A Formal Approach to Analyzing the Browsing Semantics of Hypertext

Paul De Bra, Geert-Jan Houben

Dept. of Mathematics and Computing Science, Eindhoven University of Technology,
PO Box 513, 5600 MB Eindhoven, the Netherlands

Yoram Kornatzky

Computer Systems Research Institute, Univ. of Toronto,
6 King's College Road, Toronto, Ontario M58 1A4, Canada

Abstract

Browsing through a hyperdocument often leads to orientation problems: "you don't know where you are, you are lost in hyperspace". Although some of these problems may be diminished by specific browsing tools, such as fish-eye views and history mechanisms, most problems are embedded in the structure of the hyperdocument. In order to detect and analyze potential problems in the use of a hyperdocument, we reduce different types of browsing actions to a unified and well know formalism : the Petri net.

In this paper we use the object-oriented model from [3] to describe hyperdocuments. This model, unlike other reference models [5, 6, 9], describes objects using a number of different description layers: these layers are not fixed by the model, and none of them are excluded from the model.

In order to use Petri nets to describe browsing, the operations dealing with the hierarchical structure of objects (composites, towers and cities) are translated into operations that are equivalent to the simple follow-link. Unlike Stotts, Furuta and Ruiz [14] we combine the operations offered by the hypertext system (or presentation layer) with the link structure of the document, in order to characterize the browsing semantics of document and system as a unity.

By using existing analysis tools for (different kinds of) Petri nets one can easily detect potential browsing problems like unreachable nodes and links, dead ends, unlimited numbers of simultaneously displayed nodes, infinite loops, etc. and analyze the consequences. Such analysis is most valuable to hypertext authors, whose biggest problem is the creation of the most appropriate links.

1 Introduction

The link structure in hyperdocuments is often the cause for problems when using (reading and browsing through) hypertext. The dreaded "lost in hyperspace" problem means that the user gets distracted from his original goal and looses track of his current position in the hyperdocument. The user also has no idea how to get back to information nodes that are related to the subject he is interested in. So far no hypertext system has managed to offer a universal solution to this problem, mostly because the problem does not originate in the system but is caused by the link structure of the hyperdocuments. Potential sources for the "lost in hyperspace" problem are :

- meaningless links, sometimes occurring when using hyperdocuments that are generated automatically from written (linear) documents. Even a small percentage of meaningless links can be very disturbing and distracting.
- dead ends, which always occur in hierarchically structured documents, and which force the user to go back (e. g. to the index) and choose a new direction. A backtrack and/or history mechanism may aid the user, but also hides the problems which are still present in the documents themselves.
- loops, often as short as 2 or 3 nodes, which confuse the reader who is trying to follow links to new information (not previously read). Again, the system may help by marking links not leading to new information, but the problem in the hyperdocument remains.

The first problem source can be eliminated by improving the automated understanding of natural language. We will not attempt to deal with this problem. The second and third source seem easy to detect automatically, although hypertext authoring systems typically do not offer tools for doing so. However, as more complicated structures and more powerful user-interfaces emerge, it becomes more difficult to capture the browsing semantics and find loops and dead ends.

In order to analyze the problems with a given hyperdocument one needs a formalism to describe the structure of the document and the potential ways the document can be used. To study the browsing semantics of a hyperdocument Zheng and Pong [19] use the statecharts [7] formalism, which is especially useful for modeling the behavior of hyperdocuments which contain composite structures. Stotts, Furuta and Ruiz [14] use automata theory. The set of all possible paths through a hyperdocument can be viewed as the language accepted by an automaton. The formalism of languages accepted by automata is very well known, and numerous tools for analyzing automata exist.

The main shortcoming in the previously quoted approaches is that they use a sometimes oversimplified view of the possible structures in hyperdocuments. Most hypertext models and systems provide richer structures than simple nodes and links, including Contexts[2], Hypergraphs [17] and other forms of composite objects, like the Activity Spaces in Sepia [15, 16]. In [3] we proposed a more general model: the "Extensible Data Model for Hyperdocuments", or "Tower Model" for short. Apart from basic and composite nodes and links, the model has a *Tower* constructor for describing different levels of description of the same object (somewhat similar to the layers in the Dexter model [6]), and a *City* constructor for describing different views of the same object. Hyperdocuments no longer look like simple graphs, and hence they cannot be easily described by automata. Also, the model does not forbid simultaneous access to different nodes, thus making the browsing process multithreaded.

