ON THE LOGIC OF GENERALIZED HYPERTEXT

by

Michael P. Bieber
New Jersey Instiute of Technology
Department of Computer & Information Science
&
Stern School of Business
New York University
Information Systems Department

Steven O. Kimbrough
The University of Michigan
School of Business Administration

Center for Research on Information Systems
Information Systems Department
Stern School of Business
New York University

Working Paper Series

STERN IS-93-4

On the Logic of Generalized Hypertext*

Michael P. Bieber†
New Jersey Institute of Technology
Department of Computer & Information Science
University Heights
Newark, NJ 07102
(201) 596-2681
BIEBER@CIS.NJIT.EDU

Steven O. Kimbrough
The University of Michigan
School of Business Administration
Ann Arbor, MI 48109–1234
(313) 747-4460
KIMBROUGH@Wharton.upenn.edu

June 24, 1992

forthcoming in *Decision Support Systems*

*Special thanks to Hemant K. Bhargava and Christopher V. Jones for stimulating discussions, ideas, and comments on an earlier draft. Bhargava is largely responsible for the TEFA subsystem, mentioned in the body of the paper. This work was funded in part by the U.S. Coast Guard, under contract DTCG39-86-C-E92204 (formerly DTCG39–86–C–80348), Steven O. Kimbrough principal investigator.

†Current address: New York University, Information Systems Department, mbieber@stern.nyu.edu

Abstract

Hypertext is one of those neat ideas in computing that periodically burst upon the scene, quickly demonstrating their usefulness and gaining widespread acceptance. As interesting, useful and exciting as hypertext is, the concept has certain problems and limitations, many of which are widely recognized. In this paper we describe what we call *basic hypertext* and we present a logic model for it. Basic hypertext should be thought of as a rigorously-presented approximation of first-generation hypertext concepts. Following our discussion of basic hypertext, we present our concept of *generalized hypertext,* which is aimed at overcoming certain of the limitations of basic hypertext and which we have implemented in a DSS shell called Max. We then present a logic model for browsing in generalized hypertext.

Last modification: June 23, 1992. File: Logic-Mod-GHT-DSS

This paper is a revised and expanded version of [9].

1 Introduction

Hypertext is one of those neat ideas in computing that periodically—and with impressive regularity—burst upon the scene, quickly demonstrating their usefulness and gaining widespread acceptance. The concept of hypertext offers a "natural" and comprehensive means of navigating among the information and commands of a decision support system (DSS). Beyond navigation, hypertext offers such serendipitous features as user-declared linking, comments, backtracking to previously–viewed stages in the session, and so on. Although the idea of hypertext is not a new one [12], extensive research and development efforts, and product introductions, have been initiated only within the last six or seven years (see [14, 39, 45, 53], which are also good introductory reviews of the subject), with the notable exceptions of NLS [19] and FRESS [55].

The field of hypertext has produced several noteworthy "first generation" hypertext systems. These systems are conceptually extensions of the *basic hypertext* model we introduce in §2. They rely on a mainly static view of hypertext, in which the links in hyperdocuments are specified manually. We believe that a principled, well-structured approach to generating a hypertext

network dynamically (under program control) will be one of the major characteristics of the next generation of hypertext systems. This is the approach we take in our model of *generalized hypertext,* discussed in what follows (see [23] for an influential list of outstanding problems in hypertext).

We have conceived of, designed, implemented, and delivered to the U.S. Coast Guard a decision support system shell, which has a model management system and which makes extensive use of hypertext principles. The shell—called Max—has been delivered in the form of a particular decision support system—called Max Financial—that is currently in use by the Coast Guard. We have described various aspects of this software elsewhere [2, 4, 3, 34, 35]. Max is, in part, an implementation of our generalization of the (standard) concept of hypertext, which we call *generalized hypertext.* We have described elsewhere our concept of generalized hypertext and discussed its implementation in Max [10].

Our aim in this paper is to present and discuss a logic model (or, rather, a significant fragment of a logic model, as we explain in §6) for generalized hypertext. Because hypertext is in large part a user interface idea and because hypertext systems—including Max—are so heavily procedural, it is perhaps surprising that we should want to model hypertext with logic. Indeed, we shall only provide logic models for certain elements of hypertext. Further, the attendant logical inferences for our model of basic (ungeneralized) hypertext are quite trivial, so much so that an inferential, or logical, point of view here is not likely to be very interesting. What we think is interesting is our generalization of hypertext, in which logical inference plays an absolutely critical rôle, a rôle so critical that Max—as procedural as so much of it is—is written in Prolog.

We were motivated to generalize the concept of hypertext by, among other things, the need we saw for automatic linking and for more general operation upon application objects (often called *nodes* in the hypertext literature), given the context of a decision support system shell. (See [2, 5, 6, 7, 32] for further details.) The user interface concept we were seeking, and have achieved, is what we call *interactive documents,* in which the output of a DSS is a hypertext document that can be edited and queried. But that's another story. In developing our generalizations of hypertext, we found it materially useful to take the logical point of view. It helped with clarity; it helped with rigor; it was easily coded in Prolog. It even helped with suggesting generalizations, for if you can operationalize an idea by generalizing on a

particular predicate, then it is natural to ask about the meaning and usefulness of generalizing on other predicates [8, 33]. Thus, although the modeling that follows is incomplete and represents work in progress, we hope it will be useful to others both for what it has to say about hypertext and as an illustration of the benefits of modeling with logic.

Until recently there has been little published research regarding hypertext modeling. One exception is Garg's model fragment, which focuses on abstraction mechanisms in hypertext [22]. The other exception is the Hypertext Abstract Machine or HAM [13]. HAM models a low level transaction-based storage engine for components in a hypertext system.Several hypertext systems have been implemented on top of this engine (e.g., Neptune [16] and DynamicDesign [11]). HAM can be viewed as a model of a first generation or, as we shall explain later, a *basic* hypertext system. There are no inherent provisions for what Halasz refers to as "virtual structures" or "computation" [23], which are of paramount importance in our application domain and are at the heart of the model of *generalized hypertext* that we present in this paper. In January 1990 the National Institute of Standards sponsored a hypertext standardization workshop [43], at which several hypertext "reference" models that do allow for a dynamic structuring of a hypertext network were introduced (e.g., [21, 24]). The goal of these reference models is to define what is meant by the concept of hypertext and to provide frameworks for comparing the features of various hypertext systems. It is hoped that a hypertext interchange format will emerge permitting hypertext systems to share their data (see [44]).

While our model of generalized hypertext for the most part can be mapped to these *meta-level* reference models, our focus differs in two important ways. First, instead of mapping among hypertext systems, we are concerned with mapping non-hypertext information system applications to a hypertext interface for the reasons expressed earlier. Here, our major contribution is the technique of bridge laws (see §6), which model this integration. Second, in the existing models of hypertext virtual structures and computation are *ad hoc*, as are mechanisms most current systems use to implement them (e.g., [51]). Because most of our applications are dynamic in nature (e.g., DSS which execute decision models on the fly), we provide a structure modeling hypertext interaction, which can only be instantiated at run-time. Virtual structures and filtered computation [21]—defined by bridge laws—both are at the core of our hypertext model. Our system-level hypertext engine per-

forms inference for every hypertext browsing command in a principled, well structured manner.

The strategy and organization for the remainder of the paper is this. We assume the reader is familiar with hypertext and first-order logic. In §2 we develop a particular concept of hypertext, which we call *basic hypertext.* We do not claim that basic hypertext is an adequate representation of all or most hypertext systems. Rather, we claim that it is reasonably close to, and very much in the spirit of, the mainstream of first-generation hypertext ideas. Our purpose in presenting basic hypertext is to model it, which we do in §3, and then to model our generalization of it, in §6. Our model is based on the concept of *bridge laws,* introduced in §5. We summarize and conclude in §7.

2 Basic Hypertext: Concepts

A hypertext network—often called a *hyperdocument* [14] or simply a *hypertext* [24, 21, 38]—consists of an arbitrary number of interrelated *nodes, links,* and *buttons. Nodes* are objects that are declared in a data base (of some sort) and, when displayed, are represented as text on the screen.[1] *Links,* which describe relationships between pairs of nodes (called the source and sink *link endpoints*), are also declared in a data base. Embedded in nodes—as part of the declarations establishing the nodes—are *link anchors* or *buttons,* which indicate the presence of a link and which usually are highlighted in some manner when the node embedding them is displayed. The number of buttons at a node, and the number of links in a document, is essentially unlimited.

