Hypermedia

VOLUME 4 NUMBER 2 1992

AUTOMATING HYPERMEDIA FOR DECISION SUPPORT

MICHAEL BIEBER
Boston College, Carroll School of Management
Chestnut Hill, MA 02167-3808 USA

Many 'first generation' hypermedia systems were designed to support applications, which do not require the dynamic and general characteristics necessary for our domain — decision support systems (DSS). The heart of our research is a dynamic model of hypermedia incorporating *virtual structures* and *computation*, which we call *generalized hypermedia*. Generalized hypermedia broadens and automates the 'static' or non-virtual notion of first generation hypermedia for a knowledge-based DSS shell. The shell provides a hypermedia-style interface for *navigating* among DSS application models, data and reports. Such a shell should support applications in a variety of fields, e.g., engineering, manufacturing, finance, and therefore must provide hypermedia support as general, system-level functionality. Generalized hypermedia superimposes a hypermedia *network* on a DSS application, generating all hypermedia nodes, links and link markers dynamically from the application's standard, non-hypermedia knowledge base. In this paper we demonstrate how automating hypermedia can enhance decision making with a DSS. We describe generalized hypermedia and discuss the challenges presented to it by a dynamic, real-time environment.

INTRODUCTION

One of our main research goals is to make it easier and cheaper to build and use decision support systems (DSS). We want to enable application builders to construct DSS applications faster, and with more functionality than traditional DSSs have. In this article we explore how a dynamic view of hypermedia can support decision makers and their analysts. (Our 'dynamic' focus is on the ever-changing knowledge within an application, not the media of its elements. Our methodology should apply to all media formats.)

The following scenario describes a decision support system application for vehicle procurement that has a hypermedia-style interface. We shall refer to this scenario throughout the article.

Scenario: A hypermedia-supported decision analysis

Imagine that, as Vice President for Operations, you are presented with the final report of a procurement committee. You must decide whether to accept its recommendation. The committee analyzed three alternative vehicle fleet configurations for your division, each involving complex financial calculations and reams of data. Some

information used in the committee's analysis is reliable, some is the result of experienced guesswork. The committee's report consists of a one-page summary on paper and a floppy disk. You turn on your personal computer and open the report. A summary appears in the window entitled 'Final Committee Report.' (See Fig. 1a.) Boldface text are hypermedia markers linked to additional information. You shall be accessing all DSS operations directly from these markers in the summary and lower-level documents. (While this scenario's figures include only text, other media formats are permissible and their markers also would be highlighted in some manner.)

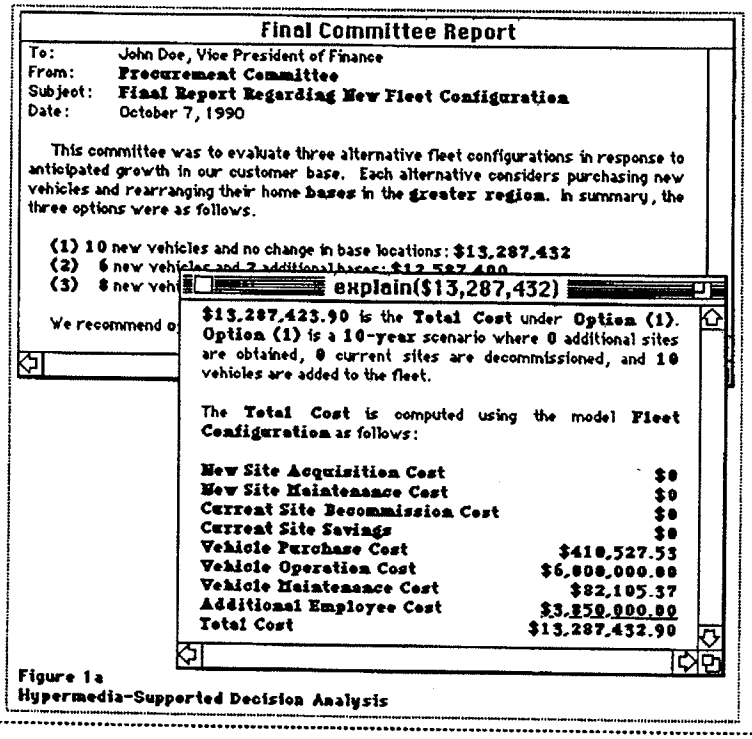

FIG. 1a. *A hypothetical interactive document from the opening scenario's DSS procurement application. Interactive documents provide the following functionality. All text highlighted in boldface are hypermedia link markers indicating the presence of hypermedia links to documents and other related information. Here the user 'drills down" for further detail by selecting the link marker '$13,287,432' representing the cost of 'Option (1).' The system generates and displays the explanation in the second interactive document.*

You want to get a feel for the accuracy and feasibility of the committee's recommendation. First you explore the definitions of parameters such as the 'greater region' and the composition of a 'base.' Next you select the cost projection for option (1): '$13,287,432.' The computer dynamically generates the explanation shown in the second window in Fig. 1a. From here you examine the total vehicle purchase cost in more detail. The computer generates a third report describing the financial model and the data used in determining the cost, including the individual vehicle sales price. (See Fig. 1b.) This

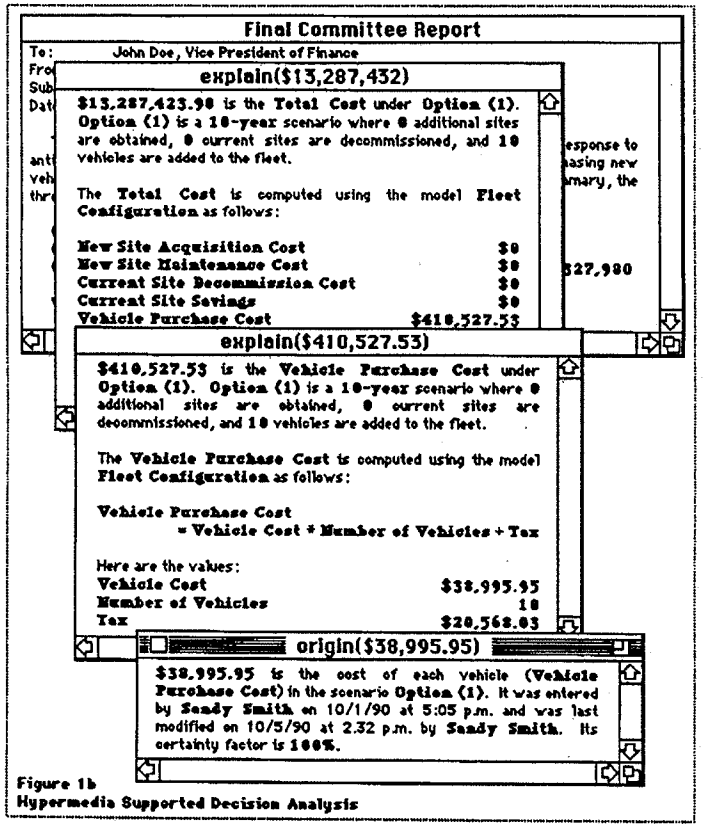

FIG. 1b. *To explore the vehicle purchase cost, the user selects the marker '$410,527.53' in the second interactive document. The system generates the explanation in the third report. Next the user selects the individual vehicle cost figure represented by the link marker '£38,995.95' in the third to last line. The system traverses the corresponding link, generating the interactive document entitled 'origin ($38,995.95)*

morning, however, you attended a meeting with the vice presidents of operations in other divisions. One of your colleagues discussed a purchase arrangement with a major auto maker. Its purchase price was lower and you should be able to make the same deal. The number in the committee's report, therefore, seems too high, so you request its origin to see when it was last updated. (See Fig. 1b.) Returning to the summary document you select the 'Procurement Committee' marker to get the name of the committee head. The system has a direct link to the company's personnel database and dials the phone number. You ask the committee head to investigate a similar purchase arrangement. She calls you back later with a revised price. Before requesting that the computer update the analysis, you inquire where else the analysis uses the purchase price. You also check which other documents produced by the procurement committee use this parameter. You then alter the purchase price in the database and return to the summary sheet. All the figures reflect the change.

