Dynamic Documents: Authoring, Browsing, and Analysis Using A High-Level Petri Net-Based Hypermedia System

Jin-Cheon Na and Richard Furuta
Department of Computer Science, Texas A&M University
College Station, TX 77843-3112, USA
Tel: 1-979-845-3839
{jincheon, furuta}@cs.tamu.edu

ABSTRACT
caT (for Context-Aware Trellis) was initially developed to support context-aware documents by incorporating high-level Petri-net specification, context-awareness, user modeling, and fuzzy knowledge handling features into Trellis, a Petri-net-based hypermedia system. The browsing behavior of documents specified in the caT model can reflect the reader's contextual (such as location and time) and preference information. Recently, to provide a framework for the authoring, browsing, and analysis of reasonably complex, dynamic documents, we added (or extended) several features in the caT system, providing hierarchical Petri net support, a structured authoring tool, browsing tools for multiple presentations of a particular document's specification, and a Petri net analysis tool. In this paper, we present the extended features of caT and give examples of using caT to define and present various documents, such as formal specification of software requirements and customized Web documents. Since caT is based on a formal model, the behavioral characteristics of developed caT models can be analyzed. Current debugging and analysis tools, integrated into the authoring tool, are also introduced.

Categories and Subject Descriptors
H.5.4 [**Information Interfaces**]: Hypertext/Hypermedia

General Terms
Design, Theory

Keywords
Dynamic documents, Petri-net-based hypertext, Trellis, caT

1. INTRODUCTION
By enhancing the Trellis model [6], caT, an extended Trellis hypertext model and its prototype system, provides good separation among a document's (1) structure, (2) data, and (3) browsing semantics. In caT, the author, using a Petri-net-based authoring tool, first builds a Petri-net-based document structure and then associates net segments with the corresponding content segments. 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" [22], 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 the caT provide dynamic, interactive interfaces rather than static document pages. 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, document components associated with the active net components are displayed.

caT also supports good separation between document specification and presentation, which allows multiple presentations of a particular document's specification. Since the caT implementation is based on a client-server architecture, several client browsers can be active simultaneously, presenting different interpretations of the document and its browsing state. A reader can choose an appropriate presentation of a document by selecting a browser (e.g., to reflect a characteristic of the reader, such as a handicap) or can use multiple browsers simultaneously (e.g., to display an overview while reading or to view multiple media types simultaneously). When read by multiple readers simultaneously, the behavior of caT 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, allowing sharing in cooperative environments. Since Petri nets inherently provide a good environment for synchronization of parallel actions, the caT model also is useful for expressing group interactions within a document structure.

caT [16] [17] was developed to extend Trellis in order to support flexible user (or agent) adaptation to changes in environmental information, such as time, location, and bandwidth/cost, by adding new features to the Trellis system; mainly high-level Petri-net specification and adaptive behaviors that respond to dynamically-changing information. When the reader browses caT documents, the system provides dynamic documents from its collection, incorporating the reader's contextual and preference information. Recently, to provide a framework for the authoring,

browsing, and analysis of reasonably complex, dynamic documents, we added (or extended) several features in the caT system. Firstly, caT incorporates a hierarchical Petri net, based on one of the features of the high-level Petri net described by Jensen [13]—the hierarchical net support allows authoring of documents with reduced graphical complexity, and increases reusability of nets. In particular, addition of hierarchical nets supports structured authoring in the authoring tool. Secondly, existing browsers have been enhanced and new browsers have been added. The resulting collection of browsers, including a text browser, an image browser, and a Web browser, allows multiple presentations of a particular document's specification. Thirdly, an analysis tool has been integrated into the authoring tool to allow verification and validation of the behavioral characteristics of developed document systems. In addition, a debugging tool has been developed to allow an author to simulate and debug nets interactively.

To explain the caT system and show its potential use in dynamic document authoring, we introduce several example documents: an online art gallery information service, an elevator protocol specification, and a personal document path. The first example is a simple online art gallery service providing a complete set of resources to its on-campus patrons and a limited one to its off-campus patrons. Next, the elevator protocol specification, shows the breadth of the model by using it in formal specification of software requirements. The reader can simulate the model (i.e., the elevator protocol) and see separate hypertextual views of the simulation process by using multiple browsers. The third example, the personal document path example, provides selections from a personal collection of regularly-accessed documents based on the reader's schedule (access time) and other preferences.

This paper is organized as follows. Petri-net-based hypermedia systems, including Trellis, are discussed in the next section. The formal caT model and its major features are described in section 3. In section 4, we introduce dynamic document examples described in the caT model. Current debugging and analysis tools, integrated into the authoring tool, are introduced in section 5. Related works are described in section 6. Finally, implementation details and conclusion are in section 7 and 8, respectively.

2. PETRI-NET-BASED SYSTEMS

The process of reading a hypermedia document can be represented as a task, which consists of sequential and parallel activities. An easy modeling of synchronization schemes together with the possibility to analyze important properties make Petri nets a good candidate for hypermedia document modeling. A basic Petri net [19] is graphically represented as a bipartite directed graph, in which the circular nodes are called *places* and the bar nodes are called *transitions*. A dot in a place represents a *token*, and 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*. An enabled transition may *fire* by removing one token from each of its input places and putting one token into each of its output places. However the basic Petri net is not always convenient for representing and analyzing complex systems because tokens in the basic Petri nets have no identity. In order to overcome this problem, Petri nets that allow tokens to have distinct identity, called high-level Petri nets, were proposed. In high-level Petri nets, a token can be a composite object carrying data. A more thorough exposition of net theory can be found in the texts by Peterson [19] and Jensen [13].