A hypertext system is software for creating, editing and maintaining hyperdocuments and—more interestingly and to the point for present purposes—for *browsing* through, or exploring, hyperdocuments. The browsing concept is central to the hypertext idea [14, 39, 46], and it is the hypertext browsing functions that are the subject of this paper and of our logic models. Our generalization of hypertext generalizes these browsing functions. A user browses a hyperdocument by viewing a displayed hypertext node, and *selecting* a link to another node. Typically this is achieved by pointing with a mouse to the

[1] Or, for hypermedia systems, represented as other types of information objects, such as pictures, videos, sounds, and animations.

button highlighted in the node's text that names a link emanating from that node, and by clicking with the mouse. Selecting a link causes the system to *traverse* the link, i.e., to determine which node is at the link's distal endpoint, here the sink. In basic hypertext (as we are characterizing the concept), once a link is traversed the destination node is displayed. Clicking on a button, then, produces a new display of a node.

In basic hypertext, there are other browsing commands than the essential and fundamental *select-and-traverse-to-sink* command just discussed. First, browsers may a select a button and request display of the node linked as a *source,* or incoming, link endpoint. Thus, to implement bi-directional linking, there is a *select-and-traverse-to-source* browsing command in basic hypertext.[2] Second, both links and nodes may have attributes, which for simplicity we model as semantic types. For example, in a hypertext system to support argumentation, a node may contain a body of text, T, with buttons representing links of semantic types "supports" and "supported by." The latter names a link whose source node contains text that tends to support the assertions in T, while the former names a link whose sink node contains text supported by T. Further, the nodes may have information associated with them that can be retrieved upon command by the browser. For example, in an argumentation system [15, 40, 54] it would be useful to know the type of information contained in a given node. Thus, we have two further browsing commands: *select-and-display-link-attribute* and *select-and-display-node-attribute.*

A fourth category of hypertext navigation is backtracking, or returning to the previous node. Hypertext navigation takes place in two directions—forwards to "destinations" related to a selected object (button) and backwards by backtracking along the "path" of documents the user already traversed during the session. Backtracking allows users to explore within an information system without fear of getting "lost." Users can "detour" from their primary task by looking at items that "catch their eye" with confidence, because they know they can always return to a familiar part of the information system to re-orient themselves.

[2]We specify a directionality in links to maintain semantic interpretations (embodied in the link's semantic type attribute). To this purpose we nominally refer to a given link endpoint as being a "source" or "sink" node. During link traversal we shall refer to the node containing the button selected by the user for traversal as the link's "origin" and the distal endpoint as the link "destination."

Finally, many hypertext systems support some version of procedural attachment (or "action links" [38]), in which links are associated with procedures for enhancing the display of one of their endpoint nodes. In such a case, traversing a link results in execution of a procedure. Normally—and always for basic hypertext as we are characterizing it—attached procedures either format results that are placed in nodes or they affect how certain nodes are displayed.

In summary, then, the browsing commands for basic hypertext—as we are defining it—are as follows.

1. *select-and-traverse-to-sink*
2. *select-and-traverse-to-source*
3. *select-and-display-link-attribute*
4. *select-and-display-node-attribute*
5. *backtrack*

Figure 1: Basic Hypertext Browsing Commands

Clearly, these are browsing commands only for a very basic hypertext system. Still, we believe they adequately capture the spirit of most existing systems, whose many features can be interpreted as fairly direct elaborations on our basic (i.e., non-generalized) model.

We now turn to the task of developing a logic model for basic hypertext.

3 Basic Hypertext: Conceptual and Logic Models

Every modeling effort inevitably requires judicious selection of which elements of the object system are to be modeled. The present effort is no exception. In general, there are two basic, complementary approaches to deciding what to leave out of a model. One can abstract away, or simply ignore, certain features of a system, and one can focus on certain parts of

an object system, to the exclusion of others. Our selection strategy for the models to follow can best be described as a focusing strategy.

Hypertext systems in general have significant elements of procedurality (e.g., displaying a node, handling a mouse click) and of inference (e.g., traversing a link, searching for a node with a given attribute). Without putting too much weight on the distinction, it is natural to think of hypertext systems as both thinking (making inferences) and acting (doing things with procedures). Further, it is natural to model the thinking, or inferential, aspects of any system with logic, and (often) not very natural to model the actions of a system with logic. Most importantly for the present context, generalized hypertext, which we are offering as a contribution to the concept hypertext (and its implementation), differs from existing hypertext systems in the greatly-expanded rôle of inferencing in the system.

It is natural, then, to model the inferential aspects of hypertext with logic, and it is appropriate to explain and contrast basic and generalized hypertext by focusing on their inferential differences. That is the aim of the logic modeling exercise we present in what follows. Before presenting the logic model for basic hypertext, however, we need a conceptual framework for hypertext systems that clearly identifies which elements are to be modeled logically. To that now.

Much as Lisp can be described as operating with a basic *read-evaluate-print* loop, so hypertext systems can conveniently be described as operating under a basic *read-evaluate-update-display* loop. Focusing (without loss of generality) on the browsing commands, the operation of a hypertext system proceeds by reading a (browsing) command from the user, evaluating the command, performing any updates to the system, and displaying the appropriate result. Our focus is on the inferential aspects of the evaluation step in the hypertext control loop, in particular on the inferences used to support the browsing commands.

We begin explaining our logic model by presenting and discussing the language—the first-order predicates and terms—used to describe a hyperdocument. The essential predicates and their intended interpretations are as follows.

node(x,y,z) x is a node with content expression y and semantic type (attribute) z.

link(u,v,w,x,y,z) u is a link from source node v to sink node w with semantic

type x, operation type y, and either display mode or procedure identifier z.

button(x,y,z) x is a button representing link y with content expression z.

sessionlog([[v,w,x,z,y], ...]) The session log contains a chronological (last-in-first-out) list of identically formatted list elements, in which v is the command executed, w the button selected in originating node x and y is the link traversed to the destination node z.

evaluate(u,v,w,x,y,z) u is a procedure that formats the content of expression v of a desination node, taking into account the link semantic type w and the destination node semantic type x, producing text y to be displayed under display mode or procedure identifier z.

startingnode(x,y,z) x is the default node to display upon system initialization, using link operation type y and link display mode z.

For terms, we shall often use integers for node, link, and button names. In addition, we have the following logical names (and their intended interpretations).

issue	semantic type for a node
position	semantic type for a node
argument	semantic type for a node
procedure	operation type for a link
display	operation type for a link
supports	semantic type for a link
supported_by	semantic type for a link
more_information	semantic type for a link
full_window	a link display mode
pop_up_window	a link display mode
[]	nil, falsehood
true	truth

We also have the following functions, and their intended interpretations.

button(x)	maps to *true* if x is a button
symbol(x,y)	maps to *true* if x is a button

with display text y

Finally, we need to characterize the structure for representing text at a node. The particular structure we shall use is not essential for the general points we wish to make. We represent text (in the hyperdocument declarations) as a list of terms of two kinds: (a) names, to be represented as alphanumeric sequences in single quotes; (b) functions of the form *button(x)*, where x is a button identifier. We shall exploit Prolog list notation for convenience, but of course logically a list is either a special name, *[]*, or is a special function of arity two.

To see how all this works, first recall that the hypertext system is running under a *read-evaluate-update-display* loop. The *read* function is performed by a user interface system. Its job is to get input from the user, to translate this input into a form that can be understood by the hypertext evaluator, and to call the evaluator. The evaluator performs appropriate inferences, extracts certain parameter values, and passes control to the updating procedure, which should be thought of as having the function of altering declarations in the data base constituting the hyperdocument in question. Once the update procedure is complete, the user interface is called with new information for display, and the loop recycles.

For some specifics of how this works, let us examine an example that is as simple as possible for illustrating the browsing commands of basic hypertext.