Next you look at the other options. The computer generates a more detailed description of each. Most of the descriptions appear correct, but one of the costs under the third option seems questionable. Selecting this cost marker, you request the source and any other information available. The source turns out to be a trusted adviser, who has attached a comment about the data. Satisfied, you return to the summary sheet. After a few more minutes of investigation you feel confident about your decision.

GENERALIZED HYPERMEDIA

Our domain, decision support systems[1,2] (DSS), is dynamic. Users perform real-time analyses on application models and receive automatically generated reports, which are linked to objects that change without the user's intervention, such as the price of a stock. Many of today's 'first generation' hypermedia systems, while successful and quite appropriate to their particular domains, are limited in their dynamic support. They were designed for 'static' environments, in which users construct and alter hypermedia networks manually. To cope in a dynamic environment we had to generalize and automate the manual, 'static' notion of hypermedia[3]. The result is our model of *generalized hypermedia*[4,5,6]. We shall contrast these two views of hypermedia after describing its role in decision support in the next section.

At its most basic level, *hypermedia*[7,8] is simply the concept of *linking* any two pieces of information and providing a computer-aided mechanism for navigating between them. (Hypermedia subsumes the

term *hypertext*, in which the information — technically — is purely textual.) We refer to the pieces of information at either end of the link as *nodes*. We signal the existence of a link from a node by highlighting a portion of the node's display text, which we call a *button* or *link marker*. When a user selects a link marker the computer *traverses* its link and displays an appropriate representation of its destination node. Minch[9] provides an extensive discussion of hypertext and DSS.

In generalized hypermedia we broaden the models of hypermedia components — nodes, links, link markers, etc — in two basic ways. First we take advantage of three of Halasz' proposed extensions to hypermedia[10]: virtual structures, computation, and filtering or 'tailoring'. We use these to generate hypermedia components on the fly from basic declarations we call *bridge laws*. As we shall see later, we use bridge laws to separate application-specific details (such as all the numbers in Fig. 1b) from the hypermedia components to which they are mapped. Bridge laws enable generalized hypermedia to superimpose a hypermedia *network* on a DSS application, generating all nodes, links and link markers dynamically from the application's standard, non-hypermedia data or knowledge base. Embedding generalized hypermedia functionality at the *system-level* of an information system enables us to provide hypermedia support independent of individual applications[11]. This makes hypermedia functionality transparent to application builders, yet available to application users.

Our second set of generalizations takes advantage of bridge laws to extend the functionality of hypermedia components and provide a more flexible mapping. For example, we model two classes of nodes[12,4]: *abstract concept nodes*, which have no 'natural' display contents[13], and display nodes, which readily can be displayed. In addition, we can model virtual nodes as any object to which information currently is linked, including the session log, formatting templates, link markers and other links. Another useful generalization is to associate multiple *n-ary* links[14] with a single link marker. We explain this second set of generalizations and the flexibility it provides for different tasks and types of users in other papers[4,6,15].

In the following sections we discuss hypermedia's benefits and the challenges it faces in the dynamic environment of decision analysis. We set the stage by explaining how a hypermedia-oriented DSS can support decision makers. This demonstrates why we need a dynamic view of hypermedia. Then we show how generalized hypermedia extends what we call *static hypermedia*, the way hypermedia often has been implemented. Next we look at the problems posed by dynamic

environments. We conclude with a challenge for the hypermedia community.

HYPERMEDIA AND DECISION SUPPORT

We shall analyze the process of decision support using Simon's *intelligence-design-choice* model of decision making[16] and Mintzberg et al.'s *authorization*[17]. *Intelligence* is the act of gathering information; design involves developing alternative scenarios or problem solutions; *choice* is selecting the 'best' option; *authorization* is the process of justifying the recommendation or decision made to those affected by it. In a typical decision analysis, these stages are interwoven, not sequential. They implement the *argumentation theory of DSS*, according to which 'the main purpose of a DSS is to support the construction, evaluation and comparison of arguments for courses of action'[18,19]. These 'arguments' also can justify the course of action. Hypermedia should facilitate such support, giving the user easy access to information and operations within the DSS application without overwhelming him or her with details.

The argumentation theory of DSS is separate from — but does support — the concept underlying many hypermedia-oriented argumentation systems (e.g., AAA[20], Aquanet[21], EUCLID[22,23], gIBIS[24], JANUS[25], PHIDIAS[26], RelType[27]). Argumentation systems, often based on Rittel's Issue Based Information System (IBIS) framework[28], use typed nodes and links to express the deliberations of the participants in some process, thereby capturing the logic employed in arriving at a decision. Such systems can support decision makers in the four stages of decision making and justification.

In the paragraphs that follow we discuss the synergism between DSS and hypermedia in each of the four stages. In preparation, a short primer on DSS applications is in order. DSS applications contain decision *models* and *data scenarios* — the complete set of input data values for a model's *variables*. Variables without input data values are calculated by *executing* the model with a data scenario. For example, consider a simple model calculating the total vehicle cost:

*Total Vehicle Cost = Single Vehicle Cost * Number of Vehicles*

A data scenario could contain input data values $20,000 for the variable *single vehicle cost* and 5 for *number of vehicles*. Executing this model calculates the value for the variable *total vehicle cost*. Often an analyst will construct several data scenarios to test a model under various circumstances (such as the range of possible prices for replacement vehicles five years from now) or a new situation (such as acquiring snow removers instead of delivery vehicles). There are many types of decision

models: algebraic models (e.g., the combined set of formulas on a single spreadsheet), optimization models (e.g., the set of equations used in a single linear program), simulation models, etc.

Intelligence

During the intelligence stage, the decision maker or analyst explores the decision domain — the collection of information directly or indirectly pertinent to an area of interest — either looking for a problem or opportunity, or responding to one. A DSS can help by maintaining the pertinent decision models, data and reports. For example, for a mortgage domain the DSS would contain models, data and reports about the economy, interest rates and the housing market. The models would help a loan officer evaluate a client's risk profile and determine what rate to charge. In a procurement domain the DSS would contain life cycle cost models (e.g., construction, operation and maintenance equations) as well as exact or statistical data on the items being purchased or constructed. Reports would record the decisions made and document their justifications in case of management questions, audits or even lawsuits regarding the vendors chosen. Similarly, scientific and engineering domains would have their own sets of models, data and standard reports.

The DSS should do more than maintain models, data and reports. It should assist the analyst in accessing and understanding them. Hypermedia can help. Presenting concise information displays embedded with links reduces the *cognitive overhead* [7,29] of manoeuvering within the complex domain of DSS models and scenarios. Analysts can explore selectively, choosing what they want and bypassing what they understand or consider less important. Several types of objects should be linked to provide this access. Related models, data and reports should be linked. Models should be linked to their submodels and variables. These variables should be linked to all possible data values registered in an application data base. All the above should be linked to any execution results derived from them, especially when these results appear in reports. The user should be able to retrieve comments, annotations, definitions, explanations and any background information inferable about items of interest. (See Fig. 2.)

In sum, the DSS interface can use hypermedia concepts when structuring and presenting relevant information in a comprehensible and useful format to DSS[3,30,9,31]. Hypermedia-style markers can act as *embedded menus*[32], giving the analyst 'context-sensitive' access to DSS operations and background information. We have dubbed this hypermedia-style accessibility the *WYWWYWI ('what you want, when you want it') principle*[33].