There are some Petri-net-based models that focus mostly on the specification of multimedia data stream synchronization requirements. But, these models are not general enough for the specification of general hypermedia applications. These systems mainly provide presenting capabilities with no (or few) user interaction capabilities. Some of the Petri-net-based models that satisfy the requirements of general hypermedia applications are Trellis [6] [22] [23] [24], HTSPN (Hierarchical Time Stream Petri Net) [21] [27], MHPN (Multimedia Hypermedia Petri Net) [26], and MORENA (Multimedia ORganization Employing a Network Approach) [1].

The Trellis project, initiated in the late 1980's, investigated the structure and semantics of human computer interaction, mainly in the context of hypertext (hypermedia) systems. In the form of colored timed Petri nets, the Trellis model associates content with places and links with transitions. When a place is marked, the corresponding content is displayed, and when a transition is enabled, the corresponding link can be selected. In the Trellis implementation, information presentation is based on a simple paradigm; namely, every node of the Petri net is shown on a different window. Due to its Petri net paradigm, Trellis provides the executable model, allowing for quick prototyping of process protocols. Thus, the Trellis project examined hypertext as a mechanism not only for structuring information, but also for structuring process in the areas of Software Engineering [7] and Computer Supported Cooperative Work (CSCW) [8]. We can conceive of the colored Petri nets as the task (or process) description and the associated contents as the information required by the task. A more detailed overall description of the Trellis project is given in [6].

3. caT

caT (Context-Aware Trellis) has been developed by incorporating new features such as hierarchical colored timed Petri-net specification [13], context-awareness [4], user modeling [2], and fuzzy knowledge handling [28], into the Trellis hypertext model. In this section, the formal definition of the caT model is introduced first and the major extended features for dynamic document authoring are followed.

3.1 Formal Definition of the caT Model

The formal definition of the caT model is specified by adding its extended features to the existing Trellis formal definition [22]. A caT hierarchical Petri net structure is a tuple, HCPN = <S, ST, STM, IOM, I>, in which

- $S = <\Sigma, P, T, A, \tau, C, G, E>$ is a set of pages; each page $s \in S$ is a non-hierarchical CPN = $<\Sigma_S, P_S, T_S, A_S, \tau_S, C_S, G_S, E_S>$; 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 $N \geq 0$;
  - $T_S = \{t_{S1}, t_{S2}, \ldots, t_{SM}\}$ 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: $T_S \rightarrow \{0, 1, 2, \ldots\} \times \{0, 1, 2, \ldots, \infty\}$ is a time function;
- C_S: $P_S \rightarrow \Sigma_S$ is a color function;
- G_S: $T_S \rightarrow$ *Boolean Expression* is a guard function;
- E_S: $(T_S \times P_S) \rightarrow$ *Arc Expression* is an arc expression function.

- 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 an input/output mapping function; P_{source} is a set of places (especially input or output places of ST transition) in the source page, P_{target} is a set of places in the target page (STM(st));
- I $\in$ S is a start page.

In the caT model, each token can have a color value and optional local token variables. Thus, each place can have different types of tokens (i.e., tokens with different local variables), declared by the color function, $C_S(p_S)$, which maps each place to a color set (type). The time function, $\tau_S(t_S)$, maps each transition to a pair of values termed release time and maximum latency respectively. For any transition $t_S \in T_S$, we write $\tau_S(t_S) = (\tau^r, \tau^m)$ and we require that $\tau^r \leq \tau^m$.

The guard function, $G_S(t_S)$, is used for mapping a *Boolean* expression to each transition, which specifies an additional constraint (threshold) which 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 for mapping 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, $P_l = <D_l, W_l, B_l>$, 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.

3.2 Hierarchical Net

Hierarchical structure support is essential for developing large Petri net applications since it reduces graphical complexity problems that are common to graph-based modeling tools, and increases net reusability. In Trellis [23], subnets are handled by the browser—when a token arrives into a place where a subnet is declared to be invoked, 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. Consequently, there are no closely coupled interactions between subnets, such as control-flow coupling or data value (i.e., token value) passing.

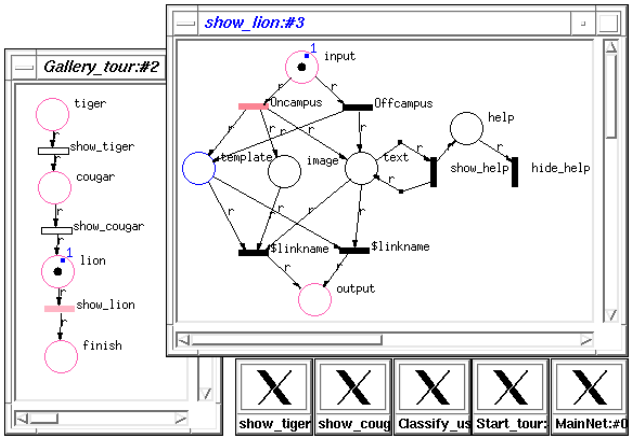
caT incorporates a hierarchical Petri net, based on one of the features of the high-level Petri net. caT shifts the responsibility for subnets from the browser to the Petri net engine (i.e., an information server that receives client requests for its service), in essence combining the nets into a single unit. In other words, in caT, a Petri net engine can hold multiple subnets, which are hierarchically linked; in Trellis, it can hold only one subnet. Additionally, in caT, a structured authoring tool allows building and browsing of hierarchical subnets.

In caT's representation, a transition in a higher-level net may be mapped to a separate subnet. Additionally, places in the higher-level are mapped to places in the subnet—generally the input to the subnet corresponds to places that lead into the mapped transition and the output from the subnet corresponds to places that come from the mapped transition. When a token arrives in a mapped place, it appears simultaneously in both nets. Consequently, tokens in the higher-level net are 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.