Example: Hyperdocument 1

Hyperdocument 1 is about as simple as can be. It has two nodes and one link. Its declarations are as follows.

node(1,['If the','Soviet Union', 'is to be competitive,', 'we must hit the inside curveball.', '—', button(1)],[])
node(2,['Quotation of the Day','The New York Times','August 16, 1989'],[])
link(1,1,2,more_information,display,full_window)
button(1,1,'Aleksei L. Nikolov')
sessionlog([])
startingnode(1,display,full_window)

First, we have to get a node displayed on the screen. Assume that the default node on startup is node 1. We use its predicates to make the appro-

priate inferences for the user interface system. There are two basic tasks at hand. The first is to determine the mode of display for the node. This is easy. We simply deduce values for x, y and z that satisfy $\exists x, y, z\, startingnode(x,y,z)$. Trivially, given the declarations in the present example, the default starting node is *1,* and it should be *display*ed in a *full_window.* Less trivially, but still easily, the declared text must be transformed for display by the user interface system. In particular, *button(1)* in the text for node 1 must be replaced with a symbol that both indicates what should be displayed and provides sufficient information to identify the symbol as a button. This can be done by replacing every occurrence of *button(x)* in a text with *symbol(x,y)* in accordance with the following laws.

$$\forall x, y\, (buttondisplay(button(x),\ symbol(x,y)) \rightarrow button(x,\_,y)) \qquad (1)$$

(We note that for the sake of notational perspicuity we are using the underscore as in Prolog's anonymous variable. No logical generality is lost by this, for an equivalent formula can always be had by replacing each anonymous variable with a unique variable and universally quantifying over the entire formula. For example the last formula above is equivalent to:

$$\forall x, y, z\, (buttondisplay(button(x),\ symbol(x,y)) \rightarrow button(x,z,y)) \qquad (2)$$

Also, we note that here and in what follows all clauses are Horn.)

Thus, the display value of the button is inferred from the hyperdocument declarations, substituted into the text representation, and the modified representation is sent to the (procedural) user interface system, along with the display mode symbol *full_window.* We assume an appropriate collection of declarations to do this, called *make_display_text(x,y),* in which x is the input text representation, as declared in the hyperdocument, and y is the output text representation, used by the user interface system to create the actual display.

Once node 1 is displayed on the screen, the system can accept and process browsing commands. In order to process browsing commands, the hypertext evaluator subsystem needs to be told what command is to be executed, what its various input parameter values are, and what the current node is. In return, the evaluator needs to tell the user interface subsystem what should be displayed, how it should be displayed, and what node to make current. Because we distinguish the essentially inferential command processor from

the procedural system update module, the command processor or evaluator must determine what needs updating. In sum, the evaluator needs to deduce values of w, x, y, and z that satisfy

$$process\_command(\langle c\rangle, \langle p\rangle, \langle n\rangle, w, x, y, z)$$

where c is a given command, p the parameter value(s) for the command, n (the current node), w (the list of update items), x (the text to be displayed), y (the mode of display), and z (the node to be made current). (Under the intended interpretation and use, the parameters in angle brackets are instantiated on input, while the other parameters are instantiated at output.)

So, now we can define this central predicate for the basic hypertext browsing commands outlined in the previous section. The first represents the *select-and-traverse-to-sink* operation. (Note: here and in what follows we use $\phi \leftarrow \psi$ for reverse material implication, i.e., as equivalent to $\psi \rightarrow \phi$.)

$$\begin{aligned}
&\forall t, u, v, w, x, y, z \qquad (3)\\
&(process_command(traverse_sink,w,v,\\
&[update_log(traverse_sink,w,v,t,z)],x,y,z) \leftarrow\\
&\qquad (button(w,t,_) \quad \wedge\\
&\qquad link(t,v,z,_,\ display,\ y) \quad \wedge\\
&\qquad node(z,u,_) \quad \wedge\\
&\qquad make_display_text(u,x))\ )
\end{aligned}$$

Paraphrased into something English-like, formula 3 says that we traverse to the sink node, z of link t—outgoing from node v and named by button w—whose text x we are to display in mode y *if* (w is a button naming the outgoing link, t, that leads from the present node, v, to node z, the text of which should be displayed in display mode y, and the text u at node z is transformed for display to x). Using hyperdocument 1 as an example, in figure 2 the user has selected the button in the source node ("Node 1") and traversed its link to the sink node ("Node 2"), which is displayed in the topmost window on the screen, indicating that it is the currently active node.

Put figure 2 about here.

There are four more browsing commands to model for basic hypertext (recall figure 1), but the pattern evident in formula 3 continues.

The second command, traversing to the source node for a button, provides for bi-directional linking. The logic model is as follows.

$$
\begin{aligned}
&\forall t, u, v, w, x, y, z \qquad (4)\\
&(process\_command(traverse\_source, w, v,\\
&[update\_log(traverse\_source, w, v, t, z)], x, y, z) \leftarrow\\
&\quad (button(w, t, \_) \quad \wedge\\
&\quad link(t, z, v, \_, display, y) \quad \wedge\\
&\quad node(z, u, \_) \quad \wedge\\
&\quad make\_display\_text(u, x)))
\end{aligned}
$$

Other than changing the name of the command, in the first line of formula 4, the only difference between (1) and (2) is in *link(t,z,v,_, display, y)*, in which we have reversed the source and sink node identifiers.

Our third browsing command is for displaying a link attribute for the link named by button *w*. In our basic model the link's sole attribute is its semantic type, e.g., *supports* or *supported_by,* for an argumentation application. A user might request the semantic type in order to preview the consequences of committing to a link traversal [37]. The model for this command is as follows.

$$
\begin{aligned}
&\forall t, u, v, w, x, y \qquad (5)\\
&(process\_command(display\_link\_attr, w, v,\\
&[], x, pop\_up\_window, v) \leftarrow\\
&\quad (button(w, t, \_) \quad \wedge\\
&\quad link(t, v, \_, u, \_, \_) \quad \wedge\\
&\quad make\_attribute\_display\_text(u, x)))
\end{aligned}
$$

Here, we have added a new predicate, *make_attribute_display_text,* that transforms an attribute expression into an expression displayable by the user interface system. Note also that we have assumed that link attribute information is always presented in a pop_up window.

The model for our fourth command, display of a node attribute (its semantic type, e.g., *issue, position* or *argument*), is quite similar to that for display of a link attribute.

$$\begin{aligned}
&\forall t, u, v, w, x, y \qquad (6)\\
&(process_command(display_node_attr, w, v,\\
&[], x, pop_up_window, v) \leftarrow\\
&\qquad (button(w, t, _) \quad \wedge\\
&\qquad link(t, v, z, _, _, _) \quad \wedge\\
&\qquad node(z, _, u) \quad \wedge\\
&\quad make_attribute_display_text(u, x)))
\end{aligned}$$

Our fifth command for basic hypertext is *backtrack.* We will not present a logic model for it here, since it is straightforward and does not affect our generalizations. (See [5] for an extended logical discussion of this matter.)

A word about procedural attachment, beginning with an example. Suppose that in Hyperdocument 1, the current text of *node 2* were instead the text *'this is a citation from', button(2)]*, and that *button(2)*'s text were what is currently in *node(2).* Under the assumption that the first button in a node's text provides a brief summary of the node's meaning, we could declare a procedure named *first_button* that displays just the text of the first button found in the destination node's content expression. In this scenario, traversing the new link *link(2, 1, 2, summary, procedure, first_button)* would also result in Figure 2.

Procedural attachment can be modeled with an additional clause for the *traverse-sink* command, formula 3. Consider the following

$$\begin{aligned}
&\forall t, u, v, w, x, y, z \qquad (7)\\
&(process_command(traverse_sink, w, v,\\
&[update_sink, w, v, t, z], x, y, z) \leftarrow\\
&\qquad (button(w, t, _) \quad \wedge\\
&\quad link(t, v, z, \ a, \ procedure, \ u) \quad \wedge\\
&\qquad node(z, r, b) \quad \wedge\\
&\quad evaluate(u, r, a, b, x, y)))
\end{aligned}$$

in which w is the source button the user selected, v is the source node containing it, t is the button's link, z is the destination node, a and b are the link

and node semantic types that are used in determining the procedure results, u is the attached procedure, x is the destination node's display text and y is the display mode for the destination node.

There are several things to note about formula 7. First, we have assumed, for the sake of modeling, the existence of the *evaluate/6* predicate, which executes a procedure. In an implementation, it may or may not be sensible to treat *evaluate/6* in a purely logical fashion. If not, then the code that implements it is straightforwardly a procedural attachment. Second, although expression 7 does not permit parameters to be passed to the procedure that is evaluated, this constraint could easily be relaxed. We are trying to keep things as simple as possible, in order not to obscure the main points of the paper.