In the definition of the Tower Model we have not included a browsing language. However, the structures in our model do suggest a number of desirable browsing operations. Apart from the common "follow-link" we will consider operators to "zoom" in and out of composite objects, towers and cities, and operators to "switch" between description levels of a tower or between towers in a city. We will also focus on the concept of "state" of a hyperdocument. This will enable us to describe the behavior of hyperdocuments (formulated in the Tower Model) by using a special kind of Petri nets, which we can emulate using normal Petri nets ([10]). We also show how system features can be translated to additions to the Petri net describing the hyperdocument, so as to be able to analyze the behavior of the hyperdocument together with the system used for browsing. We use a Petri net based approach because of its graphical nature, firm mathematical foundation (Petri nets have been studied for over 25 years now) and powerful analysis methods. Like for simple finite state automata, several analysis tools for Petri nets exist ([8, 10, 18]). Hence the behavior of hyperdocuments can be analyzed with existing tools.

The organization of this paper is as follows: in Section 2 we briefly recall the structural definitions of our Tower Model. We refer to [3] for details. In Section 3 we define the *state* of a hyperdocument, and discuss the possibilities for describing the browsing semantics by means of the possible transitions between states in a hyperdocument. In Section 4 we define the browsing operations (for the Tower Model). We formalize the browsing semantics by means of special Petri nets, for which we then show how to emulate them using normal Petri nets.

2 The Tower Model revisited

We briefly recall the basic definitions of the Tower Model from [3].

Definition 2.1 *A* node *is described by a couple (i, v), where i is the object's identification and v is the node's value (its contents and/or state).*
A link *is described by a 4-tuple (i, s, d, v), where i is the object's identification, s is the link's source anchor, d is the link's destination anchor and v is the link's value (its meaning).*
An anchor *is described by a couple (i, v), where i is the object's identification and v is the anchor's value.*

Although the value of an anchor can be anything, the assumption is that the anchor's value indicates to which node(s) the anchor is connected.

So, a hyperdocument is a set of objects which belong to the class of nodes, links or anchors. A node represents a unit of information. Nodes are connected through links, attached to the nodes by means of anchors. Anchors represent both sources and destinations of links. Typically the source of a link will be a word or phrase occurring in a text-node, while the destination will be a text-node, a picture, a fragment of audio or video, etc.

Links connect anchors, not nodes. This means that whether links between more than 2 nodes are allowed or not depends on the values that can be assigned to anchors, not on the definition of a link. In the Hyperties system the true source of a link is a word or phrase in a dictionary of source anchors, with the effect that every link appears to have many source anchors. So, from every node containing (a copy of) the source anchor one can follow the link to the (same) destination node. Such a link with more than one source anchor however can easily be replaced by a set of links, each having a source anchor that is located in only one node. In a model this makes no substantial difference. And in an implementation like Hyperties it is often not even clear to the user whether the system uses a single link with an anchor in many nodes, or whether it uses many identical links, each originating in only one node.

Having links *to* more than one node at a time is more fundamental. The result of following a link may be that a text-node appears side by side with a video fragment or animation, and this may be accompanied by a sound fragment. Such a link cannot be simulated by three links, each to a single node.

Definition 2.2 *An* object *is a node, link or anchor. The value of each object can be: a* basic *(i. e. application dependent) value, a* composite *value, a* tower *value or a* city *value.*
A hyperdocument H is a triple (N, L, A), consisting of a set of nodes N, a set of links L and a set of anchors A. (For each link $l \in L$ both the source and the destination anchor must be elements of A, and each anchor can only be connected to nodes of N.)

Basic objects (i. e. objects with a basic value) are like black boxes to the hypermedia system. Their value and behavior are application specific. To keep the description of the behavior of an entire hypermedia document simple only the possibilities for navigating (moving) between objects are included, not the full internal behavior of objects[1].

Definition 2.3 *A* composite value *is a 4-tuple (c, n, l, a), where c is a composite constructor, n is a set of nodes, l a set of links and a a set of anchors.*

A *composite constructor* can be any kind of constructor [3], including network, set and list constructors, etc., with all possible kinds of additional global information and constraints. So, not just "networks". However, we only allow constructors whose behavior can be modeled using the kinds of automata we use for describing browsing, e. g. finite state machines or Petri nets.

The nodes, links and anchors that occur in a composite object are not restricted to being basic objects. They may themselves have a basic, composite, tower or city value. Likewise, a composite object can be a composite node, link or anchor. (Most models only allow composite nodes.)

Definition 2.4 *A* tower value *is a mapping from a (fixed and finite) set of labels to a set of objects of the same kind (node, link or anchor). Each label corresponds to a description level, a level of functionality in the hyperdocument. The labels are typically given in a fixed order, in which case a tower value can be described by a tuple of objects.*

Since the objects on each level can be very complex (e. g. composite objects) a tower value is not described as a tuple of atomic values but as a tower with levels or floors, as depicted in Figure 1.

Definition 2.5 *A* city value *is a mapping from a parameter space to a set of tower objects. The parameter space is typically a set of names or a data type.*

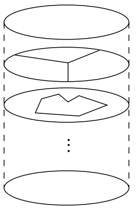

Figure 1: Graphical representation of a tower.