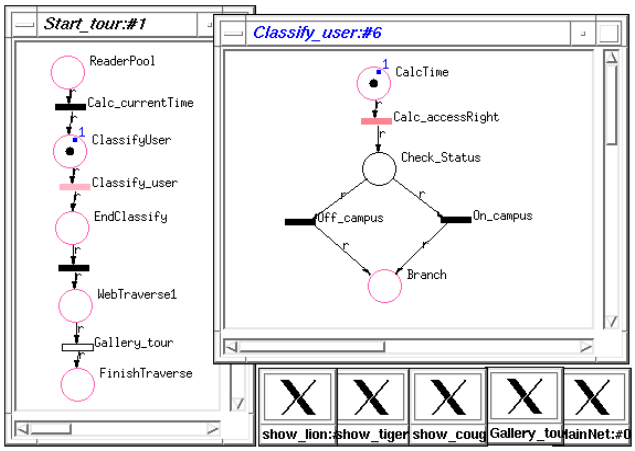
For more detailed explanation, we will consider the simple case of an online art gallery, which provides a complete set of resources to its on-campus patrons and a limited one to its off-campus patrons. The net in the left window of Figure 1(a), *Gallery_tour:#2*, specifies a path for traversing three image pages in sequential order. The subnet in 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, including 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. 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 authoring tool, the inactive transition is represented as a black rectangle bar; fireable transitions are flagged by coloring them red. The subnet transition is represented as a rectangle bar with empty fill; it is filled with pink color when there are active transitions inside of the corresponding subnet.

In the figure, the subnet's *input* place is mapped to *lion* and 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 1(a). Token passing between subnets allows subnets to be used as reusable components, like functions or procedures in high-level programming languages. Additionally, the parent net can pass presentation-related values to child subnets, such as

contents (i.e., file names) of the *image* or *text* places in *show_lion:#3*. All three subnet transitions in *Gallery_tour:#2* expand to the same structure using the same subnet, 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, a browser will display contents of the *image* and *text* places for the on-campus user (the usage of the *template* node will be discussed in the later section).



(a) Show lion



(b) Classify user

Figure 1. Gallery tour

3.3 Context-Aware Adaptation

Supporting context-aware hypertext adaptation requires incorporation of dynamically-changing information from the external environment. caT accomplishes this using four mechanisms: (1) the token's local-variable/value pairs; (2) the assignment statements associated with output arcs, which modify these values during traversal; (3) a global user-modeling profile; and (4) the conditional predicate statements, which are attached to transitions.

Adaptive hypermedia systems [2] use knowledge represented in a user model to adapt the information and links being presented to the given user. In caT, each colored token can have a connection 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. Additionally, a process external to the caT system might maintain a record of the user's current location—this is communicated to caT by updating the associated user-model profile value. Consequently, the user model profile is globally visible 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. When the user does not have a user profile in the system, predefined default values stored in the default user profile are used.

The threshold values for transition firing are determined by evaluating conditional predicates using the values associated with the current token and the current user's user-model. The conditional predicates can be used with Trellis' timing values to provide more dynamic control of the reader's traversal. Time values are thought of as defining ranges for the availability of an event. Therefore, when both the conditional and timing predicates are defined in a transition, they must be satisfied at the same time before the transition can be active. caT's conditional predicate is similar to a Storyspace guard field [14]. 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.).

As an example, we return to our earlier art gallery example. Figure 1(b) 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 previously shown in Figure 1(a). In this example, the gallery's owners have decided that full access (on-campus-access) always will be allowed for patrons on the campus subnet as well as during daytime hours for people located physically near the campus and for faculty members, regardless of location. Beginning in the subnet *Start_tour:#1* in the left side window, an assignment expression statement attached to the output arc of *Calc_currentTime* invokes a built-in time function to get the current time of day. Next, the subnet *Classify_user:#6* is invoked. Here, a statement on the output arc of *Calc_accessRight* determines whether the user fits either of the "high access" conditions (faculty or nearness to campus during daytime). It does this by invoking the fuzzy inference engine to infer an access right value with input values of the current user's location and the previously-acquired time-of-day. caT supports a function for invoking an external fuzzy logic engine to handle uncertain user contexts—the engine used is Matlab's Fuzzy Logic Toolbox [15]. 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. To assign a classification to the user, the returned access right value and the user's network address information are evaluated by the *Off_campus* and *On_campus* transition predicates. Assignments on the output arcs from the transitions assign the appropriate classification value to a variable carried by the token.

3.4 Flexible Presentation

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. For the Web presentation, a template file specifies how active content elements are to be displayed and how links are to be embedded. The node that contains the template file, called the *composite node*, can be viewed as a virtual composite node [10] constructed from several atomic nodes. The Template file has a 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',…,'transNameN'"]
[*link_location* = "below" | "above"]
[*link_sort* = "yes" | "no"]
[*link_duplicate* = "no" | "yes"]]>

The template file is essentially an HTML page with placeholders indicating where information is to be embedded when the corresponding places and transitions are marked and enabled, respectively. Each template file has a <Template> construct placed in front of the other constructs, followed by any numbers of <Transition> and <Place> constructs. Descriptive text can be placed in between constructs. <Template> specifies global settings for current page generation. <Transition> specifies an embedded link, which corresponds to a link in current subnet. We can also use the <Transition> construct within the place's content documents. <Place> specifies a target place and display options for the output links of the place. The <Place> construct is substituted with the content of the target place only when the place is active. The actual usage of each construct will be illustrated in the next section.