Finally, we note two facts. First, although our conceptual framework for basic hypertext has identified an element of inferencing, described above, the amount of inferencing—its depth and subtlety—in basic hypertext is quite limited. Second, in our basic hypertext system and in most first generation hypertext systems, every node and link in a hyperdocument must be explicitly declared by the author(s) of the document. The clauses describing a hyperdocument for basic hypertext, e.g., Hyperdocument 1, are all ground; they fail to exploit quantification. Further, they are all logically simple; no logical constants are used. Of course, real hypertext systems come with editors that materially help the authors in making the required explicit declarations. Nevertheless, it would be desireable if the expressions used to declare a hyperdocument could be understood as creating the elements of a hyperdocument—e.g., nodes and links—implicitly as well as explicitly. This is imperative for decision support systems. DSS applications tend to be large, making explicit enumeration of its elements impractical. Also, DSS applications are dynamic—execution reports that map to destination nodes are generated in real time, not in advance. Most importantly, we want the DSS shell to map hypertext to the DSS elements on behalf of the application builder instead of forcing builders to reclassify their application elements in terms of hypertext nodes, links and buttons.

4 The Need to Move beyond Basic Hypertext

However interesting the basic hypertext concept, and however useful various implementations of it have proved to be, a number of problems and limitations have been identified with this basic concept [2, 14, 20, 23, 56]. For present purposes we are concerned with the following widely-recognized problems and limitations in basic hypertext (and in current implementations of hypertext).[3]

- **Manual linking [2, 17, 20, 23, 30, 56].** Basic hypertext systems provide editing features for linking existing nodes, and for creating and manipulating buttons. These features are highly useful to the builder—or annotator—of a hyperdocument. The basic hypertext concept, however, does not encompass virtual, or inferred, linking of nodes by the system at run time. To illustrate the inferred linking concept (called *implicit linking* by DeRose [17]), consider a system with predefined keyword nodes, whose contents explain and discuss the keyword in question. The hypertext system might infer a link (and thus the existence of an accompanying button) by being able to recognize keywords in arbitrary nodes and by dynamically creating buttons out of them that are linked to the appropriate keyword nodes. With such a capability, a builder could simply type text into a node and have the system create many of the needed buttons and links associated with that node.[4] Clearly, there is considerable potential benefit—especially in terms of reducing the cost of building a hyperdocument—to having the hypertext system capable of creating buttons and links automatically.

- **Manual node creation [2, 23, 47].** This is the node version of the above link limitation. Under the basic hypertext concept, the hyperdocument builder builds nodes by using an editor to key in or to

[3] This section is drawn from [10].

[4] Although we will not discuss it further, this feature is supported in the system we discuss in [10]. Other researchers have been active in exploring this sort of feature, e.g., in the context of extended electronic mail systems [1, 26, 50]. Other researchers are working on generating links from content analysis on text [25, 48].

paste in information. There remains the possibility of the hypertext system generating nodes (along with embedded buttons) on the basis of user inputs, in conjunction with existing information in its data base [21, 41, 51]. (See [20] for mention of work on generation of graphical objects.) Again, it can be hoped that with such a capability the cost of developing a hyperdocument might be reduced. Finally, we note that automatic creation of nodes is quite different from procedural attachment (see above), which has been used to modify—or modify the display of—nodes, rather than to create them.

- **Multiple views [23, 36, 49].** Basic hypertext systems typically provide a limited number of ways to view nodes. For example, many systems permit buttons to be displayed with or without highlighting, and some offer both a user's view and a builder's view for nodes. Other views, not envisioned in basic hypertext, are possible. As a means of reducing cognitive overhead, nodes might be filtered and transformed for pertinent information before display. For example, displays specialized by type of user (novice, experienced, e.g.) might be implemented in this way.

- **Cost of building hyperdocuments [2, 34, 35].** Basic hypertext systems provide substantial support for building applications in which the user may interactively explore a large collection of associated information. Nevertheless, much more might be achieved by embedding knowledge into the hypertext system [18]. For example, contextual information could be used automatically to invoke filtering routines in support of multiple views of nodes.

With the basic hypertext concept and a list of some of its limitations at hand, we can now discuss our generalization of the concept. Our focus here is on the underlying logic of generalized hypertext (see [10] for an informal discussion of generalized hypertext). We discuss that logic in §6. First, however, we turn to a brief discussion of an essential, foundational idea for the logic of generalized hypertext.

5 Bridge Laws

The problems and limitations of basic hypertext, at least those listed in §4, can be thought of as setting a requirement that ways be found for programs (rather than people) to create links, create nodes, and impose views in hyperdocuments. This achieved, then plausibly the cost of building and maintaining hyperdocuments can be substantially reduced [10]. We find the principle of bridge laws instrumental for developing programs that can produce hypertext systems while minimizing human effort in developing those systems. Our purpose in this section is briefly to introduce the concept of bridge laws. In §6 we will apply the concept extensively.

The term *bridge law* comes from the philosophy of science literature on inter-theoretic reduction (e.g., [31, 42]). It would seem that some theories can be, or have been, explained by other, more fundamental theories. For example, it is plausible that the gas laws can be explained by the more fundamental theories of thermodynamics and statistical mechanics. More broadly, we often think of chemistry as explainable by physics plus initial conditions, biology as explainable by chemistry plus initial conditions, and so on. If one theory or science (called the explanandum) has been explained by another theory or theories (called the explanans), we say it has been reduced.

What is the logical relation between an explanandum that has been reduced by an explanans? We would like to think of the relation as a deductive one: the explandum is deduced from the explanans. Typically, however, the vocabularies and languages of the two theories are quite different. Physics, for example, talks about fundamental forces and weird atomic particles. The stuff of chemistry—chemical reactions, valences, elements, etc.—is simply not in the vocabulary of physics. So how can the laws of chemistry possibly be deduced from the laws of physics if they do not share a common vocabulary? They can't. If chemistry is to be deduced from physics—and more generally if a statement in one domain of discourse is to be derived by expressions in another domain of discourse—then there must be some way of mapping expressions in one domain to expressions in the other. Such domain-spanning expressions are called bridge laws in the literature on theory reduction in science.

Whether there are any bridge laws in science, and if so what they are, is a controversial matter in philosophy. That need not concern us here. What

matters is the idea, and the idea is simple and very useful.

> **Bridge laws.** A bridge law is a universally quantified material conditional in which the expressions in the antededent are all in one language (or domain of discourse) and the expressions in the consequent are all in another language (or domain of discourse).

In the present case, we should think of a hypertext system as working by making inferences in the hypertext domain of discouse. The underlying hypertext language, as we have seen illustrated in §2, is designed for talking about nodes, links, buttons, and so on. An application, for which a hypertext system is a front-end, can be thought of as making inferences in some rather different domain. In our implementation, Max, that domain is mathematical models. There, the application software makes inferences in a language that describes models, variables, data, and so on.

In what follows, we shall see how bridge laws may effectively bridge (Dare we say link?) two domains of discourse. How this linking is done is not particularly specific to the applications at hand. The bridge law method is general. A system builder, in adding a hypertext front-end to a given application, can be thought of as using bridge laws to tell the hypertext system what in general is important in the attached application. We now turn to a discussion of the logic of generalized hypertext, a logic in which bridge laws play an important part.

6 Generalized Hypertext: Concept and Model

Our concept of generalized hypertext is basic hypertext plus generalizations with regard to nodes, links, buttons and link traversal. These generalizations are further extended by system-level support for contextual dependencies pertaining to users and to application domains. The aim of the present section is to present, discuss and model these generalizations (see [5, 10] for further discussion).

6.1 Contexts

The purpose of *contexts* (also called *filters*) is to set or remove certain default assumptions.[5] For example, if the applicable context is that for naïve users, then the user interface system may simplify the display in certain ways, perhaps by eliminating acronyms in favor of their full expressions by hiding technical information in documents, or by pruning the number of button or commands. Logically, we represent contexts by adding an extra argument to our basic predicates. What was, for example, simply a node, becomes a node in a given context. Also, because contexts are individuated it is possible to reason about them. We do so in two main ways.

First, contexts may be used for mapping (transforming) and filtering data objects in the system. For example, in our present system, contexts may be set in which only certain entities (documents, commands, or models) are available. The entities that are available are determined during run time by mapping certain information to a set of possible entities, then filtering the set using contextual information (see expression 14 in §6.3, below). Second, contexts may be declared in much the way nodes, links, and buttons are declared, and in this way become generalized hypertext objects. We model contexts with the predicate *ghtcontext(c, a)* with *c* being the context identifier and *a* its attributes, e.g., a definition, its semantic type or the application which declared it. (As a mnemonic we use *ght* as a prefix in many generalized hypertext predicates.) More than one context may be active simultaneously. As we shall see later, each generalized hypertext node, link and button has a context argument. During link traversal (see §6.6) the value for that argument must be an active context identifier for its instance to be accessed. There is a special context symbol *all*, which indicates an instance is accessible under all context settings. The hypertext evaluator maintains the set of currently active contexts in the predicate *ghtactivecontexts(l)* with *l* being a list of context identifiers. *l* always includes the symbol *all*. A more sophisticated model of contexts can be found in [5]. One aspect of this model is the organization of contexts into inheritance hierarchies representing different characteristics of an application's environment and user abilities.