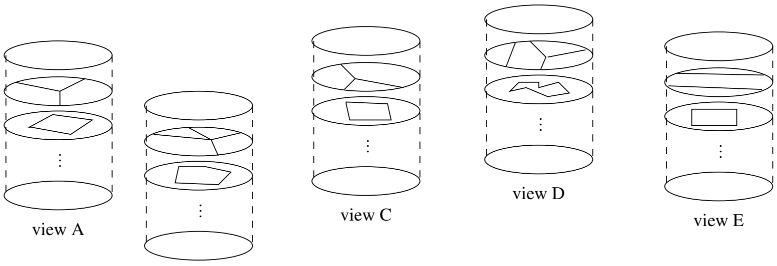

Figure 2: Graphical representation of a city.

A city object (object with a city value) describes a set of views onto an information object. This can be an enumerated set or it can be a (possibly infinite) set of views that are calculated from a common object description. Cities are very much like the ViewSpecs defined by Engelbart [4]. Figure 2 gives a graphical representation of a city. A typical example of a city value is a mapping from angles to cross sections of a three dimensional object. This city object would be a so-called virtual object (i. e. the towers in this city are generated by a function), whereas most objects in a hyperdocument are stored objects.

The set of description levels is usually fixed throughout a hyperdocument. The model does not specify the number of possible levels, but that number will usually be small. Not all objects need a non-empty value for each given level, but generally a fixed number of levels is imposed on the entire document. In a city however, the number of towers (i. e. views) can be very large, possibly infinite, and there is a much less drastic semantic difference between the towers in a city than between the levels of a tower. Because of this distinction it is usually clear which aspects of a hypermedia system should be modeled as towers and which aspects correspond to cities.

3 The State of a Hyperdocument

Before turning to the semantics of browsing we shall introduce the notion of the "state" of a hyperdocument. For this we are mainly interested in the "accessibility" of nodes, links and anchors.

Normally a hypertext system does not control the computer completely. For instance, it may not have complete control over the screen layout. Windows may be obscured because of user or window-manager actions, and information which is intended to be displayed in a window as a whole may be restricted to a smaller window which can be scrolled. Such restrictions are not important for describing the browsing semantics. If the user can access the anchor for a link, for instance by scrolling the window first, or by moving a few windows, we consider the anchor to be accessible. If a node can be made visible by moving windows, the node is accessible.

Definition 3.1 (State of a hyperdocument) *The* state *of a hyperdocument is a function S from the objects in the document to the boolean values* true *and* false. *The value* true *means that an object is accessible;* false *means that it is not.*

For browsing only the nodes and anchors that are presented (on the screen) are important. Hence we only use the part of the state information that is related to navigation through the document.

[1]For now we assume that there are no timing constraints. (We shall remove this restriction later.)

Since the number of objects in a hyperdocument may be infinite (cities may have an infinite parameter space) the state of a hyperdocument may be a function with an infinite domain. However, this infiniteness does not generate problems if we assume that all elements of a city together can only have a finite number of objects. (Thus, a node-city only has a finite number of anchors.) Under this assumption there are only a finite number of possible combinations of accessibility values for the objects in a city.

Since we also assumed (in the previous section) that the structure of a composite can be described by some kind of automaton like a finite state machine or a Petri net, we know that composites have a finite number of components as well. Hence all objects in a hyperdocument are essentially finite, as far as the structure of their interconnections is concerned.

When the user fires up a hypermedia application, a screenful (or windowful) of information appears. The hyperdocument enters its *initial state* in which some objects (nodes and source anchors) are accessible, either because they are visible on the screen, or because it is possible to access them in some other way (maybe through some command language). Source anchors of outgoing links are usually visible as buttons or as highlighted words or phrases. We assume that all (source) anchors connected to an accessible node are also accessible.

The user-interface provides some way to activate an outgoing link, for instance by moving a cursor using a pointing device, and then pressing a button on the keyboard or pointing device. When a link is thus activated, or "followed", the information in the node(s) containing the destination anchor of the link is (are) displayed and the source node(s) are removed from the screen unless they are part of the destination as well. A typical example in which the source nodes remain accessible is the Unix Guide system [1] with in-line (or should we say "in window"?) replacement of source anchors by the destination of a link.[2] A less typical example is Multicard [11], which generates windows on top of each other, thereby creating the illusion that the source node of a link is removed. By moving windows however the source node can be shown to still be accessible. In general all we know is that following a link in a hyperdocument means that the state of that hyperdocument changes, i. e. the set of objects that are accessible changes.