4. DYNAMIC DOCUMENTS

4.1 Elevator Protocol Specification

Petri nets are one of the mechanisms that have been examined for their usefulness in providing formal specification of software requirements [9]. In addition, hypertext has been suggested as an important component of systems supporting the software engineering process. Thus a requirements specification describing the actions of a building's elevator was presented previously in Trellis [7]. However the specification of the net suffers from graphical complexity, even though it considers only one elevator, since it is represented with one-level (i.e., non-hierarchical) specification with simple (or non-structured) tokens. The one-level specification of the net requires that related components be repeated multiply in the description. In caT, the extended features, especially hierarchical net and structured tokens, allow specifying the elevator specification in more compact fashion.

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

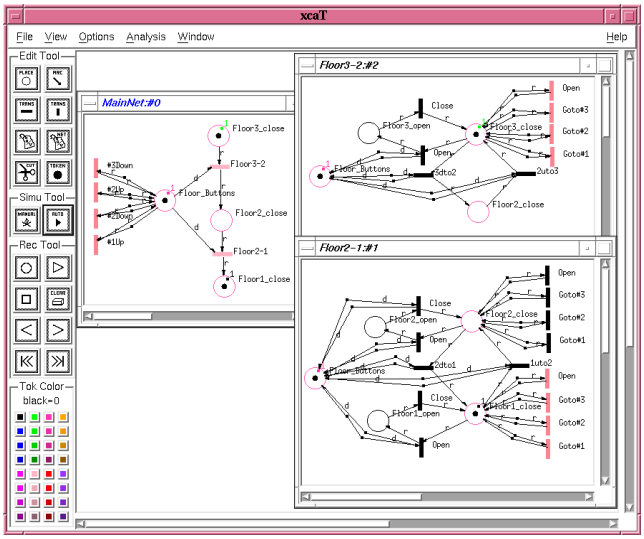


Figure 2: Elevator specification

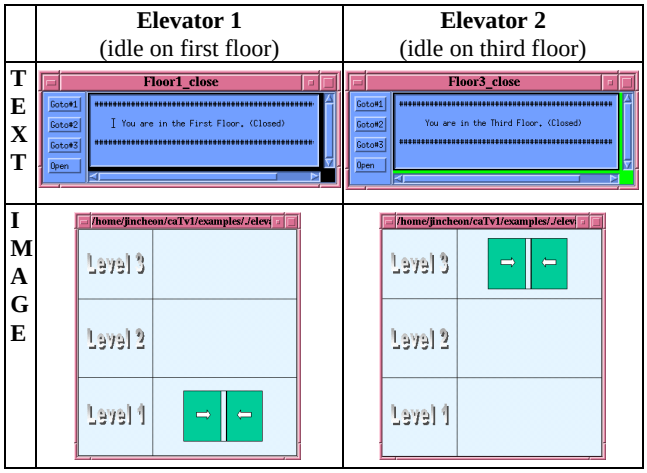
Figure 2 shows a caT elevator specification, which consists of three subnets. In the figure, the display is of the caT authoring tool, which is a modified version of χTrellis' χTed editor: χTrellis is the latest version of Trellis and χTed is a single window-based authoring tool. The *Floor3-2* and *Floor2-1* 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 *Floor_Buttons* in each subnet share the same token through the mapping. In this way, the some tokens in the main net are shared with its subnets—this is similar to a global variable. For example, in *MainNet:#0*, the token in the *Floor_Buttons* place stores the status of button pressed, such as *#3Down, #2Up, #2Down,* and *#1Up*, and the token information is used in the other subnets. In caT, values can be associated with tokens, and here each different colored token is used for different purposes. In *MainNet:#0*, the *maroon* token in *Floor_Buttons* represents the status of 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 button pressed, such as *open, goto#3, goto#2,* and *goto#1*. In addition, conditional statements, associated with transitions, are specified for controlling the token

(elevator tokens) flows; expression statements, associated with output arcs, are specified for updating the status of button pressed into token local variables; and timing values cause the simulation to react dynamically to the rider's action (i.e., button selection).

As a result, caT's elevator specification is simplified, compared to the previously-mentioned Trellis specification, because modular subnets, separating distinct functions into separate descriptions, can be specified and interrelated, and because information about the elevator's state can be encoded into the structured token.

In caT, each place may contain several different types of contents, such as text, image, and sound, for use in selective presentations. For example, the image browser shows the image content of the active place. Figure 3 shows the application of selective presentation during simulation of this specification using two separate browsers: the standard χTrellis browser, `xtb`, and the image browser. Figure 3(a) shows an `xtb` display, which allows selection of links representing the elevator buttons on the wall on each of the floors—the buttons are grouped here for convenience. In the browser, when a color for browsing is specified, only places with tokens of the given color will generate a new window. Figure 3(a) is the presentation of `xtb` with the color *maroon*. Figure 3(b) shows the status of two elevators: the first elevator (*black* token) is idle on the first floor, and the second elevator (*green* token) is idle on the third floor. The windows in the "TEXT" row show text-based presentations of `xtb` with the corresponding color token. Selection of links corresponds to pushing buttons inside of the elevator. Following the links (i.e., the execution of the net) produces displays that correspond to the states in the operation of the elevator. Simultaneously, the windows in the "IMAGE" row show graphic presentations of the image browser—in the image browser, the available links appear in a separate control window.

(a) Presentation of floor buttons



(b) Presentations of two elevators

Figure 3: Executing the elevator specification

The elevator protocol specification example shows how separate browsers can provide different views of a document's state, which viewed together increase the understanding of the document's content. It also illustrates the duality between document specification and software requirement specification in this context. The concepts of rapid prototyping and incremental development are useful both in the requirements specification context as well as in the dynamic document writing context. After the specification is defined in caT, the reader can execute (or simulate) the output model and see his (or her) interested hypertextual view of simulation process using different browsers. In addition, the analysis tool, integrated with the authoring tool, helps to investigate the output specification, identifying conditions such as dead links (i.e., deadlock in the software context).