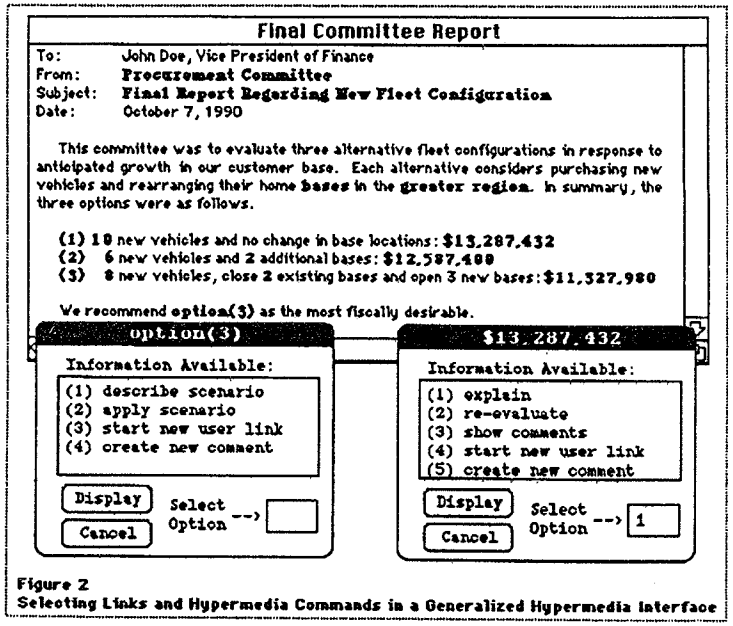

Figure 2
Selecting Links and Hypermedia Commands in a Generalized Hypermedia Interface

FIG. 2. *This is the precursor to Fig. 1a. The user has selected two link markers, the 'option (3)' data scenario and the execution result '$13,287,432.' The system then displayed all appropriate DSS operations and connections to information related to these markers, e.g., describe the selected model, execute the selected model, explain the selected execution result. Each of these application-specific operations represents a hypermedia link. In addition, the hypermedia engine adds several hypermedia-specific commands for creating user-defined links and comments, as well as a link to existing comments for the execution result selected. The user can choose a link to traverse or a command to invoke. In this illustration the user has chosen an 'explanation' link for the marker '$13,287,432,' i.e., the user has typed '1' in the second query dialog. Traversing the explanation link results in the second interactive document of Fig. 1a.*

Design

To address the opportunity or problem-at-hand, the analyst designs possible solution scenarios. Doing so can involve creating additional decision models and developing alternate data scenarios. To support hypermedia-style navigation, the DSS must register the new models and scenarios, and create the hypermedia network representing them — the nodes, links and link markers that the analyst will use to access them[3]. This also is an opportune time for the analyst to make comments about the scenario values and add links to related information.

Choice

'Choice' involves evaluating the alternative solution scenarios. According to the *principle of limited search*[18], the analyst will use the DSS to investigate alternatives until he makes an optimum choice (if one can be determined) or quits, at which time he will choose the best alternative found (assuming it is 'satisfactory'). Normal DSS operations for evaluation include executing models, and performing 'what if' and "sensitivity" analysis. For example, an analyst evaluating the cost of operating different vehicles may check model results over a range of possible petroleum prices or under fluctuating inflation rates.

As before, hypermedia provides easy access to DSS operations such as model executions and explanations, as well as other relevant information. (See Fig. 2.) At this point, analysts should annotate each scenario further, explaining how they tested it and why they did or did not choose it. This will be especially important for justifying recommendations during the authorization stage.

Authorization

The decision maker not only has to reach a decision, but also must convince others of its validity. Analysts should be able to elicit a DSS's help in promoting an action, presumably through a report that both describes the recommendation and supports it with data. (If the analyst works for the decision maker, as the procurement committee effectively did in the opening scenario, the report first must persuade the decision maker.) The decision maker then can use this report both to convince others and to document the decision made.

The report's reader is now self-sufficient. He can explore the report and all supporting information by browsing on the computer without active assistance from the report's author. The DSS (ideally) produces the answer to any question the reader has, especially if analysts or decision makers have added comments. He can investigate any piece of information, exploring all the way 'down' to its origin (as we saw in Fig. 1b), thereby increasing his overall understanding of, and confidence in, the decision.

Annotated trails such as guided tours[34] and scripted documents[35] provide another justification tool. Trails lead a reader along a specific path through a hypermedia network. In a DSS, trails can represent entire decision schemata[9]. The trail would present the analysis in a logical sequence and include comments describing the options, analyses and decisions made. As noted previously, argumentation-style information also could help in documenting decisions.

AUTOMATING HYPERMEDIA

Building an adequate hypermedia network to support even the authorization stage is impractical in a static hypermedia system. Consider the interactive documents supporting the summary report in Fig. 1b. Normally an analyst would have to construct these manually. Suppose the system could generate them automatically, as well as all other lower-level definitions and explanations. This would expedite the analyst's task dramatically! It is this functionality that we describe now. We begin by demonstrating limitations of static hypermedia.

Boundaries of static hypermedia

Static hypermedia systems were not designed to support the DSS environment we just described and could do so only with a great restriction in functionality. To illustrate this, in Appendix 1 we give the internal code for part of the static hypermedia knowledge base defining the interactive documents of Figs. 1a and 1b. The system would generate such code for each node, link and link marker created by the application builder through the DSS's interface. This long list of declarations, however, highlights a problem — static hypermedia knowledge bases generally consist of explicit, pre-declared entries. This means that each application builder must anticipate his users and declare *a priori* all the nodes (e.g., decision models and variables), links (e.g., the relationships between models and DSS operations for them) and markers (e.g., the names and display values of nodes appearing in each report) he believes users will want to access in the future. The number of hypermedia components grows exponentially with the number of models, variables, scenarios and reports. All the possible executions, execution results and explanations implicit in Figs. 1a, 1b and 2 would require an immense amount of pre-analysis. Besides, an application builder could not pre-declare all the hypermedia elements in his application because a DSS usually is dynamic. Its knowledge base contains definitions, models, data and documents, but not execution results or their explanations. The latter are produced dynamically upon user request, and so must any hypermedia links mapped to them. Our only feasible option is to generate the hypermedia network automatically upon demand.

Knowledge-based DSS shell

Understanding the architecture of our DSS shell will frame our discussion of bridge laws and the hypermedia engine and help illustrate how we automate hypermedia.

We implement generalized hypermedia as system-level functionality within the interface of a *knowledge-based DSS shell* that supports

multiple DSS applications (see Fig. 3). By a *knowledge-based* system we mean one that stores 'knowledge' about its external environment (e.g., user profiles, communication protocols, application-specific decision models, data and commands, external database information) and its internal computations (e.g., the current context, hypermedia commands, bridge laws). *Shells*[36,19,37] provide standard functionality and a common 'look' to a range of applications, with the goal of decreasing the effort involved in building and using them. For example, we can consider spreadsheet packages such as Lotus 1-2-3 as shells. They provide a standard interface (the tabular worksheet), a set of functions (e.g., statistical, financial, arithmetic) and a set of menu commands. The applications are the individual worksheets or sets of interrelated worksheets built by entering formulas and data values in the interface provided.

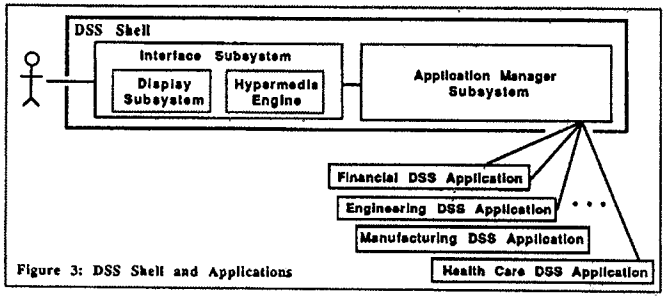

FIG. 3: *DSS shell and applications*

Our knowledge-based DSS shell produces *interactive documents*, such as those in Figs. 1a and 1b. Interactive documents give direct, context-sensitive access to reports, operations and other components of DSS applications. Ideally the reports generated by a DSS will be concise — focused on a specific component or analysis with access embedded to further details and related information. Generalized hypermedia provides this access dynamically.