Since there is almost always more than one link to follow there are many paths the user can follow while browsing a hyperdocument. Ideally, there should never be a situation in which there is no link to follow at all. In such a case the user is stuck. We call this a *dead-end* in the hyperdocument. Also, ideally, the user should not follow links that lead to nodes he already viewed or read before, or when he does, he should be made aware of it. We call this a *loop* in the hyperdocument. Of course, since most hyperdocuments have only a finite number of nodes loops and dead-ends cannot both be avoided.

In [14] Stotts, Furuta and Ruiz investigate the browsing semantics of hyperdocuments by primarily analyzing the documents themselves. They pay less attention to the hypermedia system for which these documents are intended and its associated browser features. As a result, they find that many hyperdocuments lead to dead-ends because they have a mostly hierarchical structure. They chose to ignore that the hypermedia system may provide ways to switch from one node to another without the presence of a link (and anchors) in the hyperdocument itself. Most systems provide a button to go back to the last node that was displayed, and another button to go back to a start- or index-node. An example is the Hyperties system [12]. Also, some systems provide a button to remove "old" nodes from the screen explicitly. An example of this feature is found in the Trellis system [13]. This "button" behaves like a link with an empty destination anchor. We argue that the implicit links provided by these system-dependent features should be taken into account (in Section 4 we will show how concepts like guided tours and history-based navigation can be expressed using such (virtual) links). It is impossible to browse a hyperdocument without using the system for which the document was written. When translating documents between systems, the implicit links provided by one system should be converted to explicit links if the destination system does not provide them as a system-dependent feature.

When one assumes that only one node can be displayed at once (and hence that following a link

[2] In "enquiry" mode however, the Unix Guide system behaves differently.

removes the source node from the screen) then the meaning of browsing operations can be easily modeled using finite state machines because there is a one-to-one mapping between the state of a hyperdocument and the node that is displayed. The Tower Model does not make this assumption. A consequence is that the number of possible states becomes 2^n where n is the number of nodes in the hyperdocument. It is clear that the finite state machine representation is no longer suitable. The Trellis system suffers from the same problem because it also does not make this assumption. Experience with Trellis has shown that the number of states needed to represent the browsing semantics of Trellis documents by means of finite state machines does indeed generate very large automata (though not really exponentially large).

4 Browsing Hyperdocuments in the Tower Model

4.1 Follow-link

Definition 4.1 (The follow-link operation) *Let $H = (N, L, A)$ be a hyperdocument, which is in state S. A link $l = (i, s, d, v) \in L$ is* enabled *when $S(n) = true$ for some node n to which s is connected. After following an enabled link l the hyperdocument is in state S', defined as follows:*

- *$S'(n) = false$ for nodes n to which s is connected and d isn't (i. e. source nodes which are not part of the destination become inaccessible).*
- *$S'(n) = true$ for the nodes n to which d is connected (i. e. destination nodes become accessible).*
- *$S'(n) = S(n)$ for all other nodes (i. e. the accessibility of nodes not involved in the follow-link operation remains unchanged).*

One may wish to also describe the effect of following links on the state of anchors and links, but we only look at the accessibility of nodes. When a node is accessible, then so are all the anchors to which it is connected, and all outgoing links from these anchors are enabled. The case in which some anchors connected to accessible nodes may not be accessible themselves is avoided by modeling such nodes as cities with views that are only connected to the accessible anchors.

4.2 Boolean Petri net

The effect of following a link can be shown using a special kind of Petri net, the so-called boolean Petri net. For the completeness of our discussion we start with a short description of standard Petri nets ([10]). A *standard Petri net* $PN = (P, T, F)$ is a bipartite directed graph composed of a finite set of *places* (P) and a finite set of *transitions* (T). The *flow relation* $F \subseteq (P \times T) \cup (T \times P)$ describes the set of *arcs* connecting places and transitions. Place p is an input place of transition t if $(p, t) \in F$ and p is an output place of t if $(t, p) \in F$. Each place contains 0 or more *tokens*. The state of the Petri net is given by the distribution of the tokens over the places. A transition t can fire if and only if each of its input places contains at least one token. Firing t consists of removing one token from each of the input places and adding one token to each of the output places of t.

To describe the effect of following a link we use a special kind of Petri net, in which every place always contains exactly one token, which has the value *true* or *false*. Figure 3 shows the three possible kinds of connections and their effect. A transition can only fire when all input places have a token with the value *true*.

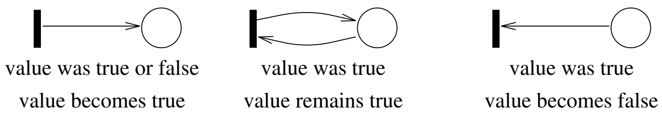

Figure 3: Operations in "boolean" Petri nets

One may need several transitions to represent a link, one for each node the source anchor of a link is connected to. Figure 4 shows the graphical representation of following a link. The link is enabled if node n1 is accessible. Following the link makes nodes n2 and n3 accessible. In this example node n1 remains accessible as well. Hence after following the link the next possible action may be a follow-link from any of the three nodes.