4.2 Personal Document Path

A personal document path application allows a reader to identify a favorite collection of Web documents and to associate them with contextual information. On browsing, the personal document path provides contextualized information to the reader based on his current environment. The personal path example considers the following simple scenario of personal space. When the reader first accesses a personal path initial page during office hours, he will be asked if he is busy or not. If he is not busy, he will be presented with a short path containing Web pages such as news, weather, technical information, local Web server usage report page, etc., before seeing a research-related search engine page. If he is busy, he will see the research-related page directly. When the reader connects during non-office hours, he will see his favorite general purpose Web search engine after seeing the brief path of Web pages.

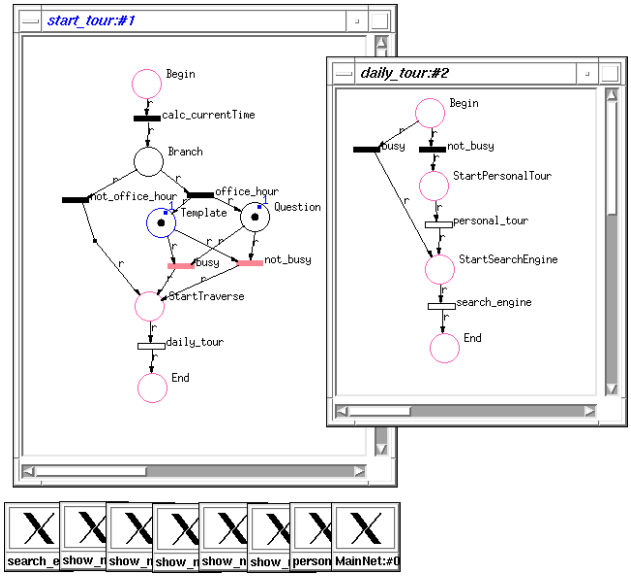
Figure 4(a) shows the model with *start_tour:#1* and *daily_tour:#2* subnets on top and the other subnets iconized. In the *start_tour:#1* subnet, a token starts from the *Begin* place, and gets the current access time. Based on the access time, the token moves to either the *office_hour* or *not_office_hour* transition (these transitions have *Boolean* conditional statements and are fired by the system automatically without user interaction using a time-based Petri-net feature). When the token passes through the *office_hour* transition, a query document (see Figure 4(b)) for the reader's feedback is generated and after getting the reader's input, the token moves to the *StartTraverse* place. The reader's feedback information is stored in the current token and that value is used for matching with conditional statements attached to transitions afterward. When the token passes through the *not_office_hour* transition, it directly reaches the *StartTraverse* place bypassing the query document generation.

The template file attached to the *Template* node in the *start_tour:#1* net is as follows:

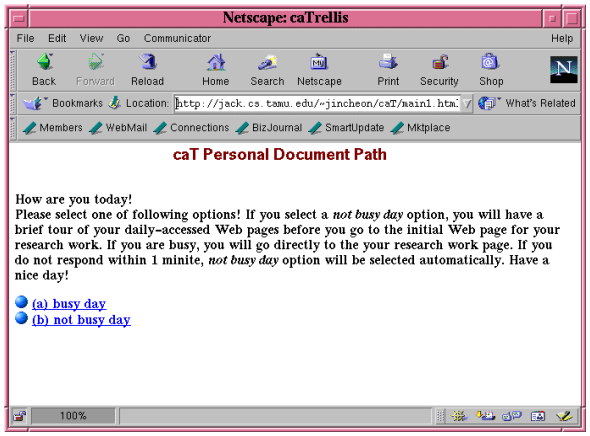
```
<Template multiple_frame="no">
<Place name="Question" link_display="none">
<Transition name="busy" activelink_name="(a) busy day">
<Transition name="not_busy"
            activelink_name="(b) not busy day">
```

The first two lines specify that the output page will be frameless, and output links attached to the *Question* place will not be

displayed automatically after the content of the *Question* place. Instead, <*Transition*> constructs (the last two constructs) are used for specifying active link names of the output links. The output Web document from the current net status is shown in Figure 4(b).



(a) Nets when accessed during office hours

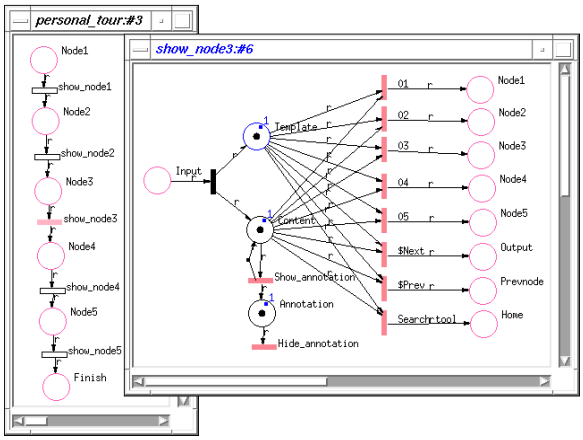


(b) Web browser-based display

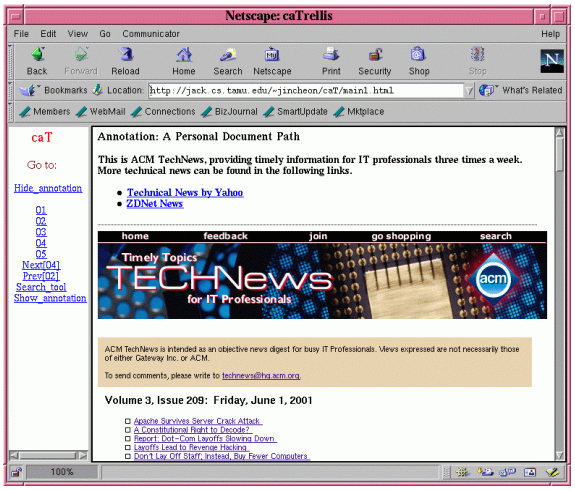
Figure 4: Personal document path: start_tour net