The shell has two major subsystems — the interface subsystem and the application manager subsystem.

The interface subsystem

The interface subsystem contains the hypermedia engine[11] and the display subsystem. The hypermedia engine creates interactive documents and menus from application requests. In doing so, it maps hypermedia nodes, links and markers to the requests' contents. The hypermedia engine adds value by maintaining user-declared links and

comments on behalf of applications. The display subsystem shows the menus and interactive documents, and passes back user requests for analysis.

The application manager subsystem

The application manager subsystem provides the interface between the hypermedia engine and individual DSS applications. DSS applications contain decision models and data specific to their individual domain. The application manager subsystem provides the commands that execute the models, provide explanations and generate the contents of decision reports. (In this way it is analogous to spreadsheet packages and programming languages, which provide the commands that users of individual worksheets and programs employ.)

When automating hypermedia, it is important to implement its functionality — creating, exploiting and managing links and linked information items, etc. — in a general manner that will work for both existing and future shell applications. Like database systems and user interface management systems, the hypermedia engine, therefore, must be an application-independent, system-level tool for providing hypermedia functionality.

We have implemented many of these concepts in a text-based prototype system called Max, which is used by the U S Coast Guard[19,6]. While Max supports most of the functionality described in this article, it currently implements a preliminary version of bridge laws, our technique for generating a hypermedia network for a DSS application.

Bridge laws: a technique for automating hypermedia

How do we automate hypermedia in a knowledge-based DSS? The hypermedia engine cannot take an arbitrary application's knowledge base and magically infer which elements correspond to hypermedia nodes and links. Instead, the shell's application manager subsystem must provide some translation routines that the engine can use to make its inferences. We adapted the term *bridge laws*[38,39,40] for these translation routines because they serve as a 'bridge' or connection between elements defined in the language of the application's knowledge base and those in the shell's hypermedia engine. Bridge laws are at the heart of our model of generalized hypermedia. Generalized hypermedia bridge laws exploit logical quantification, i.e., they apply to all cases that satisfy the set of conditions specified within the bridge law. For example, one may declare a bridge law with a structure similar to:

For each variable that satisfies conditions X, Y and Z:
map a hypermedia link between the variable and the decision model using it, and
map a hypermedia link between the variable and its data value in each scenario compatible with the model.

Logical quantification (i.e., specifying 'for each' or, in Appendix 2's logical equations, '∀') enables individual laws to map entire classes of application objects (e.g., models or calculations) to hypermedia components; the same bridge law will map every object in the application knowledge base that satisfies the bridge law's conditions.

As an example, in Fig. 2 three links emanate from the '$13,287,432' marker in the Final Committee Report. This marker represents the calculated cost for the configuration scenario labeled 'option(1).' The first two links, 'explain' and 're-evaluate,' represent DSS operations. The third, 'show comments," is a link to user comments about the calculation. (For the sake of this discussion we shall ignore the two interface-level commands, 'start new user link' and 'create new comment.') How can the hypermedia engine generate this set of links in a general manner that applies to any calculation, not just this specific one? Given any link marker representing a DSS calculation, in Appendix 2 we describe a set of seven bridge laws that generates this trio of links. These bridge laws will work for every DSS calculation — existing or future — resulting from executing a DSS application model. We describe similarly compact sets of bridge laws that map application models, variables, data scenarios, etc. to hypertext components elsewhere[4]. This methodology provides for a relatively small and stable set of bridge laws serving a broad range of applications.

Our methodology applies beyond DSS[37]. Bieber and Isakowitz, for example, have developed bridge laws mapping a relational database management system[41]. The model of generalized hypermedia achieves this broad applicability, in part, by providing each mappable hypermedia component with arbitrary attributes that can represent any aspect of the target system (including its media formats). For example, we can use these attributes to represent the node and link types we find in argumentation systems. Indeed, we have mapped a generalized hypermedia network to a static hypermedia argumentation system[4].

Bridge laws should be developed by the person who knows the application manager subsystem the best — the systems programmer who builds or maintains it. Once in place they should map a hypermedia network to any application, as long as the application builder declares its models and data in a format the application manager subsystem can recognize. This, however, is simply analogous to expecting each spreadsheet builder to use the standard formulas of his spreadsheet package.

Thus, application builders and users need have no knowledge of bridge laws. To them, hypermedia functionality occurs automatically! We summarize this point in Fig. 4. The application manager subsystem provides the bridge laws that the hypermedia engine uses to map hypermedia nodes, links and link markers to an application's objects and operations. The user navigates among and exercises the application components indirectly through their hypermedia representation, yielding the benefits described earlier. We discuss bridge laws in more detail and provide additional examples in other papers[4,5,12,41].

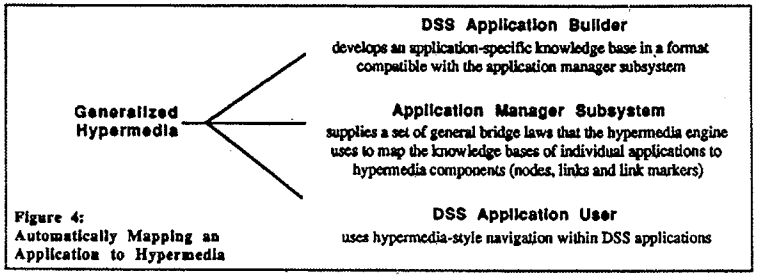

FIG. 4. *Automatically mapping an application to hypermedia —*

Three aspects combined distinguish generalized hypermedia from other hypermedia approaches: (1) generalized hypermedia is a fundamentally dynamic model; (2) it can provide system-level support to any application software with commands and data in a well-defined internal structure; and (3) it uses general bridge laws as a method of mapping a hypermedia network to an application without adding to or deleting from the application's data or knowledge base. Generalized hypermedia is not the only dynamic model of hypermedia. Systems such as Expertext[42], KMS[43], NoteCards[10], Guide[44] and HyperCard[45] do support dynamic ('virtual') linking and node generation ('computation') on an *ad hoc* basis. Schnase and Leggett provide a specific example in KMS[46]. StrathTutor[47], for example, processes all components dynamically. Many of the recent hypermedia modelling efforts (e.g., Trellis[48], Dexter[14], HyperBase[11]) also allow dynamic elements. Generalized hypermedia is not the only model providing general, system-level hypermedia functionality. There are operating system-level hypermedia toolkits for adding hypermedia constructs — nodes, links, markers, etc. — to application data (e.g., the Andrew Toolkit[49], and Maurer and Tomek's proposed 'core system'[50]). Others have developed autonomous facilities that run as separate components providing hypermedia services to distributed applications running concurrently in networked environments

(e.g., PROXHY51 and the Sun Link Service[52]). Few methods externally superimpose hypermedia constructs over an application without adding to the application's data or knowledge base (e.g., Puttress and Guimaraes's Hypertext Object-oriented Toolkit[53]). Our methodology of bridge laws is, however, a unique approach to generating a hypermedia network for 'non-hypermedia' applications.

Generalized hypermedia bridge laws make extensive use of three features that Halasz[10] identifies among the outstanding issues in hypermedia research[4]: (1) creating and manipulating *virtual structures* of hypermedia components; (2) *computing* over the knowledge base during link traversal; and (3) *tailoring* the hypermedia network.

Virtual hypermedia component structures

Generalized hypermedia bridge laws are examples of virtual components. By *virtual* we mean not fully *resolved*; not all the parameters have been *instantiated* or filled in with values. The uninstantiated parameters can take any value compatible with the bridge law's conditions. For example, the '$13,287,432' link marker in Fig. 2 is virtual when the user selects it because the hypermedia engine does not determine which links are associated with it until someone selects it. (This is the hypermedia equivalent of 'just in time' delivery.) In this example the hypermedia engine uses link bridge laws such as those in Appendix 2 to resolve two compatible instantiations: a link to the number's explanation and a link to its reevaluation. One can think of virtual components being represented by templates that prescribe both the component's internal format — what its parameters are — and how, during link traversal, the hypermedia engine should *infer* the values for these parameters. When the hypermedia engine has instantiated all its parameters, we say that the virtual component is *resolved* and that the hypermedia engine has created or generated a specific *instance* of the component.