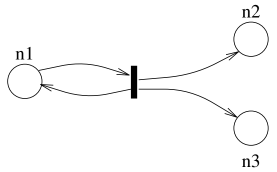

Figure 4: Graphical representation of following a link.

4.3 Representation of Browsing

We will now develop a representation of the browsing structure for a hyperdocument by means of a Petri net. We shall describe this approach in a number of steps:

1. Translate the link structure to a boolean Petri net. Since the link structure may be different depending on the description level of the towers, we look at only one level at a time. Hence we have one Petri net for each description level.

2. Describe the operations to switch between description levels, by adding transitions to the boolean Petri net. These operations connect the Petri nets at the different levels, and thus generate one big Petri net with more browsing possibilities.

3. Add transitions for operations to move between views in a city, in case the city is finite.

4. Add transitions for zooming in and out of objects. These operations are typical for going from a browser-type representation of a composite object to one of the elements of the composite v.v.

5. When all these operations have been added, the resulting boolean Petri net is translated to a standard Petri net, by means of an algorithm (described below), in such a way that the results from analyzing the standard Petri net can be easily translated to the boolean Petri net.

A Petri net based approach is used because of the graphical nature, firm mathematical foundation and the availability of efficient analysis tools and mature tools.

Step 1 (Translation of link-structure to boolean Petri net.) *Let $H = (N, L, A)$ be a hyperdocument. We create a boolean Petri net $BPN = (P, T, F)$ such that:*

- *for every node $n \in N$ there is a place labeled n, i.e. $P = N$;*

- *for every node n connected to the source anchor of a link l there is a transition with a single incoming arrow from n and with one outgoing arrow to each node connected to the destination anchor of l, i.e. $T = L$ and $F = \{(n, l) \in N \times L \mid$ the source anchor of l is connected to $n\} \cup \{(l, n) \in L \times N \mid$ the destination anchor of l is connected to $n\}$.*

- *The initial state S of the hyperdocument is such that $S(n) = true$ for the nodes n that are accessible when the hyperdocument is first "opened", and $S(n) = false$ for all other nodes. The initial marking for BPN assigns the same truth-values to the places as S does to the nodes.*

- *If the hyperdocument consists of Tower objects, the boolean Petri net is created for each description level, thus obtaining as many nets as there are levels. There are no links from objects on one description level to objects on another description level.*

Most hypermedia systems do not offer several description levels with different link structures. Some of the more recent features, including the Activity Spaces in the Sepia system, are indeed implementations of different link structures over a common set of nodes. In Sepia, the different link structures (e. g. a Content Space, Argumentation Space, Rhetorical Space, Planning Space, etc.) can be displayed and edited simultaneously. In this case, moving between windows on the screen actually means moving between description levels.

Step 2 (Moving between description levels) *Let $H = (N, L, A)$ be a hyperdocument. Let, for each existing description level i, $BPN_i = (P_i, T_i, F_i)$ be the boolean Petri net generated in step 1 for level i. We create a boolean Petri net $BPN = (P, T, F)$ which contains $\cup BPN_i$. For each tower node n, we add transitions between the places n_i corresponding to n on different description levels, for each change in description level that is possible in the hyperdocument. Usually the possible moves between description levels will be the same for all objects in a hyperdocument.*

Often it will be possible to go from each description level to each other description level. Graphically we will then draw a line through all description levels. In a composite tower node for instance, the moves between description levels exist for all components in the composite. In a graphical representation, these moves look like service- or elevator shafts in the tower.

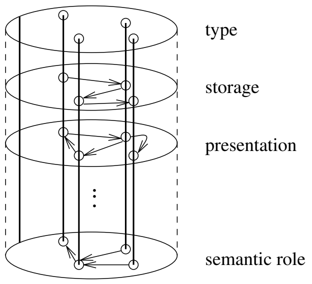

Figure 5: Composite Tower with elevator shafts

Step 3 (Changing views in a city) *Moving between towers in a city is analogous to following links and can be modeled as such in the boolean Petri net. The only special thing is that these links will almost always make the tower (node) connected to the source anchor inaccessible, and these links will have a destination anchor connected to only one tower (node).*

Composite objects are constructed from other objects. We have assumed that the composite constructor is such that its effect on browsing can be represented by an automaton. This can be a finite state machine (easily represented as a Petri net) or a Petri net. If it is a boolean Petri net we can overlay it on top of BPN, otherwise we do so after the translation to a standard Petri net (described later).