In the *daily_tour* substitution (or subnet) transition, *StartTraverse* and *End* in *start_tour:#1* is mapped to *Begin* and *End* in the *daily_tour:#2* subnet, respectively. Therefore, when a token arrives to *StartTraverse*, *Begin* in *daily_tour:#2* gets the same token value. In the *daily_tour:#2* subnet, the token moves to the *StartPersonalTour* place before it goes to *StartSearchEngine* if the reader is "not busy" or if access is during non-office hours; otherwise it moves directly to the *StartSearchEngine* place. Thus, a busy reader will skip the *personal_ tour* subnet.

Figure 5 shows the subnets and a corresponding output page when the reader is browsing the third node of the *personal_tour:#3* subnet (only related subnets are displayed). Nodes in *personal_tour:#3* have input/output mappings with nodes in children nets. For example, in the *show_node3* transition of *personal_tour:#3*, *Node3* is mapped to *Input* and *Node3* in the *show_node3:#6* net; *Node4* is mapped to *Output* and *Node4*; additionally, the other nodes have mappings for allowing access from a subnet to the other subnets through a parent net.



(a) Nets when the third node accessed in the path



(b) Web browser-based display

Figure 5: Personal document path: personal_tour net

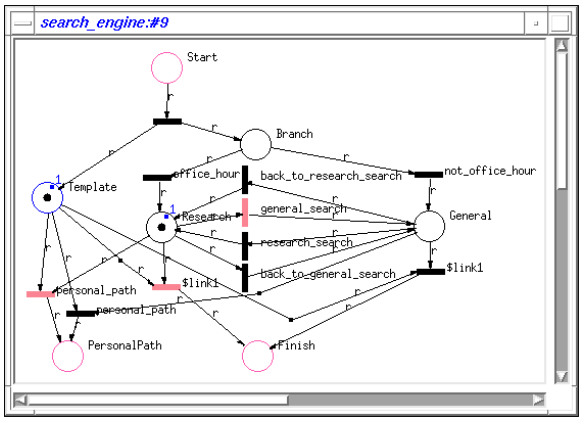
Figure 5(a) represents subnets and their marking status when the reader sees the third page with annotation. The template file of the *Template* place in the *show_node3:#6* subnet is as follows:

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

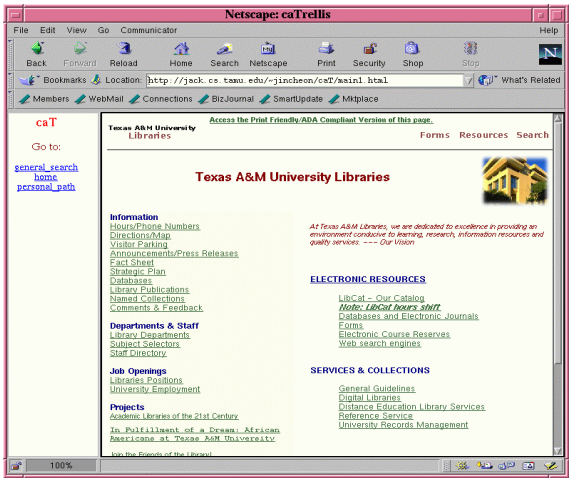
According to this template specification, the output page will have two frames. Output links from the *Annotation* and *Content* places will not be displayed in the main frame, and the links on the left control frame will be used instead for path browsing. The system

replicates the transition-associated links in the left control frame. If normal Web links are followed, changing the information display from that generated by caT, the links in the left control frame allow the user to continue to traverse the net specification. Figure 5(b) shows an output page corresponding to Figure 5(a). The reader sees the third page with the annotation after clicking the *Show_annotation* link, and can hide the annotation by clicking the *Hide_annotation* link.

Figure 6 shows a subnet and a corresponding output page when the reader browses the *search_engine:#9* subnet. When the reader browses during office hours, he will see a research-related search page; otherwise he will see a general Web search engine page. In the subnet, the *office_hour* transition has a condition statement that is satisfied only when tokens have an access time within an office hour range. Figure 6(a) represents a subnet and its marking status when the reader sees a research-related search page. The template file for the *search_engine:#9* net is the same as the one used in the *show_node3:#6* subnet except for the place names (*Research* and *General*). Figure 6(b) shows an output page corresponding to Figure 6(a). When the reader sees a research search page, he can move to a general search page or back to a personal path page. After he moves to the general search page, the reader can move back to the research search page.



(a) Net when accessed during office hours



(b) Web browser-based display

Figure 6: Personal document path: search_engine net

The personal document path example shows how the caT system supports browsing of contextualized Web documents. The author can describe various dynamic documents using the caT system, and the reader can browse the dynamic documents using Web browsers.

5. DEBUGGING AND ANALYSIS TOOLS

The authoring tool supports two kinds of tools for net analysis. The first is an interactive debugging tool and the second is an analysis tool with which the author can analyze the hierarchical Petri nets by building a Reachability Tree (RT). Currently, the following characteristics can be verified by the analysis tool: terminal state existence (i.e., if a state *m* exists in which no transitions are enabled), boundness (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 marks, boundness is important since the tokens may represent limited system resources. In the context of Trellis, [25] explores the analysis of basic Petri nets for various browsing characteristics.