In what follows, we frequently mention context and contextual inferences, and normally use the variable, *c*, as a context variable.

[5]This differs from the "context" objects in HAM [13], which serve to partition, rather than tailor, the hyperdocument.

6.2 Node Generalization

In basic hypertext, nodes are normally collections of text with embedded buttons and (in many hypertext systems) graphics. Further, as we have seen, these nodes are represented explicitly in the system. We generalize nodes in two principal ways. First, while nodes may be collections of text with embedded buttons, under our concept (when idealized) a node may be any information item about which the system may reason. Abstractly, nodes are objects that may be named (referred to) in various ways (explicitly or implicitly), and linked to other nodes. Just about any sort of entity may be the endpoint of a link (including contexts and other links), and all such endpoints are considered to be nodes. Further, information about nodes may be declared in the system, and this information (including contextual information) may be used by the system during its link traversal operations.

Our second generalization, then, is that nodes need not be explicitly represented in the system. They may be inferred at run time from declarations used to build the system, as well as from other information, such as context. Logically, and in our system operationally, what this means is that nodes may be declared with formulas having quantified variables. (This implements what Halasz calls *virtual structures* [23].) Recall that in example 1 the node declarations, viz.

node(1,['If the','Soviet Union', 'is to be competitive,', 'we must hit the inside curveball.', '—', button(1)],[])
node(2,['Quotation of the Day','The New York Times', 'August 16, 1989'],[])

were all logically simple ground expressions. The natural generalization, from a logical point of view, is to exploit quantification in order to define nodes implicitly for the system. This is exactly what we do, using the predicate *ghtnode(x,y,z),* with x the node identifier, y the node attributes, and z the context. For example, nodes 1 and 2 could be inferred as instances of *node/4* with the following law, where *s-type* and *n-type* signify semantic and node types respectively.

$$\forall x, y, z \qquad (8)$$
ghtnode(x,[[contents,y],[s-type,z],
[n-type,explicit]],basic_hypertext) $\leftarrow$

node(x,y,z)

Given this move, this abstraction, we can declare laws that make just about anything a node. Links can be nodes. (This implements what Lange calls second order links [38].)

$$\forall u, v, w, x, y, z \tag{9}$$
(ghtnode(u,[[source,v], [sink,w], [s-type,x],[n-type,link],
[operation-type,y],[display-mode,z]],basic_hypertext) $\leftarrow$
link(u,v,w,x,y,z))

Buttons can be nodes.

$$\forall x, y, z \tag{10}$$
(ghtnode(x,[[link,y],[contents,z]],basic_hypertext) $\leftarrow$
button(x,y,z))

More interestingly, the hypertext system can be mapped to an application and laws can be declared that make application-specific entities into generalized hypertext nodes. In our system, Max Financial, the generalized hypertext system is matched to a model management system. Nodes may be declared with bridge laws (see §5) that map between predicates in the two systems. For example, every model in the model management system (called TEFA [4]) is a node in the generalized hypertext system (called Maxi [10, 34]). The following formula is a bridge law that achieves this.

$$\forall c, v, w, x, y, z \tag{11}$$
(ghtnode(x,[[math-ex,z],[source,w],[description,v],
[n-type, model],[model-type,y]],c) $\leftarrow$
model(x,y,z)) $\wedge$
context(c,x,model) $\wedge$
source(x,w) $\wedge$
description(x,v)))

Note that *ghtnode/3* is a predicate from the generalized hypertext system, while the other four predicates in formula 11 belong to the model management system and are used to declare models. In short, such bridge laws can

be added at will, and what counts as a node can be extended more or less indefinitely. Individual bridge laws can be very general (as in formula 11), or they can refer to particular objects explicitly. Our goal is for a system builder to be able to combine existing application predicates and hypertext utility predicates to declare a small set of bridge laws that can map the components of all applications in the system.

6.3 Link Generalization

In basic hypertext, each link establishes a relation between a single source node and a single sink node, called the link endpoints. We generalize links in two principal ways. First, links may be "n-ary" [24]: they may fork into multiple links. Thus, in selecting a button, which names a link, the user may then be asked to choose among several sub-links. We call such collections of sub-links *link ensembles.* For example, in §6.4 we shall see that the name of a mathematical model may be a button in a document. Upon selecting such a button, the link traversal routine will infer that several options are presently available, e.g., to run the model, to describe the model, and to suggest a scenario (data set) for running the model (see figure 3). Link ensembles often represent different aspects of the "object of interest" represented by the button the user selects.

Put figure 3 about here.

Each of these sub-links, or link forks, is traversable by the system and may be thought of as a DSS application command. In basic hypertext each link may be thought of as a command to display one of the two endpoints of a link. This generalization allows arbitrary commands for operating upon a link endpoint, and so is a richer concept than what is normally meant by procedural attachment.

Our second generalization is that links need not be explicitly represented in the system. Like nodes, they may be inferred at run time from logically quantified declarations used to build the system, as well as from other information, such as user inputs and context. In fact, generalized hypertext buttons will often indicate the presence of such virtual links. These links are not generated until the user actually chooses to traverse them.

In order to see the logical import of these generalizations, recall our link

declaration from example 1—*link(1,1,2,more_information, display, full_window)*—which is interpreted as "Link 1 is of type *more_information* and links source node 1 with sink node 2." Our link predicate for generalized hypertext declares link ensembles, where each element in the ensemble is a command that can be used (perhaps with other information) to operate upon the link's object of interest. Traversing a link that corresponds to an application command causes the application to execute that command. Often this results in the generation of a report, which a bridge law can map to the link's desination node. Thus, we have the generalized link predicate *ghtlink(u, v, w, x, y, z, c)* for which the intended interpretation is that u is the link identifier, v is the source node, w the sink node, x the link's attributes, y the display mode of the destination node, z the ensemble elements, or commands, associated with the link, and c is as usual the pertinent contextual information. Adverting to example 1, we can define our basic hypertext link as a generalized hypertext link, with the following laws.

$$\forall u, v, w, x, y \tag{12}$$
$$(ghtlink(u,v,w,\ [[s\text{-}type,x]],y,[display],basic\_hypertext) \leftarrow$$
$$link(u,v,w,x,display,y))\ )$$

$$\forall u, v, w, x, y \tag{13}$$
$$(ghtlink(u,v,w,\ [[s\text{-}type,x]],\_,[y],basic\_hypertext) \leftarrow$$
$$link(u,v,w,x,procedure,y))\ )$$

Formula 12 is for displaying text and 13 is for executing attached procedures.

Although this example illustrates both of our link generalizations—link ensembles and implicit declaration of links—a more interesting example is provided by connecting the generalized hypertext system with an application. Again, we will illustrate this with a bridge law between Maxi and TEFA. The following law describes the generalized hypertext link ensemble associated with every model in Max Financial.

$$\forall c, u, v, w, z \tag{14}$$
$$(ghtlink(u,v,w,[[l\text{-}type,model],[owner,TEFA]],\_,z,c) \leftarrow$$

(model(u,_,_) $\wedge$
available_commands(u,z,c)))

There are several things to note about expression 14. First, *ghtlink/7* is a predicate in the (generalized) hypertext system, while *model/3* and *available_commands/3* are predicates in the application system, here the model management system, TEFA. Second, the pertinent originating and destination nodes v and w for for formula 14-style links are not specified, here. These nodes may exist only virtually. Until there is a clear need to determine them, they continue to be only implicit. Third, calling either or both of the TEFA predicates may result in a large amount of inferencing. For example, TEFA is normally able to execute any of three commands on a given model: *run, describe,* and *suggest_scenario* (cf., figure 3). That it is able to do so, however, is something that is inferred by TEFA depending on the current context. Also, the list of commands returned by TEFA, z, may be fairly complex. Typically, it has the following sort of form: *[symbol(TEFA, run), symbol(TEFA, describe), symbol(TEFA, suggest_scenario)].* (Note: Here *symbol(x,y)* is a function with the intended interpretation that it maps to *true* if y is a symbol from the x subsystem.) Expressing the link ensemble this way allows the hypertext system and the user interface to manage *run, describe* and *suggest_scenario,* on the screen and inferentially, in a knowledge base.