Step 4 (Zoom operations) *Composite objects are usually presented in a graphical way. The individual objects are represented in an iconic way, and by selecting such an object, one "opens" or "zooms in on" the object. For each possible zoom operation from a composite to one of its elements we add a transition to BPN, from the composite to the element. This can be done with or without making the composite inaccessible, but normally the composite will remain accessible. The zoom-out operation is similar, although here the element will typically become inaccessible.*

Note that the notion of composite objects in the Tower Model does not make hyperdocuments hierarchical. There are no constraints saying that links to elements of a composite should be forbidden, or that links from an element of a composite to a node outside the composite should be forbidden. Composites in the Tower Model simply provide a way to superimpose a grouping structure over the link structure. By turning the composites into objects themselves it becomes possible to provide links from basic nodes to composites and vice versa, like in Tompa's Hypergraph Model, [17] But, in addition to that model, we also consider composite links, which may for instance represent a series of arguments tying a set of premises to a conclusion. The Argumentation Spaces in Sepia [15] form an excellent example of composite links.

Step 5 (Reduction to standard Petri nets) *Let $BPN = (P, T, F)$ be the result of steps 1 to 4. We create a (standard) Petri net $PN = (P', T', F')$ as follows :*
Each place $p \in P$ is replaced by two places $p_t, p_f \in P'$.
Each transition t is replaced by 2^n transitions, where n is the number of places to which t has an outgoing arrow and no incoming arrow. The simple construction of t is made clear in Figure 6.

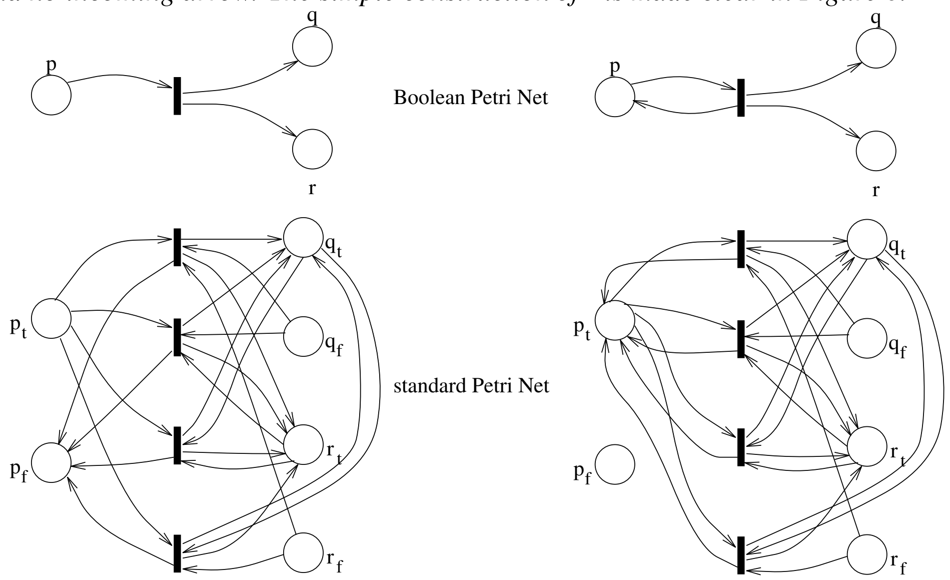

Figure 6: Translation from boolean Petri net to Petri net.

The translation simply replaces a boolean place by a pair of places : one for the value *true* and another for the value *false*. Hence for every node there is a place in which the presence of a token means that the node is accessible, and there is another place in which a token means that the node is inaccessible. (And only one of these nodes can have a token at the same time.) In order to avoid generating more than one token in an output place every transition must "read" the value present in the output place. However, since a boolean place is replaced by two "normal" places, a transition is needed that reads from the *true* place and another transition that reads from the *false* place. (And both transitions write to the *true* place because the destination node becomes accessible.) A transition is needed for every possible combination of truth values in the destination places. When a transition has n output places (which are not also an input place) there are 2^n such combinations, hence an equal number of transitions.

Although there is an exponential factor in the translation, the size of the standard Petri net is only exponential in the number of nodes that are the destination of a single link. One could call this the out-degree of the link. In most hyperdocuments this number is 1, but in some cases this number may be larger than one, but it will always be a very small number. In any case, this translation keeps a close relationship between the nodes and links in the hyperdocument and the places and transitions in the (standard) Petri net. Note that the total number of tokens in the net is constant, i.e. the net is 'structurally bounded'.

4.4 Other Navigation Techniques

In Section 3 we discussed how browser-specific features should be translated into (virtual) links such that the description of the hyperdocument is captured in one uniform (platform-independent) manner.

An example of such features is the concept of *guided tours*. Guided tours are used in an attempt to solve the "lost in hyperspace" problem by giving predefined paths through the hyperdocument: readers do not have to make difficult choices where to go next. It is therefore straightforward how such tours can be expressed as sequences of links. Even the features that some systems offer for leaving guided tours (before the end) or for returning to the start can be simply simulated using (virtual) links.