When the user selects the '$13,287,432' link marker, the hypermedia engine uses the bridge laws for link markers and links to *infer* which links could exist for this marker. The engine tentatively fills in a template for each link with values drawn from models and data in the application knowledge base. The user then chooses one of the tentative links to traverse, as shown in Fig. 2. At this point the link marker is resolved, but not yet the link. The hypermedia engine now must infer several pieces of information to finish resolving the link chosen. One is the type of link — its DSS operator (e.g., 'explain' or 're-evaluate'). Another is the link's destination. The hypermedia engine uses additional bridge laws to infer these from the contents of the application knowledge base. Once these

parameters are known, we consider the link fully resolved because the template is complete. We have yet, however, to generate the destination we have identified.

Computing over the knowledge base during link traversal

The next step is *computation*, i.e., actually generating the contents of the destination node specified by the link. Normally, a DSS operation (e.g., executing a decision model) must be performed within the application manager subsystem to create the destination node (e.g., a decision report describing the results of the operation). Once created, the hypermedia engine prepares the outcome for display in an interactive document using bridge laws to determine which portions to highlight as new link markers. These new markers will be virtual, for as we saw, none will be associated with a link until a user selects it and the resolution cycle repeats.

Tailoring the hypermedia network

Applications should *tailor* processing and resultant reports to a given user's skill level and the specific task for which he is using the DSS application. The shell can do this by maintaining multiple *filter sets* or modes, each associated with a different 'view' of application knowledge base components and with a different set of report formats. Bridge laws invoke different views by including filters in their sets of conditions[4,12]. The hypermedia engine checks filter settings as part of every inference it performs.

In summary, we say that link traversal in generalized hypermedia follows a *select-infer-traverse-infer* pattern. The user is interested in some object in an interactive document. If that object is a link marker, he can *select* it. The display subsystem from Fig. 3 passes the marker's internal identifier to the hypermedia engine. The hypermedia engine now *infers* the set of links that the user can traverse from this marker. The hypermedia engine applies the marker's identifier to two sets of bridge laws. The application manager subsystem provided one set. The hypermedia engine uses these to infer the DSS operations available for the object that the link marker represents, mapping each operation to a separate link. The second set of bridge laws was provided by the hypermedia engine itself. These generate links to interface-level items such as comments about the object that the link marker represents. The result is a set of virtual links displayed to the user as in Fig. 2. The user chooses one of these for *traversal*. Traversing a link representing a DSS operation causes the application manager subsystem to perform that operation upon the appropriate objects in the application knowledge

base. This 'computation' generates execution results, which the application manager subsystem embeds in a report passed to the hypermedia engine as a 'request for display' message. The hypermedia engine now determines how to display the message's components. Usually it creates an interactive document. Compiling an interactive document includes *inferring* which parts to highlight as hypermedia link markers and which to leave as plain text[4,12]. (We note that the hypermedia engine uses filters pervasively in this cycle.) Displaying this report concludes the *select-infer-traverse-infer* cycle. It is this process that enables users to access DSS applications through a hypermedia interface.

Thus, link traversal in generalized hypermedia is a dynamic process. If the contents of an application knowledge base or the filter settings change, it is quite possible that traversing a link today will yield different results from that of its traversal yesterday. Dynamically mapping hypermedia to a set of objects presents challenges that static hypermedia systems do not face. These are the topic of the next section.

CHALLENGES IN A DYNAMIC ENVIRONMENT

In this section we consider problems caused when the elements underlying the hypermedia network change without warning beforehand or notification afterwards.

For purposes of comparison, consider the standard problem of a link endpoint document having been deleted — with or without the knowledge of the hypermedia system — and therefore no longer being available as a link destination. This problem arises in forward traversal as well as backtracking. (Utting and Yankelovich discuss this, for example, in conjunction with Intermedia's 'Web Views'[54].) In a DSS, not only may user documents be deleted, but so may objects in application knowledge bases. Even when objects remain, their data and parameter values may change. Imagine a document reporting the current value of a stock.

How does this affect a dynamically generated hypermedia network? The user can never be sure that the link he traverses today will be valid or even available tomorrow. This can lead to surprises for users who do not anticipate changes, and link to or comment upon an application object that does change (e.g., data scenarios, data values, results of model executions) or application reports containing such objects. The user-declared link or comment may not be valid the next time that object appears in a generated report. The same goes for application objects 'pasted' into user-created documents when these documents are re-opened. This calls for some type of version management, allowing the user to specify whether he wants to see up-to-date information or an

older version. This is especially important for the 'authorization' decision making stage, in which one may need to recreate the application environment at the time a decision or recommendation was made in order to justify it. In addition, being able to explain changes would be an especially helpful feature in a DSS where the change, say, to a model execution result may be caused indirectly by a change to an underlying data value or model parameter. We are just beginning to tackle versioning, which other hypermedia developers do address (e.g., DIF in the software engineering domain[55] and HAM[56], which is a static hypertext model).

'Links to real-world objects are subject to a kind of entropic degradation, because the linked-to information is subject to change.'[57] We should determine (somehow) when user-declared links and comments are no longer accurate or relevant. This is a problem with both static and dynamic environments. We may be able to take advantage of the known structure of the application knowledge base to determine automatically when, at least a subset of these links and comments is no longer valid. If the user could declare some validity conditions based on parameter or data values in the underlying application knowledge base, then the hypermedia engine should be able to check these before making the link or comment available. For example, the user may declare a comment or link to an action plan, which becomes accessible only when the price of a stock moves outside a specific, computed range. When this condition does exist, selecting the appropriate virtual marker will find this link or comment. When the condition does not exist, the link will not be available. Another approach could be to embed a formal logic-based *truth maintenance system* within the shell that would analyze validity conditions[30].

Part of processing application-generated document contents is locating the endpoint markers of user-declared links or comments. Recall that these are interface-level hypermedia entities provided by users; applications are not notified of their existence. If markers are associated with, say, arbitrary portions of text instead of embedded application objects such as application-provided keywords, the hypermedia engine must handle the case where the text string has shifted to a new location within the generated text. For example, assume a user comments on the last sentence of the '$38,995.95' document in Fig. 1b concerning certainty factors. The text of this sentence is not stored intact, rather it is recompiled each time this explanation is requested. Depending on what comes before it (e.g., the length of the name of whoever last modified this data value), the character position of this

sentence could shift between displays. Thus, solely registering the character position of text associated with comments or user-declared link endpoints is not adequate. The hypermedia engine must capture additional information when the user creates the link or comment. We have developed a preliminary algorithm for doing this based on a knowledge of the structure of the document containing the arbitrary link endpoint[4].

Tailoring causes another kind of change. Our implementation of filtering will allow conditional virtual entities and computation, as well as multiple report formats for different users and tasks. Thus the same report 'object' may have different contents generated for different users (e.g., a different level of detail). User-declared comments and links to one version may map entirely differently to another (or perhaps not at all). Virtual markers in two versions may yield different sets of link options. We have yet to determine how difficult such features will be to implement effectively.

Our model of generalized hypermedia addresses several of these issues to the extent noted in this section. We intend to refine generalized hypermedia further regarding all of these issues.

CONCLUDING REMARKS

Hypermedia is recognized as a method for reducing the perceived complexity of information systems. Generalized hypermedia is a method for reducing the cost and effort of creating and using hypermedia-oriented interfaces. Generalized hypermedia adds value by providing a hypermedia-style interface to a DSS application without an *author* having to create any nodes or links. (Of course, users can add their own comments and other annotations, just as in regular hypermedia systems.) The direct access to information provided by a generalized hypermedia DSS makes it easy to use for someone who is new to a decision domain. He can explore the domain at his own pace, read the comments and definitions, and experiment with application models and data. Another benefit of hypermedia is the degree of user control. More advanced users easily can bypass information with which they are familiar.