6.4 Generalizing Buttons

We generalize buttons in much the same way we generalized nodes and links: by declaring expressions that use the predicate in question but that are quantified and are logically complex. Recall our button declaration from example 1.

button(1,1,'Aleksei L. Nikolov')

Expressed as a generalized hypertext button, we have the following law.

$\forall x, y, z$ (15)
(ghtbutton(x,[[link,y],[contents,z]],basic_hypertext) $\leftarrow$
button(x,y,z)))

But as usual, more interesting laws can be defined as well. In our system, models and modeling variables are represented by buttons, via bridge laws between the generalized hypertext system and the application, TEFA.

$$\forall c, x \qquad (16)$$
$$(ghtbutton(x,[[contents,x],[owner,TEFA],[type,model]],c) \leftarrow$$
$$(model(x,\_,\_) \quad \wedge$$
$$context(c,x,model)))$$

$$\forall c, x \qquad (17)$$
$$(ghtbutton(x,[[contents,x],[owner,TEFA],[type,variable]],c) \leftarrow$$
$$(variable(x,\_) \quad \wedge$$
$$context(c,x,variable)))$$

Notice that, context permitting, the name of each model and of each modeling variable is a button. But, given that, e.g., a model's name is a button, what link ensemble does it name? Clearly, from the declarations the symbol also serves as a link name. Thus, we see that a given model may be represented as a button, a node, and a link, and that this is declared through universally-quantified laws that are instantiated inferentially as the occasion requires.

6.5 Generalizing the Session Log

Backtracking within generalized hypertext must take the context into account. To reduce disorientation when the user returns to previously-viewed nodes, they should be displayed in the manner they originally appeared. Thus, the original context and all other parameters necessary for restoring the node's original appearance must be captured in the session log. We model the generalized hypertext session log as *ghtsessionlog([[v, w, x, y, z, c, cc], ...], a)*, a chronological (*last-in-first-out*) list of identically formatted list elements, in which v is the command executed, w the button selected in "originating" node x, y is the link traversed to the "destination" node z, c was the context before traversal and cc is the new context resulting from the traversal.

6.6 Generalizing Link Traversal

In basic hypertext, as noted above, link traversal is normally performed through a select-traverse-display mod-
el: the user selects a button (e.g., by pointing to it with a mouse and clicking on the mouse), the system finds the link named by the button, traverses it, and displays the node found at the link's end point. (In the case of procedural attachment, the system may find a procedure at a link endpoint. If so, the system calls the procedure, which normally changes the content, or perhaps display, of a node.)

Our concept for generalizing link traversal is as follows. Inference (indeed, arbitrary processing) may occur both before and after traversal of a link. After the user selects a button, the system may perform a series of inferences in order to determine what the available links are, possibly taking context into account. If there are several options available, the user is then prompted to choose a particular option and (when needed) to supply parameters. (Alternatively, the system may invoke a default.) Inferencing is performed again in order to validate the refined request. Upon successful validation, the system determines (finds or generates) the appropriate link and traverses it. Traversal—which may itself be a complex inferencing process and may use application-level procedures—produces a symbol that names a node and computes the contents that represent it [23]. Inferencing, or processing, is then performed on these contents (e.g., for the purpose of formatting and display). Usually, this final inferencing results in display of a new node. Thus, our generalization follows a select-infer-traverse-infer model.

In order to see the logical import of this, recall our first browsing command for basic hypertext, formula 3. We have labelled various subclauses for the purpose of easy reference in the discussion that follows.

$$\forall t, u, v, w, x, y, z \tag{18}$$

$$(process\_command(traverse\_sink, w, v, \tag{19}$$

$$[update\_log(traverse\_sink, w, v, t, z)], x, y, z) \leftarrow \tag{20}$$

$$(button(w, t, \_) \quad \wedge \tag{21}$$

$$link(t, v, z, \_, display, y) \quad \wedge \tag{22}$$

$$node(z, u, \_) \quad \wedge \tag{23}$$

make_display_text(u,x))) (24)

Here the arguments passed are the input parameter w (normally the button selected), the active node v, followed by any update messages such as updating the session log (or the current context), the display text x, its display mode y and z the new node to make active.

How does the formula associated with line 18 need to be modified to accomodate our concept of generalized hypertext? First, contextual information must be brought into play. Second, at line 22 we need to generalize from trivially inferring a link by essentially looking it up in a table. Third, link traversal is accomplished at line 23 with, again, a trivial inference to look up u, the text associated with the node. Fourth, a step needs to be added after 23 in order to cover post-traversal inference. Although line 24 is in fact an inferential step, it has the not very interesting function of performing a simple transformation on a node's text to prepare it for display. These remarks lead naturally to the generalization of 18.

$\forall c, cc, r, s, t, v, w, x, y, z$ (25)

(process_command(traverse_sink,w,v, (26)

[update_log([traverse_sink,w,v,t,z,c,cc]), (27)

update_context(c,cc)],x,y,z,c) $\leftarrow$ (28)

(ghtbutton(w,t,c) $\wedge$ (29)

ghtlink(t,v,z,_,y,r,c) $\wedge$ (30)

map_node(z,w,r,y,c,s,cc) $\wedge$ (31)

make_display_text(s,cc,x))) (32)

Note that an argument, c, to represent the current context, has been added to the head of the formula associated with line 25. In 25, our generalization of 18, the first step, 29, corresponds to 21, but with contextual information added and the possibility that the values of its variables are deduced, rather than merely looked up. The second step, 30, corresponds to 22, again with the possibility of extensive inferencing. Step 31 corresponds to 23, but greatly generalizes it. From 30 we have the list r of eligible links forks, and perhaps the name of the destination node z. Resolving z, its display text s and its display mode y, however, often depends on the application command executed, since it normally determines the resulting report.

At 31, we apply the link ensemble to the node in the context, c, to produce s, the output of applying a link/command to the node, and cc, the new context. Extensive inferencing normally is performed here. In the event that r has many elements, the user must be prompted to make a single choice (see figure 3). The application (in our implementation, the model management system TEFA) than is called to execute the command on the object in the context. Again, complex inferencing will typically ensue. At 32, s, corresponding to u in 24, is transformed for display, and the process completes by determining the mode of display, y, in the new context, cc, of the display object, x, returned.

Thus, our generalized hypertext command model, 25, is quite similar to its basic hypertext relative, 18, but 25 has the effect of greatly generalizing 18. There are analogous changes for the other browsing commands discussed for basic hypertext.

7 Conclusion

Under our concept of generalized hypertext, nodes are objects (declared or inferred) and links declare (explicity or implicitly) operations that may be applied to objects, usually producing a hypertext node upon completion. These generalizations are the outcome of our intention to construct a hypertext system in which the cost of building hyperdocuments is greatly reduced through automatic creation of nodes and links on the basis of application-specific declarations. Bridge laws, and system-level procedures that implement these generalizations, work on application-specific declarations in order to make the necessary inferences for automatic linking, for automatic node, link, and button creation, and support for multiple views of nodes, links, and buttons.

An important element of our design concept is that the application should declare—explicitly or implicitly—what is important to it, and the generalized hypertext system should exploit these declarations in order to infer links, nodes, and views. Further, it is our hope that the recognized hypertext problems of network and cognitive disorientation are reduced by inferencing procedures that are broadly available for reporting on and explaining various system entities, notably nodes, links, and buttons. The essential idea is to employ a standard format to declare information about system entities

that map application elements, and to use generic (application-independent) inferencing procedures to generate nodes, links, and alternate views, and to provide system-level explanation features.

Briefly, and finally, we see several main areas for generalized hypertext that need to be addressed in future research. First, connections between simplifications of the logic model for generalized hypertext and other representational schemes need to be explored. Such other representational schemes (e.g., graph grammars [27, 28, 29] and Petri nets with automation semantics [21, 52]) offer the possibility of computational benefits. They also offer the possibility of a more detailed theoretical understanding of the nature and workings of the underlying representation. With special cases comes deeper understanding. Second, while our model provides for expression of context or local environments (e.g., the seventh argument of *ghtlink/7*), we have not here—or yet in our implementation, Max—exploited this information to anywhere near its full value. That, too, must wait for future research. Third, the logic modeling techniques employed above can, we are confident, be applied to modeling the internal structure of generalized hypertext nodes. Thus, for example, relationships between nodes (and even the existence of particular nodes) could be inferred automatically based on the application of general laws to the specific structures of existing nodes. Interesting relationships would include aggregation of information from several nodes (e.g., composite components [23, 24]), generalizations of information at a given node, and precedences of various sorts among nodes. If the useful possibilities here are limited, it is hard to see what those limitations are.