History-based browsing is another example of a feature that some hypermedia systems offer to give the reader the possibility to just return on the path that is followed. One possibility is to just walk back on the path that has been followed. An other feature is to place bookmarks and be able to return to a node on which a bookmark has been set when that node was visited. Again, it is fairly simple to express these features using Petri nets. (Following a link makes the backward link available.) Similarly, some systems keep a list of nodes that have been visited (like a list containing a bookmark for every visited node), so the user can jump back to any previously visited node.

Figure 7 shows what needs to be added to the boolean Petri net structure in order to represent a feature that lets the user return to any previously visited node. The transition which activates a node also marks the node as being able to return to. From any node one can jump to the (global) history, and then reactivate any node in the history. This figure illustrates how easy it is to add the functionality of several navigational aids provided by hypertext systems to the Petri net representation[3].

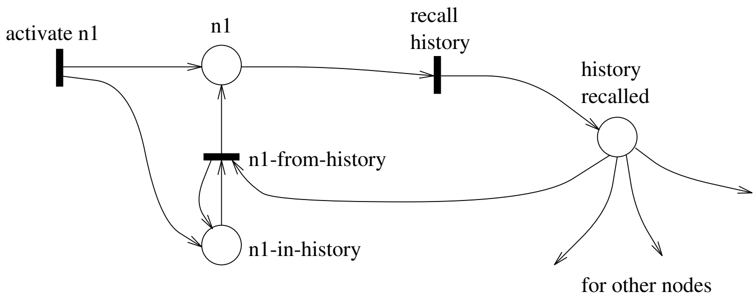

Figure 7: boolean Petri net representation of history based navigation

4.5 Analysis

One of the main advantages of using a Petri net based approach is the availability of powerful analysis methods.

A very powerful way to analyze a Petri net is the construction of the so-called *reachability graph*, also referred to as occurrence graph or coverability tree. The reachability graph can be used to answer a variety of questions: *Is every node reachable? Are there any links which cannot be followed? What is the maximum number of active nodes? Are there any dead ends?* Note that these questions are answered for a specific initial state of the hyperdocument. To reduce the size of the graph several reduction techniques have been developed ([8, 18]).

Given a Petri net it is possible to calculate *place* and *transition invariants* by using linear algebraic techniques ([10, 8]). These invariants can be used to investigate dependencies between nodes (e.g. either node n_1 is active of node n_2 is active but not both) and to detect loops. Place and transition invariants can be used to prove properties of the hyperdocument that are independent of the initial state.

Similar techniques can be used to calculate *traps* and *siphons* (deadlock) in a Petri net ([10]). A trap in the Petri net indicates that there is a set of nodes such that the number of active nodes in this set never decreases. A siphon may indicate nodes which become unreachable after a while (i.e. transient nodes).

Because of the close relationship between the standard Petri net and the hyperdocument it is easy to interpret these analysis results in terms of the hyperdocument. Moreover, the translation of the analysis results to hyperdocument properties can be automated and these results can be obtained using commonly available tools.

[3]The implementation of a complete history-list or backtracking mechanism, and some other popular navigation aids requires the ability to store node-id's in a stack. Although with some extensions to the Petri net formalism it is possible to emulate a stack using Petri nets, the resulting nets are too complicated to be used for analysis. Since most navigation problems originate in the structure of the hyperdocument, and not in these navigation tools, we omit this simulation in this paper.

The reduction of browsing in a hyperdocument in the Tower Model to a Petri net does not take into account the possible temporal constraints in hypermedia systems. Temporal constraints may be of different kinds:

- The information contained in a node must be displayed for some time before any link from any anchor connected to the node can be followed. This can be modeled by adding a timestamp to the tokens in the Petri net (see [18]). Such a timed Petri net can be used to describe various aspects of the temporal behavior of the hyperdocument.
- The anchors connected to a node may not all be accessible at the same time. Consider for instance an animation in which objects which are anchors appear and disappear. By turning the animation into a city object, assigning the anchors to the appropriate views, creating links that simulate the sequence of images in the animation and then assigning the appropriate timestamps to the tokens in the views, one can again use timed Petri nets to deal with these time constraints.
- Following a link may take time as well. This is simulated by making the transition add some extra time to the timestamps of the produced tokens.

It is also possible to use a colored Petri net ([8, 18]), i.e. a Petri net where each token has a specific value. This way it is possible to model multiple browsing sessions at the same time (see Stotts and Furuta [13]) or nodes having specific attributes like expiration dates, access control lists, etc.

Petri nets extended with 'time' and/or 'color' have been investigated by many authors (e.g. [8, 18]) and analysis tools for these nets have been built. Some critics of timed and colored Petri nets observe that since these nets can simulate a Turing machine, most interesting problems become undecidable. However, this is only true if the number of colors is infinite, which is usually the case. For describing the behavior of a hypertext we are not interested in the infinite behavior of the hypertext, but can limit the timeline to a day or so. Analyzing the behavior of a timed Petri net during it's first day or week is of course decidable.