Generalized hypermedia can be applied to any domain that is well enough understood and expressible for the systems programmer building an application manager subsystem to map the components of its application knowledge bases to generalized hypermedia components with bridge laws.

What about domains that are not well-structured? Raymond and Tompa present an interesting example — the natural language found

within a dictionary[58]. Given the current state of natural language processing research, it would be impractical to write bridge laws for such a domain.

In a way, we view generalized hypermedia bridge laws as a method for identifying what is 'important' in the applications that the application manager subsystem supports, by specifying which objects the DSS analysts and users can access. There is, of course, an inherent danger in the builder of the application manager anticipating what future users will want to access. This is an important issue, which we intend to study as we gain more experience with users. Even so, we believe that the preliminary bridge laws that our prototype Max currently incorporates yield a substantial — and satisfactory — degree of access to DSS information and operations.

Our research continues on several fronts. One is to improve our model of generalized hypermedia. Another is to explore the potential of *task environments* — local environments organized around the procedure the analyst should follow when he is performing an individual task[4]. Imagine, for example, a hypermedia-oriented project management system, in which each subtask has its own filter settings identifying information relevant to it (data, documents, reports, models). The task environment would tailor the user's view to his specific goal (and perhaps to his experience, skills or preferences), giving access only to information and commands pertaining to the current subtask. Task environments also could serve as an enhanced form of documentation, building upon the notion of, e.g., *guided tours* in NoteCards[34] or Zellweger's *scripted documents*[35]. An advanced implementation of task environments could incorporate 'active' agents that guide the user through the process, reminding him of steps left out and making suggestions. As with the other decision models supplied by applications, a task environment could be declared using bridge laws and filters, and then managed by the generalized hypermedia interface on behalf of an application.

Puttress and Guimaraes state that hypermedia integration 'must be accomplished without requiring major changes to the existing environment'[53] Application manager builders should have to write as few bridge laws as possible and not have to modify their system otherwise. We intend to help builders by constructing a bridge law editor that accepts formats other than first order logic.

The overwhelming majority of information systems today are dynamic systems that do not utilize hypermedia. We view this as an opportunity! To meet this opportunity we currently are generalizing our

model of the hypermedia engine to serve applications other than DSS[37]. We hope that our work encourages more developers to tackle dynamic models of hypermedia. Hypermedia should be a widely-implemented paradigm for information presentation. We challenge other hypermedia developers to help us make this possible.

ACKNOWLEDGEMENT

Steve Kimbrough of the University of Pennsylvania has played an integral role in the development of these ideas. It is he who originally applied the concept of bridge laws to hypermedia. Charles Hardwick of the University of Houston - Clear Lake invited me to write this article's original version, for which I am grateful. Patricia Carlson of the U. S. Air Force's Human Resources Training Laboratory at Brooks Air Force Base, Mark Frisse of Washington University, Chuck Kacmar of Florida State University, Tomás Isakowitz of New York University, Dick Maffei of Boston College and Merrill Warkentin of George Mason University each made voluminous, invaluable suggestions for previous drafts. I also found the comments of the anonymous referees most helpful. This work was motivated and supported in part by the U. S. Coast Guard, under contract DTCG39-86-C-E92204 (formerly DTCG39-86-C-80348), Steven O. Kimbrough principal investigator.

AUTHORS NOTE

This article expands an earlier version entitled 'Automating hypertext for decision support' presented at the Hypermedia and Information Reconstruction Conference sponsored by the University of Houston - Clear Lake in December 1990. An older draft was entitled 'Hypertext in real time: generating hypertext dynamically within a decision support system.'

TECHNICAL APPENDIX 1: INSIDE A STATIC HYPERTEXT KNOWLEDGE BASE

This appendix — referenced during our discussion of bridge laws — presents a systems programmer's view of the nodes, link markers and links in the static hypertext knowledge base underlying Figure 1a's and Fig. 1b's reports. Each report, link marker and connection is represented by a 'node', 'marker' or 'link' *predicate*. Every time a user manually creates a document or link through the DSS interface, the system generates corresponding instances of these predicates in the system's knowledge base. We precede each set of predicates with the meaning of the predicates' arguments.

Format: *node(identifier, report title, list of the text and link markers comprising its contents)*

```
node(1, title("Final Committee Report"),
    content(["To:<t>John Doe, Vice President of    Finance<cr>From:<t>", marker(1),
    "<cr>Subject:<t>", marker(2),
    "<cr>Date:<t>October 7, 1990<cr><cr><t>This Committee was to evaluate...",
    ...]))
node(2, title("explain($13,287,432)"), content([marker(18), "is the", marker(19),
    "under", marker(20), ...]))
node(3, title("explain($410,527.53)"), content([marker(46), "is the", marker(47),
    "under", marker(48), ...]))
node(4, title("origin($38,995.95)"),
    content([marker(66), "is the cost of each vehicle (", marker(67), ") in the
    scenario", ...])) ... etc.
```

Format: *marker(marker identifier, the link associated with the marker, marker's display value)*

```
marker(1, link(1), content("Procurement Committee"))
marker(2, link(2), content("Final Report Regarding New Fleet Configuration"))
marker(3, link(3), content("bases"))
marker(4, link(4), content("greater region"))
marker(5, link(5), content("(1)"))
marker(6, link(6), content("10"))
marker(7, link(7), content("$13,287,432")) ... etc.
```

Format: *link(link identifier, the report where the link originates, the link's destination report)*

```
link(1, node(1), node(5))
        • • •
link(6, node(1), node(10))
link(7, node(1), node(2)) ... etc.
```

TECHNICAL APPENDIX 2: GENERALIZED HYPERMEDIA BRIDGE LAWS

This appendix presents simplified examples of bridge laws, which the builder of the application manager subsystem specifies using logical predicates and logical quantification. The hypermedia engine uses these to generate the hypermedia links emanating from all calculations in interactive documents that result from executing DSS application models, such as the execution result $13,287,482 that the user selects in Fig. 2. The bridge laws themselves invoke both standard application manager routines and other bridge laws. For clarity, we precede bridge

law names with the code 'ght.' We precede the standard application manager routines with the code 'appl.' We describe additional bridge laws for models, scenarios, keywords, user links, etc. in other papers[4,12]. We also have applied bridge laws to the structure of a relational database[41].

Format of the generalized hypermedia link predicate

The hypermedia engine uses generalized hypermedia links of the following format. All bridge laws for links must have this format as well. (Note that we have simplified the link format for this discussion. The true format[4] allows an arbitrary set of link attributes as well as a destination link marker to highlight upon "arrival" at the link destination[59].)

```
ght_link(id, originating node, originating link marker, destination node, link
computation operation, valid filter set)
```

Bridge law declaring links to destination nodes from markers representing DSS model execution results

This bridge law establishes links from model execution results (i.e., calculations). The hypermedia engine calls it whenever the user selects a link marker in an interactive document. The bridge law determines whether the link marker represents a calculation resulting from executing a DSS application model. If so, it maps a link from the execution result to three kinds of destination nodes — an explanation report, a reevaluation report and a listing of user comments about the calculation, if any exist — in the following manner. Calling this bridge law causes the hypermedia engine, in turn, to invoke its component predicates: the application routine *appl_type/2*, which identifies the type of link marker selected; the bridge law *ght_operation/4*, which determines the link operation type *op*; and bridge law *ght_identifier/4*, which generates the name of the destination node *dest*. Often the link operation is a computation that generates the contents of the destination node during traversal. Arguments with underscores accept any values passed. An underscore for the originating node enables this bridge law to map links in all originating nodes, i.e., from any calculation in any interactive document!

```
(∀ dest, f1, f2, id, op) (ght_link(id, _, marker(id), dest, op, filter([f1, f2])) <—
    (appl_type(id, type(execution_result)) &
     ght_operation(type(execution_result), id, op, filter(f1)) &
     ght_identifier(id, op, dest, filter(f2))))
```

Identifying the type of application element selected as an execution result

We assume that an application (or, in our case, the application manager subsystem) can identify its own objects from the objects' identifiers[53].