References

[1] Ackerman, Mark S. and Thomas W. Malone, "Intelligent Agents, Object-Oriented Databases, and Hypertext," *AAAI-88 Workshop, AI and Hypertext: Issues and Directions,* Mark Bernstein, ed., (1988), 1–3.

[2] Bhargava, Hemant K., Michael P. Bieber, and Steven O. Kimbrough, "Oona, Max, and the WYWWYWI Principle: Generalized Hypertext and Model Management in a Symbolic Programming Environment," *Proceedings of the Ninth International*

Conference on Information Systems, Janice I. DeGross and Margrethe H. Olson, eds., (November 30-December 3, 1988), 179-191.

[3] Bhargava, Hemant K., Steven O. Kimbrough, and Ramayya Krishnan, "Unique Names Violations, a
Problem for Model Integration or You Say Tomato, I Say Tomahto," *ORSA Journal on Computing,* **3**, no. 2 (Spring-1991), 107–120.

[4] Bhargava, Hemant K. and Steven O. Kimbrough, "On Embedded Languages for Model Management," forthcoming, *Decision Support Systems.*

[5] Bieber, Michael P., *Generalized Hypertext in a Knowledge-Based DSS Shell Environment,* Ph.D. dissertation, Decision Sciences Department, University of Pennsylvania, 1990.

[6] Bieber, Michael P., "On Merging Hypertext into Dynamic, Non-Hypertext Systems," Technical Report, Boston College, Computer Science Department, Chestnut Hill, MA, 02167-3808, 1991. (This supersedes "Issues in Modeling a 'Dynamic' Hypertext Interface for Non-Hypertext Information Systems," *Hypertext '91 Proceedings,* San Antonio, CA, December 15-18, 1991, ACM, 203–218.

[7] Bieber, Michael P., "Automating Hypermedia for Decision Support," forthcoming in *Hypermedia.*

[8] Bieber, Michael P. and Tomás Isakowitz, "Text Editing and Beyond: A Study in Logic Modeling," this issue of *Decision Support Systems.*

[9] Bieber, Michael P. and Steven O. Kimbrough, "Towards a Logic Model of Generalized Hypertext," *Proceedings of the Twenty-Third Hawaii International Conference on Systems Sciences,* IEEE Computer Society Press, Washington, D.C., (1990), 506-515.

[10] Bieber, Michael P. and Steven O. Kimbrough, "On Generalizing the Concept of Hypertext," *MIS Quarterly,* **16**, no. 1 (March 1992), 77-93.

[11] Bigelow, James and Victor Riley, "Manipulating Source Code in DynamicDesign," *Hypertext '87 Proceedings,* Chapel Hill, NC, (November 1987), 397-408.

[12] Bush, Vannevar, "As We May Think," *The Atlantic Monthly,* no. 176, (July 1945), 101-8.

[13] Campbell,Brad and Joseph M. Goodman, "HAM: A General Purpose Hypertext Abstract Machine," *Communications of the ACM,* **31**, no. 7, (July 1988), 856-861.

[14] Conklin, Jeff, "Hypertext: An Introduction and Survey," *Computer,* (September 1987), 17–41.

[15] Conklin, Jeff and Michael L. Begeman, "gIBIS: A Tool for All Reasons," *Journal of the American Society for Information Science,* **40**, no. 3, (1989), 200-13.

[16] Delisle, Norman and Mayer Schwartz, "Neptune: A Hypertext System for CAD Applications," *Proceedings of the 1986 SIGMOD Conference,* (May 1986), 132–143.

[17] DeRose, Steven J., "Expanding the Notion of Links," in *Hypertext '89 Proceedings,* 1989, pp. 249–258.

[18] DeYoung, Laura, "Hypertext Challenges in the Auditing Domain," in *Hypertext '89 Proceedings,* 1989, pp. 169–180.

[19] Engelbart, Douglas C., and W.K. English, "A Research Center for Augmenting Human Intellect," *AFIPS Proceedings,* Fall Joint Computer Conference, (Fall 1968).

[20] Feiner, Steven and Kathleen R. McKeowen, "Generating Coordinated Multimedia Explanations," *Proceedings of the Sixth IEEE Conference on Artificial Intelligence Applications,* Santa Barbara, California, March 1990.

[21] Furuta, Richard and P. David Stotts, "The Trellis Hypertext Reference Model," *Proceedings of the Hypertext Standardization Workshop,* in [43], pp. 83–94.

[22] Garg, Pankaj K., "Abstraction Mechanisms in Hypertext," *Communications of the ACM,* **31**, no. 7, (July 1988), 862-870

[23] Halasz, Frank G., "Reflections on NoteCards: Seven Issues for the Next Generation of Hypermedia Systems," *Communications of the ACM,* **31**, no.7, (July 1988), 836–855.

[24] Halasz, Frank and Meyer Schwartz, "The Dexter Hypertext Reference Model," in [43], 95-134.

[25] Hammwoehner, Rainer and Ulrich Thiel, "Content Oriented Relations between Text Units—A Structural Model for Hypertexts," *Hypertext '87 Proceedings,* (November 1987), 155–174.

[26] Harp, Brian, "Facilitating Intelligent Handling by Imposing Some Structure on Notes," *AAAI-88 Workshop, AI and Hypertext: Issues and Directions,* Mark Bernstein, ed., (1988), 79-83.

[27] Jones, Chris, "An Example-Based Introduction to Graph Grammars for Modeling," in J. F. Nunamaker, Jr. (ed.), *Proceedings of the Twenty-Third Annual Hawaii International Conference on System Sciences, Vol. III. DSS and Knowledge-Based Systems and Collaboration Technology Tracks,* Los Alamitos, CA: IEEE Computer Society Press, (January 1990), 433-442.

[28] Jones, Chris, "An Introduction to Graph-Based Modeling Systems, Part I: Overview," *ORSA Journal on Computing,* **2**, no. 2, (1990), 136–151.

[29] Jones, Chris, "An Introduction to Graph-Based Modeling Systems, Part II: Graph-Grammars and the Implementation," *ORSA Journal on Computing,* **3**, no. 3, (1990), 180–206.

[30] Jordan, Daniel S., Daniel M. Russell, Anne-Marie S. Jensen, and Russell A. Rogers, "Facilitating the Development of Representations in Hypertext with IDE,"in *Hypertext '89 Proceedings,* Pittsburgh, PA, November 1989, pp. 93-104.

[31] Kimbrough, Steven O., "On the Reduction of Genetics to Molecular Biology," *Philosophy of Science,* **46**, no. 3, (September 1979), 389-406.

[32] Kimbrough, Steven O., "On Shells for Decision Support Systems," University of Pennsylvania, Decision Sciences Department, working paper, 1986.

[33] Kimbrough, Steven O., and Ronald M. Lee, "Logic Modeling: A Tool for Management Science," *Decision Support Systems,* **4**, (1988), 3–16.

[34] Kimbrough, Steven O., Clark W. Pritchett, Michael P. Bieber, and Hemant K. Bhargava, "An Overview of the Coast Guard's KSS Project: DSS Concepts and Technology," *Transactions of DSS-90, Tenth International Conference on Decision Support Systems,* Linda Volonino, ed., Boston, Massachusetts, May 21-23, 1990, pp. 63-77.

[35] Kimbrough, Steven O., Clark W. Pritchett, Michael P. Bieber, and Hemant K. Bhargava, "The Coast Guard's KSS Project," *Interfaces,* **20**, no. 6 (November-December 1990), pp. 5-16.

[36] Koved, Larry, "Imposing Usability and User Performance with Hypertext Documents," *AAAI-88 Workshop, AI and Hypertext: Issues and Directions,* Mark Bernstein, ed., (1988), 121–2.

[37] Landow, George P., "Relationally Encoded Links and the Rhetoric of Hypertext," in *Hypertext '87 Proceedings,* Chapel Hill, North Carolina, (November 1987), 331-343.

[38] Lange, Danny B., "A Formal Model of Hypertext," in [43], 145-66.