5 Conclusions

We have described browsing in hyperdocuments by means of Petri nets: we have represented the hyperdocuments in the Tower Model and used standard Petri nets or timed Petri nets, depending on whether we have temporal constraints or not. With only few restrictions (mainly to enforce finiteness) the navigation through a hyperdocument has been described independently of the behavior of application-dependent information elements.

The translation from a hyperdocument to the corresponding Petri net is such that problems in the behavior of the Petri net can be immediately traced back to design errors in the hyperdocument itself (missing or undesirable links for instance, causing dead-ends and/or loops).

When describing the navigational semantics of a document we have included features that are typically available in the hypermedia system. Such features are not explicitly present in the stored documents. Examples are: the possibility to view more than one node at the same time; a "close" button for removing old nodes from the screen; buttons for returning to a start, table of contents, or index node; graphical interfaces that enable the user to zoom in on nodes from a composite, etc.

By basing the structure of the hyperdocuments on the very general Tower Model our description of the (explicit and implicit) browsing semantics of hyperdocuments is more powerful and more generally applicable than previous work on browsing.

Acknowledgement

We thank Wil van der Aalst from Eindhoven University of Technology for his help on part of the work on (boolean) Petri Nets.

References

[1] BROWN, P. A hypertext system for unix. *Computing Systems 2*, 1 (1989), 37–53.

[2] CAMPBELL, B., AND GOODMAN, J. HAM: a general purpose hypertext abstract machine. *Commun. ACM 31*, 7 (1988), 856–861.

[3] DE BRA, P., HOUBEN, G.-J., AND KORNATZKY, Y. An extensible data model for hyperdocuments. In *Proc. ACM Conf. on Hypertext'92* (Dec. 1992), pp. 222–231.

[4] ENGELBART, D. The augmented knowledge workshop. In *A History of Personal Workstations* (1988), Addison-Wesley, pp. 187–236.

[5] FURUTA, R., AND STOTTS, P. The Trellis hypertext reference model. In *Proc. of the Hypertext Standardization Workshop, National Institute of Standards* (Jan. 1990).

[6] HALASZ, F., AND SCHWARTZ, M. The Dexter hypertext reference model. In *Proc. of the Hypertext Standardization Workshop, National Institute of Standards* (Jan. 1990).

[7] HAREL, D. Statecharts: A visual formalism for complex systems. *Science of Comp. Programming 8* (June 1988), 231–274.

[8] JENSEN, K. *Coloured Petri Nets. Basic concepts, analysis methods and practical use.* EATCS monographs on Theoretical Computer Science. Springer-Verlag, 1992.

[9] LANGE, D. A formal model of hypertext. In *Proc. of the Hypertext Standardization Workshop, National Institute of Standards* (Jan. 1990).

[10] MURATA, T. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE 77* (1989), 541–580.

[11] RIZK, A., AND SAUTER, L. Multicard: An open hypermedia system. In *Proc. ACM Conf. on Hypertext'92* (Dec. 1992), pp. 4–10.

[12] SHNEIDERMAN, B. User interface design for the hyperties electronic encyclopedia. In *Proc. ACM Hypertext'87 Conf.* (Nov. 1987), pp. 189–194.

[13] STOTTS, P., AND FURUTA, R. Petri-net-based hypertext: Document structure with browsing semantics. *ACM Trans. on Information Systems 7*, 1 (1989), 3–29.

[14] STOTTS, P., FURUTA, R., AND RUIZ, J. Hyperdocuments as automata: Trace-based browsing property verification. In *Proc. ACM Conf. on Hypertext'92* (Dec. 1992), pp. 272–281.

[15] STREITZ, N., HAAKE, J., HANNEMANN, J., LEMKE, A., SCHULER, W., SCHÜTT, H., AND THÜRING, M. Sepia: a cooperative hypermedia authoring environment. In *Proc. ACM Conf. on Hypertext'92* (Dec. 1992), pp. 11–22.

[16] STREITZ, N., HANNEMANN, J., AND THURING, M. From ideas and arguments to hyperdocuments: Travelling through activity spaces. In *Proc. ACM Hypertext'89 Conf.* (1989), pp. 343–364.

[17] TOMPA, F. A data model for flexible hypertext database systems. *ACM Trans. on Information Systems 7*, 1 (Jan. 1989), 85–100.

[18] VAN DER AALST, W. *Timed coloured Petri nets and their Application to Logistics*. PhD thesis, Eindhoven Univ. of Technology, 1992.

[19] ZHENG, Y., AND PONG, M.-C. Using statecharts to model hypertext. In *Proc. ACM Conf. on Hypertext'92* (Dec. 1992), pp. 242–250.