To activate the debugging tool, the user first pushes a recording button (see the "Rec Tool" buttons in Figure 2). Thereafter, when 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 boundness, set by the author, to check overflow places. Additionally, the author can move backward from the current marking since each transition fire is saved. After finishing the traversal, the author can replay the traversal path using a play button, which 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 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: maximum token number in a place, maximum analysis time, maximum marking number, and maximum dead marking number. In general, the time complexity of Petri nets is very expensive. Therefore, the author would likely start with moderate option values, such as 3-minute analysis time rather than unlimited time. Similarly, if maximum token number is not set, a RT may have infinite markings especially when we analyze an unbounded net. Therefore, initially authors are likely 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, adding tokens does not increase the token count and is flagged as an overflow for that place. After building a RT, the author can see the result, and can 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.

6. RELATED WORK

Most of the Petri-net-based hypermedia systems support a similar scheme to that of Trellis. MORENA [1] is targeted at the description and execution of hypermedia applications allowing flexibility and adaptability through message passing. Composite nodes, defining a hierarchy, have their own structure (nodes,

transitions, and arcs) and encapsulate information (such as synchronization specification). MHPN [26] links Petri net objects to MHEG (Multimedia and Hypermedia information coding Expert Group) [12] objects 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. The HTSPN [21] model enables a unified specification of temporal and logical synchronization within hypermedia systems, using the Dexter model [10] and Time Stream Petri Nets model (TSPN) [5] as background. An interpreted version of the HTSPN model (I-HTSPN [27]) provides means to specify hypermedia document on top of the hypermedia structure.

In terms of Petri net structure, caT is unique in supporting the structured token among the Petri-net-based systems. The local-variable/value pairs associated with caT's structured token help to reduce specification complexity; the structured token allows net status information to be associated with the token, rather than requiring its encoding into the state of the Petri net. In addition, in caT, a transition is mapped to a subnet along with input/output mappings between places to serve as input/output to the subnet. The advantage of the expansion of a transition over the one of a place (used in the other systems) is that this permits specification of multiple input and output places and allows more flexible interface between main net and subnet.

As an alternative to Petri-net-based models, HMBS (Hypertext Model Based on Statecharts) [3] uses the structure and execution semantics of statecharts [11] to specify both the structural organization and the browsing semantics of a hyperdocument. HMBS is a model for applications that have a hierarchical structure, such as books, scientific papers, etc. XHMBS (the eXtended Hyperdocument Model Based on Statecharts) [18] is an extension of HMBS with additional mechanisms for describing the time sequencing and information synchronization requirements typical of multimedia.

SMIL (Synchronized Multimedia Integration Language) [20] is a meta-language that allows authors to write interactive multimedia presentations. It supports effective timing and synchronization, adaptation to users and systems, and modeling of a flexible presentation and user interface. SMIL 1.0 became W3C Recommendation on 15th June 1998, and SMIL 2.0 is expected to become a W3C Recommendation soon. Currently the RealPlayer 8, GR*i*NS, and Quicktime 4.1 players support SMIL 1.0. However, SMIL is a script language and does not support analysis to the same degree that Petri-net-based systems do.

7. IMPLEMENTATION

In the caT implementation, every caT model is an instance of an information server—an engine that receives remote procedure call (RPC) requests for its services. Clients are separate processes that may have visual user interfaces and communicate with one or more engines via RPC. The server and most client tools (the authoring tool and browsers) have been implemented in C, C++, and Motif, and run in a Linux environment. caT's Web presentation required inclusion of a new intermediate message-handling server to translate between Web-browser-based Java applets and the information servers. The general architecture of the caT system is shown in Figure 7. Autofire, a client with no visual interface, supports automatic firing of transitions that have timing predicates.

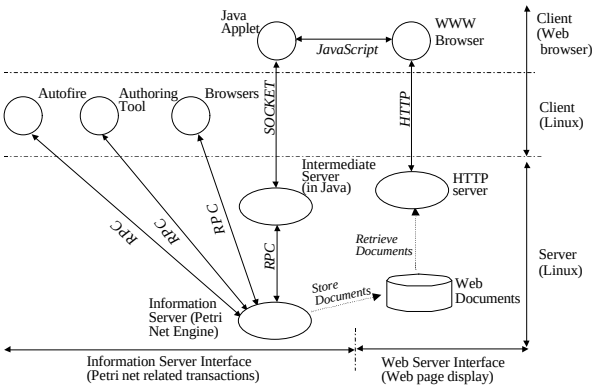


Figure 7: caT system architecture

8. CONCLUSION

caT was developed to support context-aware documents by extending the Trellis model. In this paper, we introduced the extended features of caT that are essential for the authoring, browsing, and analysis of fairly complex, dynamic documents: hierarchical nets, the structured authoring tool, flexible presentation using various browsers, and analysis tools. Then, we gave examples of using caT to show its usability for specifying and presenting various dynamic documents: executable formal specification of software requirements and customized Web documents. 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, and the reader, who is not necessary to know the model, can browse the dynamic documents using various browsers, including Web browsers, to have his selective document views in different environments. In addition, the analysis tool allows the author to investigate the document specification before it is delivered to the reader. In the near future, we plan to investigate and develop more interesting documents, such as cooperative documents, in order to evaluate and enhance the usability of the caT system.

9. REFERENCES

[1] BOTAFOGO, R. and MOSSÉ, D. "The MORENA Model for Hypermedia Authoring and Browsing", In *Proceedings of the International Conference on Multimedia Computing and Systems* (Los Alamitos, Ca., USA, May 1995), IEEE Computer Society Press, pp. 42-49.

[2] BRUSILOVSKY, P. "Methods and techniques of adaptive hypermedia", *User Modeling and User Adapted Interaction, 6*, 2-3 (1996), pp. 87-129.