The appl_type/2 predicate determines that an object with the identifier id is a model execution result if it satisfies the *appl_execution/4* predicate. For simplicity we shall not describe the arguments with underscores. We provide a full description elsewhere[4].

```
(∀id) (appl_type(id, type(execution_result)) <—
    (appl_execution(id, _, _, _)))
```

Bridge laws inferring link operation types for model execution results and all elements with comments

The first two bridge laws find 'explain' and 're-evaluate' links for model execution results, respectively. Each bridge law retrieves the filter set under which this link is valid using the predicate *appl_operation_available/3*. This enables the application manager's builder to specify filters to restrict the availability of particular DSS operations.

```
(∀f) (ght_operation(type(execution_result), _, operation_type(explain), filter(f)) <—
    (appl_operation_available(type(execution_result), operation_type(explain),
    filter(f))))

(∀f) (ght_operation(type(execution_result), _, operation_type("re-evaluate"), filter(f)) <—
    (appl_operation_available(type(execution_result), operation_type("re-evaluate"),
    filter(f))))
```

The next bridge law determines whether any comments exist for the specific calculation the user has selected. The system may restrict access to specific comments through the filter associated with each.

```
(∀f, id) (ght_operation(_, id, operation_type("show comment"), filter(f)) <—
    (user_comment(id, _, filter(f))))
```

Bridge laws inferring destination node identifiers

Given the link operation type, these bridge laws infer the identifier of the destination node. (The application manager subsystem will generate the contents of this node when the hypermedia engine actually traverses the link.) The destination identifiers — explain(id), reevaluate(id, m, s) and user_comment(id) — are passed in the third argument. In the first two laws, the application manager's builder uses the predicate appl_destination_available/3 to specify filters restricting access to particular DSS reports. If users should be able to access these operations and reports under all analysis conditions, then this predicate should return the filter code 'all.' We see an example of this in the third bridge law, which identifies the comment report.

```
(∀f, id) (ght_identifier(id, operation_type(explain), explain(id), filter(f)) <—
    (appl_type(id, type(execution_result)) &
     appl_destination_available(id, operation_type(explain), filter(f))))
```



```
(∀f, id, m, s) (ght_identifier(id, operation_type("re-evaluate"), reevaluate(id, m, s),
    filter(f)) <—
    (appl_type(id, type(execution_result)) &
     appl_execution_result_model(id, m)) &
     appl_execution_result_scenario(id, s)) &
     appl_destination_available(id, operation_type("re-evaluate"), filter(f))))

(∀id) (ght_identifier(id, operation_type("show comment"), user_comment(id),
    filter(all)))
```

REFERENCES

1 SPRAGUE, R. *and* CARLSON, E. *Building effective decision support systems.* Englewood Cliffs, NJ: Prentice-Hall, Inc., 1982.

2. TURBAN, E. *Decision support and expert systems: management support systems, 2nd Ed.* New York: Macmillan Publishing Company, 1990.

3. BALASUBRAMANIAN, P.R., ISAKOWITZ, T., JOHAR, H. *and* STOHR, E. Hyper Model Management Systems. *In: Proceedings of the Twenty-Fifth Hawaii International Conference on System Sciences*, Volume III, Kauai, HI, January 1992, 462-472.

4. BIEBER, M. *Generalized hypertext in a knowledge-based DSS shell environment.* Philadelphia, PA: University of Pennsylvania, Decision Sciences Department Ph. D. Dissertation, 1990.

5. BIEBER, M. *and* KIMBROUGH, S.O. On the logic of generalized hypertext. *Decision Support Systems* (forthcoming).

6. BIEBER, M. *and* KIMBROUGH, S.O. On generalizing the concept of hypertext. *Management Information Systems Quarterly* (forthcoming).

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

8. NIELSEN, J. *Hypertext and hypermedia.* San Diego CA: Academic Press Inc., 1990.

9. MINCH, R. Application and research areas for hypertext in decision support systems. *Journal of Management Information Systems, 6(3)*, 1989, 119-138.

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

11. SCHÜTT, H. *and* STREITZ, N. HyperBase: A hypermedia engine based on a relational database management system. *In*: A.RIZK, N. STREITZ, and J. ANDRÉ, *eds. Hypertext: concepts, systems and applications, proceedings of ECHT'90*, Versailles, November 1990. Cambridge: Cambridge University Press, 95-108.

12. BIEBER, M. Template-driven hypertext: a methodology for integrating a hypertext interface into information systems. Chestnut Hill, MA: Boston College, Computer Science Department Technical Report, 1991. (Technical Report #BCCS-91-3.)

13. COLLIER, G. Thoth II: hypertext with explicit semantics. *In*: Hypertext '87 Proceedings, Chapel Hill, NC, November 1987, 269-290.

14. HALASZ, F. and SCHWARTZ, M. The Dexter hypertext reference model. *In*: J. MOLINE, D. BENIGNI, and J. BARONAS, *eds. Proceedings of the hypertext standardization workshop*, Gaithersburg, MD, January 1990. National Institute of Standards, Special Publication SP500-178, 95-134.

15. BIEBER, M. Fundamentals of flexible hypermedia. Chestnut Hill, MA: Boston College, Computer Science Department Technical Report, 1992. (Technical Report #BCCS-92-5.)

16. SIMON, H. *The new science of management decision.* New York: Harper and Row, 1960, 1977 (revised ed.).

17. MINTZBERG, H., RAISINGHANI, D. *and* THEORET, A. The structure of unstructured decision processes. *Administrative Science Quarterly, 21*, 1976, 246-275.

