the browsing tool.

3. I understand that I will be given several tasks to complete. The tasks involved will encompass the following: (1) performing the predefined tasks with the software system designed by the principal investigator and (2) upon completion of the tasks, I will complete a questionnaire for the purposes of examining the issues outlined in Paragraph 2.

4. I understand that the assignment will be completed in the computer laboratory at the Center for the Study of Digital Libraries, HRBB 406. The duration of my involvement will be approximately 2 hours. I understand that there will be about 10 participants in the study.

5. I understand that I will not be compensated for my participation in the study.

6. I understand that I will not be exposed to any risks in the study, and that there will be no immediate and direct benefit to me as well.

7. I understand that I may refuse to answer any questions on the questionnaire if the questions make me uncomfortable.

8. I understand that my participation is voluntary and that I may withdraw from the study at anytime with no consequences.

9. I understand that if I am unwilling to complete the assignment in stated in Paragraph 3, my participation will be terminated by the investigator, but I will suffer no consequences.

10. I understand that the study will be confidential that means no identifying information will be elicited.

11. I understand that I will not incur additional costs while participating in this study.

Initial: _____ Date: _____

I understand that this research study has been reviewed and approved by the Institutional Review Board – Human Subjects in Research, Texas A&M University.

For research-related problems or questions regarding subjects' rights, I can contact the Institutional Review Board through Dr. Michael W. Buckley, Director of Support Services, Office of Vice President for Research at (979) 458-4067.

I have read and understand the explanation provided to me. I have had all my questions answered to my satisfaction, and I voluntarily agree to participate in this study.

I have been given a copy of this consent form.

_____

Signature of Subject and Date

_____

Signature of Principal Investigator (Jin-Cheon Na) and Date

For information about the study, please contact either:

Jin-Cheon Na (Principal Investigator)
408D H.R. Bright Building
Texas A&M University
College Station, TX 77843-3112
Phone: (979) 845-4924

Or

Richard Furuta (Chair, Graduate Committee)
Associate Director, Center for the Study of Digital Libraries
Associate Professor, Computer Science
402C H.R. Bright Building
Texas A&M University
College Station, TX 77843-3112
Phone: (979) 845-3839

APPENDIX D

EVALUATION QUESTIONNAIRE

caT Evaluation

ID: _____ Date: _____
Major: _____

Background Information:
Instructions:

- *Please circle the letter/number that you think is the best answer for a given question.*
- *Please do not answer a question if it makes you uncomfortable or you would not like to answer it for any other reason.*

1. What type of computers do you use? (Please circle all that apply)
   a) Apple Macintosh
   b) Windows 95/98/ME/2000/NT
   c) Unix (Sun, HP, Linux, SGI, etc.)
   d) Other (Please specify) _____

2. How long have you authored web pages in HTML or developed web-based software (CGI scripts, Java applets, etc.)?
   a) Less than 6 months
   b) 6 months to a year
   c) A year to two years
   d) More than two years
   e) Not had an opportunity so far

3. Have you known Petri nets before the evaluation?
   a) Yes (where did you learn?) _____
   b) No

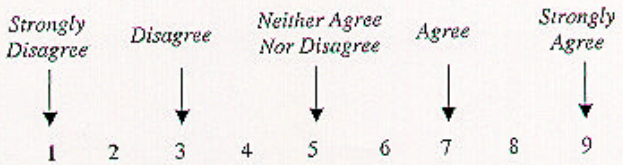
4. Have you used Petri net tools?
   a) Yes (Please specify all) _____
   b) No

5. Have you read Web-based customizable publications (newspaper, educational documents, etc.)?
   a) Yes (Please specify all) _____
   b) No

Interface Design and Overall Evaluation:

Instructions:

- *Please answer the questions using the following scale:*



Interface Design:

1. The tool provides intuitive interface (Easy to learn, easy to use).

   1 2 3 4 5 6 7 8 9

2. Authors can easily and independently use the program.

   1 2 3 4 5 6 7 8 9

3. The program runs properly (bug-free).

   1 2 3 4 5 6 7 8 9

4. Please provide your suggestions regarding how the tool may be improved to interact better with the authors.

Overall Evaluation:

1. How would you rate your experience with the tool?

   1 2 3 4 5 6 7 8 9

2. Do you think that the tool will be useful for adaptive web document publishing?

   1 2 3 4 5 6 7 8 9

3. Do you think that the length of time to use the program is not excessive, learning justifies amount of time spent?

   1 2 3 4 5 6 7 8 9

4. Will you use the tool if the tool is available to you?
   a) Yes
   b) No

5. What tasks would you use the tool for?

6. What are the best aspects of the tool?

7. What are the worst aspects of the tool?

8. What other features would you like to have in the tool?

VITA

Jin-Cheon Na was born in Seoul, Korea on July 30, 1964. He earned his Bachelor's degree in Electrical Engineering from Hanyang University, Seoul, Korea in 1987, and earned his Master's degree in Computer Science from Oklahoma State University, Stillwater, in 1990. He worked at Agency for Defense Development (ADD), TaeJon, Korea, as a researcher, from May 1991 through May 1997. He enrolled in the Department of Computer Science at Texas A&M University, College Station, in the Spring of 1998, and started his research under the direction of Dr. Richard Furuta. He worked with the Center for the Study of Digital Libraries (CSDL) at Texas A&M University from July 1998 through August 2001, under the supervision of Dr. Richard Furuta, Dr. John J. Leggett, and Dr. John D. Oswald. His research interests include hypertext, computer-human interaction, digital libraries, CSCW (Computer Supported Cooperative Work), and knowledge-based systems. His permanent address is:

1021-23 Bangbae-3-Dong

Seocho-Gu

Seoul, Republic of Korea

The typist for this dissertation was Jin-Cheon Na.