[39] Marchionini, Gary and Ben Shneiderman, "Finding Facts vs. Browsing Knowledge in Hypertext Systems," *Computer,* (January 1988), 70-80.

[40] Marshall, Catherine C., Frank G. Halasz, Russell A. Rogers, and William C. Janssen, Jr., "Aquanet: A Hypertext Tool to Hold Your Knoweldge in Place," in *Hypertext '91 Proceedings,* San Antonio, TX, (December 15-18, 1991), 261-276.

[41] Mayes, J.T., M.R. Kibby, and H. Watson, "StrathTutor: The Development and Evaluation of a Learning-by-Browsing System on the Macintosh," *Computers and Education,* **12**, (1988), 221–9.

[42] Nagel, Ernest, *The Structure of Science: Problems in the Logic of Scientific Explanation,* Harcourt, Brace & World, Inc., New York, New York, 1961.

[43] National Institute of Standards, *Proceedings of the Hypertext Standardization Workshop,* Gaithersburg, (January 16-18, 1990), Special Publication SP500-178.

[44] Newcomb, Steven, Neill Kipp, and Victoria Newcomb, "The 'HyTime' Hypermedia / Time-Based Document Structuring Language," *Communications of the ACM,* **34**, no. 11, (November 1991), 67–83.

[45] Nielson, Jakob, *Hypertext and Hypermedia,* Academic Press, New York, New York, 1990.

[46] Nielsen, Jakob, "The Art of Navigating through Hypertext," *Communications of the ACM,* **33**, no. 3, (March 1990), 296-310.

[47] Parunak, H. Van Dyke, "AAAI Hypertext Position Paper," *AAAI-88 Workshop, AI and Hypertext: Issues and Directions,* Mark Bernstein, ed., (1988), 140–142.

[48] Parunak, H. Van Dyke, "Toward a Reference Model for Hypermedia," *Proceedings of the Hypertext Standardization Workshop,* in [43], pp. 197–209.

[49] Perlman, Gary, "Asyncronous Design/Evaluation Methods for Hypertext Technology Development," in *Hypertext '89 Proceedings,* 1989, pp. 61–81.

[50] Schatz, Bruce R., "Proposal to Attend Workshop on 'AI and Hypertext'," *AAAI-88 Workshop, AI and Hypertext: Issues and Directions,* Mark Bernstein, ed., (1988), 147–9.

[51] Schnase, John L. and John J. Leggett, "Computational Hypertext in Biological Modelling," in *Hypertext '89 Proceedings,* Pittsburgh, PA, November 1989, pp. 181-198.

[52] Stotts, P. David and Richard Furuta, "Petri-net Based Hypertext: Document Structure with Browsing Semantics," *ACM Transactions on Information Systems,* **7**, no. 1, (January 1989), 3-29.

[53] Shneiderman, Ben, and Greg Kearsley, *Hypertext Hands-On! An Introduction to a New Way of Organizing and Accessing Information,* Addison-Wesley publishing Company, Reading, MA, 1989.

[54] Smolensky, Paul, Brigham Bell, Barbara Fox, Roger King and Clayton Lewis, "Constraint-Based Hypertext for Argumentation," *Hypertext '87 Proceedings,* (November 13-15, 1987), 215-246.

[55] Van Dam, Andries, *FRESS (File Retrieval and Editing System),* Text Systems, Barrington, RI, (July 1971).

[56] Van Dam, Andries, "Hypertext '87: Keynote Address," *Communications of the ACM,* **31**, no.7, (July 1988), 887–95.

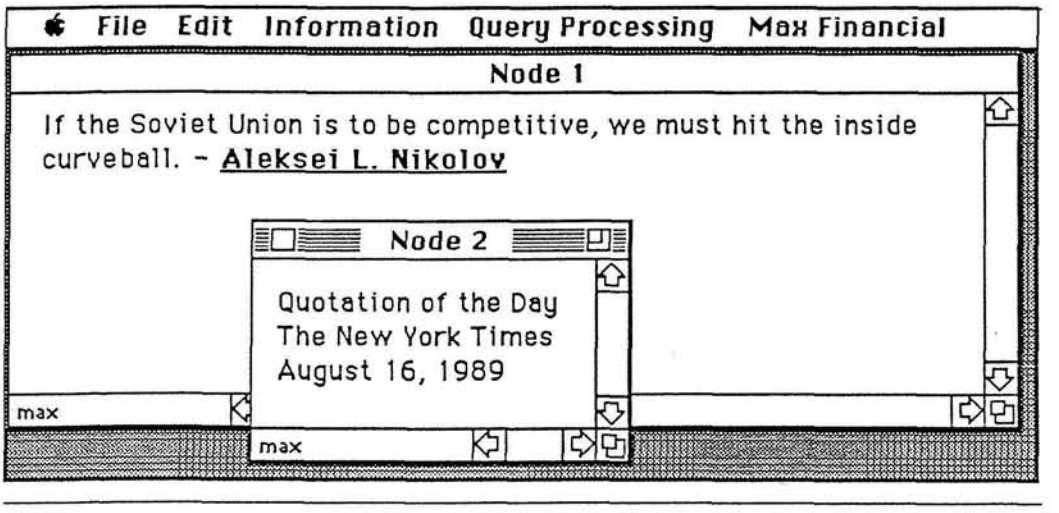

Figure 2: Basic Hypertext Illustration

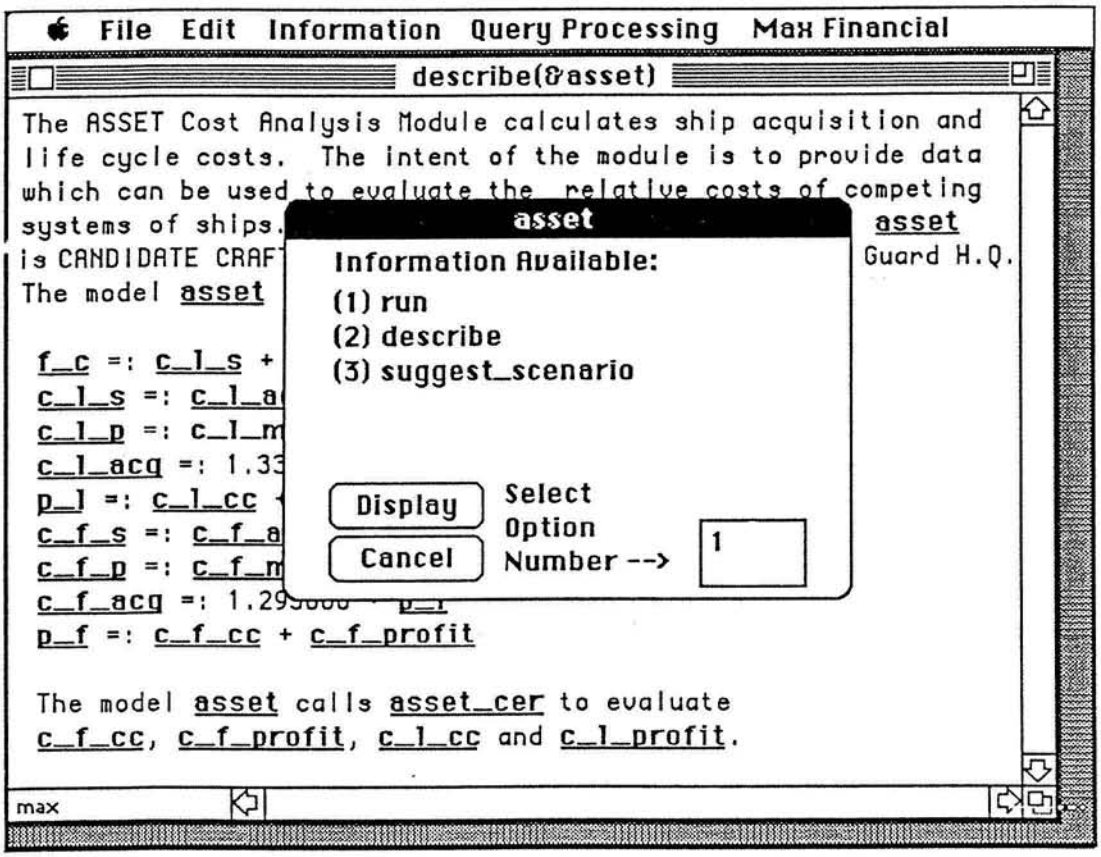

Figure 3: Description of the ASSET model, from Max Financial