18. KIMBROUGH, S.O. Notes on the argumentation theory for decision support Systems. *In: Proceedings of the 1990 International Society on DSS Conference*, Austin, TX, September 1990, 17-39.
19. KIMBROUGH, S.O., PRITCHETT, C., BIEBER, M. *and* BHARGAVA, H. The Coast Guard's KSS Project. *Interfaces, 20(6)*, 1990, 5-16.
20. SCHULER, W. *and* SMITH, J. Author's Argumentation Assistant (AAA): A Hypertext-based Authoring Tool for Argumentative Texts. *In*: A. RIZK, N. STREITZ, *and* J. ANDRÉ, *eds. Hypertext: concepts, systems and applications, proceedings of ECHT'90*, Versailles, November 1990. Cambridge: Cambridge University Press, 137-151.
21. MARSHALL, C., HALASZ, F., ROGERS, R. *and* JANSSEN, W. Aquanet: a hypertext tool to hold your knowledge in place. *In:* Hypertext '91 Proceedings, San Antonio, TX, December 1991, 261-275.
22. BERNSTEIN, B., SMOLENSKY, P. *and* BELL, B. Constraint-based hypertext system to augment human reasoning. *In: Proceedings of Rocky Mountain Conference on Artificial Intelligence '89*, Denver, CO, June 1989, 21-30.
23. SMOLENSKY, P., BELL, B., FOX, B., KING, R. *and* LEWIS, C. Constraint-based hypertext for argumentation. *In: Hypertext '87 Proceedings*, Chapel Hill, NC, November 1987, 215-246.
24. CONKLIN, J. *and* BEGEMAN, M. gIBIS: A Tool for All Reasons. *Journal of the American Society for Information Science, 40(5)*, 1989, 200-213.
25. FISCHER, G., MCCALL, R. *and* Morch, A. JANUS: Integrating hypertext with a knowledge-based design environment. *In: Hypertext '89 Proceedings*, Pittsburgh, PA, November 1989, 105-117.
26. MCCALL, R., BENNETT, P., D'ORONZIO, P., OSTWALD, J., SHIPMAN, F. *and* WALLACE, N. PHIDIAS: integrating CAD graphics into dynamic hypertext. *In*: A. RIZK, N. STREITZ, *and* J. ANDRÉ, *eds. Hypertext: concepts, systems and applications, proceedings of ECHT'90*, Versailles, November 1990. Cambridge: Cambridge University Press, 152-165.
27. BARMAN, D. RelType: relaxed typing for object-orientated hypermedia representations. *In*: Object-oriented programming in AI: Workshop Notes from the Ninth Annual National Conference on Artificial Intelligence (AAAI-91), Anaheim, CA, July 1991.
28. RITTEL, W. *and* KUNZ, W. Issues as elements of information systems. Berkeley, CA: Center for Planning and Development Research, University of California, 1970. (Working Paper #131.)
29. GLUSHKO, R. Design issues for multi-document hypertexts. *In*: Hypertext '89 Proceedings, Pittsburgh PA, November 1989, 51-60.
30. BLANNING, R., WHINSTON, A., DHAR, V., HOLSAPPLE, C., JARKE, M., KIMBROUGH, S., LERCH, J. *and* PRIETULA, M. Model management systems. *Forthcoming in*: B. KONSYNSKI, *and* E. STOHR, *eds. Information systems in decision processes: charting new directions for DSS research: A multidisciplinary approach.* (Chapter 7).
31. HERRSTROM, D. *and* MASSEY, D. Hypertext in Context. In: E. BARRETT, *ed., The society of text: hypertext, hypermedia, and the social construction of information.* Cambridge, MA: MIT Press, 1989, 45-58.
32. KOVED, L. *and* SHNEIDERMAN, B. Embedded menus: selecting items in context. *Communications of the ACM, 29(4)*, 1986, 312-318.
33. BHARGAVA, H., BIEBER, M. *and* KIMBROUGH, S.O. Oona, Max, and the WYWWYWI principle: generalized hypertext and model management in a symbolic programming environment. *In: Proceedings of the Ninth International Conference on Information Systems*, Minneapolis, MN, November 1988, 179-92.
34. MARSHALL, C. *and* IRISH, P. Guided tours and on-line presentations: how authors make existing hypertext intelligible for readers. *In: Hypertext '89 Proceedings*, Pittsburgh, PA, November 1989, 15-42.
35. ZELLWEGER, P. Scripted documents: A hypermedia path mechanism. *In: Hypertext '89 Proceedings*, Pittsburgh, PA, November 1989, 1-14.

36. KIMBROUGH, S.O. On shells for decision support systems. Philadelphia, PA: University of Pennsylvania, Department of Decision Sciences, 1986. (Working paper #86-07-04.)
37. BIEBER, M. Issues in modeling a 'dynamic' hypertext interface for non-hypertext information systems. *In: Hypertext '91 Proceedings*, San Antonio TX, December 1991, 203-218. An expanded version of this paper available as: BIEBER, M. On merging hypertext into dynamic, non-hypertext systems. Chestnut Hill, MA: Boston College, Computer Science Department, 1991. (Technical Report #BCCS-91-14.)
38. NAGEL, E. *The structure of science: problems in the logic of scientific explanation.* New York: Harcourt, Brace and World, Inc., 1961.
39. HAUGELAND, J. The nature and plausibility of cognitivism. *The Behavioral and Brain Sciences, 1* 1978, 215-26; reprinted with minor revisions in Haugeland J., *ed. Mind design: philosophy, psychology, artificial intelligence.* Cambridge, MA: The MIT Press, 1981.
40. KIMBROUGH, S.O. On the reduction of genetics to molecular biology. *Philosophy of Science, 46(3)*, 1979, 389-406.
41. BIEBER, M. *and* ISAKOWITZ, T. Bridge laws in hypertext: A logic modeling approach. Chestnut Hill, MA: Boston College, Computer Science Department, 1991. (Technical Report #BCCS-91-4.)
42. BARLOW, J., BEER, M., BENCH-CAPON, T., DIAPER, D., DUNNE, P. *and* RADA, R. Expertext: hypertext-expert system theory, synergy, and potential applications. In: N.SHADBOLT, *ed., Research and Development in Expert Systems VI: Proceedings of Expert Systems 89, the Ninth Annual Technical Conference of the British Computer Society Specialist Group on Expert Systems,* London, September 1989. Cambridge: Cambridge University Press, 116-127.
43. AKSCYN, R., MCCRACKEN, D. *and* YODER, E. KMS: A distributed hypermedia system for managing knowledge in organizations. *Communications of the ACM, 31(7)*, 1988, 820-835.
44. BROWN, P.J. A hypertext system for UNIX. *Computing Systems, 2(1)*, 1989, 37-53.
45. APPLE COMPUTER, INC. *HyperCard User's Guide.* Cupertino, CA, 1989.
46. SCHNASE, J. *and* LEGGETT, J. Computational hypertext in biological modeling. *In: Hypertext '89 Proceedings*, Pittsburgh, PA, November 1989, 181-198.
47. MAYES, J.T., KIBBY M.R. *and* WATSON, H. StrathTutor: the development and evaluation of a learning-by-browsing system on the Macintosh. *Computers and Education, 12*, 1988, 221-229.
48. FURUTA, R. *and* STOTTS, P.D. The Trellis hypertext reference model. *In*: J. MOLINE, BENIGNI, D. *and* J. BARONAS, *eds., Proceedings of the Hypertext Standardization Workshop*, Gaithersburg, MD, January 1990. National Institute of Standards, Special Publication SP500-178, 83-94.
49. SHERMAN, M., HANSEN, W., MCINERNY, M. *and* NEUENDORFFER, T. Building hypertext on a Multimedia Toolkit: an overview of Andrew Toolkit Hypermedia Facilities. *In*: A. RIZK, N. STREITZ, *and* J. ANDRÉ, eds., *Hypertext: concepts, systems and applications, proceedings of ECHT'90*, Versailles, November 1990. Cambridge: Cambridge University Press, 13-24.
50. MAURER, H. *and* TOMEK, I. Broadening the scope of hypermedia principles. *Hypermedia 2(3)*, 1990, 201-220.
51. KACMAR, C. *and* LEGGETT, J. PROXHY: A process-oriented extensible Hypertext Architecture. *ACM Transactions on Information Systems, 9(4)*, 1991, 399-419.
52. PEARL, A. Sun's link service: a protocol for open linking. *In: Hypertext '89 Proceedings*, Pittsburgh PA, November 1989, 137-146.
53. PUTTRESS, J. *and* GUIMARAES, N.M. The toolkit approach to hypermedia. In: A. RIZK, N. STREITZ, *and* J. ANDRÉ, *eds., Hypertext: concepts, systems and applications, proceedings of ECHT'90*, Versailles, November 1990. Cambridge: Cambridge University Press, 25-37.
54. UTTING, K. *and* YANKELOVICH, N. Context and orientation in hypermedia networks. *ACM Transactions on Information Systems, 7(1)*, 1989, 58-84.

55. GARG, P. *and* SCACCHI, W. Ishys: designing an Intelligent Software Hypertext System. *IEEE Expert, 4(3)*, 1989, 52-64.
56. CAMPBELL, B. *and* GOODMAN, J. HAM: A general purpose Hypertext Abstract Machine. *Communications of the ACM, 31(7)*, 1988, 856-861.
57. NEWCOMB, S.R., KIPP, N.A. *and* NEWCOMB, V.T. The 'HyTime' hypermedia/time-based document structuring language. *Communications of the ACM, 34(11)*, 1991, 67-83.
58. RAYMOND, D. *and* TOMPA, F. Hypertext and the Oxford English Dictionary. *Communications of the ACM, 31(7)*, 1988, 871-879.
59. LANDOW, G. Hypertext in literary education, criticism, and scholarship. *Computers and the Humanities, 23*, July 1988, 173-198.

(Accepted for publication March 1992)