[3] DE OLIVEIRA, M.C.F., TURINE, M.A.S., and MASIERO, P.C. "A Statechart-Based Model for Hypermedia Applications", *ACM Transactions on Information Systems, 19*, 1 (Jan. 2001), pp. 28-52.

[4] DEY, A.K. and ABOWD, G.D. "Toward a Better Understanding of Context and Context-Awareness", *Georgia*

Institute of Technology, GVU Technical Report GIT-GVU-99-22, (June 1999).

[5] DIAZ, M. and SÉNAC, P. "Time stream Petri nets, a model for multimedia streams synchronization", In *Proceedings of the First International Conference on Multi-media Modeling*, (1993), pp. 257-273.

[6] FURUTA, R. and STOTTS, P.D. "Trellis: a Formally-defined Hypertextual Basis for Integrating Task and Information", Lawrence Erlbaum Associates, (2001), pp. 341-367.

[7] FURUTA, R. and STOTTS, P.D. "A hypermedia basis for the specification, documentation, verification, and prototyping of concurrent protocols", *Technical Report TAMU-HRL 94-003*, Texas A&M University, Hypertext Research Lab, (June 1994).

[8] FURUTA, R. and STOTTS, P.D. "Interpreted Collaboration Protocols and their Use in Groupware Prototyping", In *Proceedings of ACM 1994 Conference on Computer Supported Cooperative Work*, ACM, (Oct. 1994), pp. 121-132.

[9] GHEZZI, C., JAZAYERI, M., and MANDRIOLI, D. "Fundamentals of Software Engineering", Prentice Hall, (1991).

[10] HALASZ, F. and SCHWARTZ, M. "The Dexter Reference Model", In *Proceedings of NIST Hypertext Standardization Workshop*, (1990), pp. 95-133.

[11] HAREL, D. "Statecharts: a visual formalism for complex systems", *Science of Computer Programming, 8*, (1987), pp. 231-274.

[12] ISO 13522-1. "Information Technology – Coding of Multimedia and Hypermedia Information – Part 1: MHEG Object Representation, Base Notation (ASN. 1)", (1994).

[13] JENSEN, K. "Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use Volume 1", *EATCS Monographs on Theoretical Computer Science*, Springer-Verlag, (1992).

[14] JOYCE, M. "Storyspace as a hypertext system for writers and readers of varying ability", In *Proceeding of Hypertext '91* (San Antonio, USA, Dec. 1991), ACM, pp. 381-387.

[15] The MathWorks, Inc. "Fuzzy Logic Toolbox User's Guide", (1999).

[16] NA, J.-C. and FURUTA, R. "Context-Aware Digital Documents Described In A High-Level Petri-net-based Hypermedia System", *Digital Documents and Electronic Publishing (DDEP00)* (Munich, Germany, Sept. 2000).

[17] NA, J.-C. and FURUTA, R. "Context-Aware Hypermedia in a Dynamically-Changing Environment, Supported by A High-Level Petri Net", In *Proceedings of Hypertext '2000* (San Antonio TX, USA, May 2000), pp. 222-223.

[18] PAULO, F.B., TURINE, M.A.S., DE OLIVEIRA, M.C.F., and MASIERO, P.C. "XHMBS: A Formal Model to Support Hypermedia Specification", In *Proceedings of Hypertext '98* (Pittsburgh PA, USA, 1998), ACM, pp. 161-170.

[19] PETERSON, J.L. "Petri Net Theory and the Modeling of Systems", *Prentice-Hall*, Englewood Cliffs, N.J., (1981).

[20] SMIL. "Synchronized Multimedia Integration Language (SMIL 2.0) Specification", W3C Proposed Recommendation (05 June 2001), http://www.w3.org/TR/smil20/.

[21] SÉNAC, P., DE SAQUI-SANNES, P., and WILLRICH, R. "Hierarchical Time Stream Petri Net: a model for hypermedia systems", In *Application and Theory of Petri Nets 1995*, Lecture Notes in Computer Science no. 935, G. DE MICHELIS and M. DIAZ (Eds.), Springer (1995), pp. 451-470.

[22] STOTTS, P.D. and FURUTA, R. "Petri-net-based hypertext: Document structure with browsing semantics", *ACM Transactions on Information Systems, 7*, 1 (Jan. 1989), pp. 3-29.

[23] STOTTS, P.D. and FURUTA, R. "Hierarchy, composition, scripting languages, and translators for structured hypertext", In *Proceedings of the European Conference on Hypertext '90* (Nov. 1990), pp. 180-193.

[24] STOTTS, P.D. and FURUTA, R. "Dynamic Adaptation of Hypertext Structure", In *Proceedings of Hypertext '91* (1991), pp. 219-231.

[25] STOTTS, P.D., FURUTA, R., and CABARRUS, C.R. "Hyperdocuments as Automata: Verification of Trace-Based Browsing Properties by Model Checking", *ACM Transactions on Information Systems, 16*, 1 (1998), pp. 1-30.

[26] WANG, H. K. and WU, J.-L. C. "Interactive Hypermedia Applications: A Model and Its Implementation", *Software-Practice and Experience, 25*, 9 (September 1995), pp. 1045-1063.

[27] WILLRICH, R., SÉNAC, P., DIAZ, M., and DE SAQUI-SANNES, P. "A Formal Framework for the Specification, Analysis and Generation of Standardized Hypermedia Documents", In *Proceedings of Multimedia '96* (1996), IEEE Press, pp. 399-406.

[28] ZADEH, L.A. "Knowledge representation in fuzzy logic", *IEEE Trans. Knowledge and Data Engineering*, *1*, 1 (1989), pp. 89-100.