Decision Support Systems 14 (1995) 251–267

On integrating hypermedia into decision support and other information systems [1,2]

Michael Bieber *

Computer and Information Science Department, New Jersey Institute of Technology, Newark, New Jersey, USA

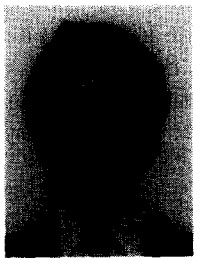
The goal of this research is to provide *hypermedia* functionality to all information systems that interact with people. Hypermedia is a concept involving access to information, embodying the notions of context-sensitive navigation, annotation and tailored presentation. This paper presents the architecture of a system-level *hypermedia engine*, designed both to manage full hypermedia functionality for an information system and to bind interface-oriented *front-end* systems with separate computation-oriented *back-end* systems. The engine dynamically superimposes a hypermedia representation over a back-end application's knowledge components and processes. The hypermedia engine generates this representation using *bridge laws*, which capture the internal structure of client systems. Users access the application through its hypermedia representation. The paper also describes a set of minimal requirements for integrating the hypermedia engine with an information system. These guidelines apply to all integration efforts, not just that described here. Information systems will require some supplementary routines for the engine to manage hypermedia functionality for them. The more sophisticated and cooperative the information system, the higher the level of hypermedia support the engine will provide.

Keywords: Bridge laws; Decision support systems; Filters; Hypertext; Hypermedia; Hypermedia engine; Information navigation; Information systems architecture; Integration

1. A vision of hypermedia and information systems

We envision a world in which information increasingly empowers people. Decision makers, analysts, researchers, trainees, students and casual browsers all will have access to information they need or desire, in a format tailored to their individual tasks and personal preferences.

The concept of *hypermedia* embraces the spirit of such access to information and eventually, we believe, will be incorporated in the interfaces of all decision support systems (DSS), and indeed, all information systems that interact with people. (Various authors, e.g., [43], support this prediction.) Our research goals are to facilitate this integration and to produce tangible results. Once an information system includes hypermedia functionality, the specific applications it supports (e.g., worksheets within a spreadsheet package, models within a linear programming package and expert systems within an expert system shell) automatically become hypermedia applications. Users communicate in hypermedia's direct, context-sensitive fashion and hypermedia functions supplement the system's original commands.



Michael Bieber is Assistant Professor in the Department of Computer and Information Science at New Jersey Institute of Technology. He currently is Visiting Assistant Professor of Information Systems at New York University. His research interests include hypermedia, decision support and information presentation, and has applied this research on a team developing knowledge-based decision support systems (DSS) for the U.S. Coast Guard. He earned his Ph. D. in Decision Sciences from the Wharton School, University of Pennsylvania. Before entering academia, he worked as a systems analyst for IBM and the University of Pennsylvania.

* Corresponding author. Email: bieber@cis.njit.edu

[1] I wish to thank Tomás Isakowitz, Steve Kimbrough and Robert Minch for their invaluable help with this paper and its concepts. This research 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.

[2] This expands an earlier version: M. Bieber, Providing Information Systems with Full Hypermedia Functionality, in: Proceedings of the Twenty-sixth Hawaii International Conference on System Sciences (Wailea, Jan. 1993).

SSDI 0167-9236(93)E0026-A

The goal of this paper is to encourage an ongoing discussion about providing the users of all information systems with dynamic hypermedia functionality. We began this discussion in [7,8] by proposing a solution – a hypermedia engine that builders can integrate with their systems. From this we derived a starting set of minimal requirements for hypermedia integration, which we believe apply to all integration efforts, not just our own. This paper extends the architecture we originally introduced in [7,8]. Here we deepen our description of the hypermedia engine's internal structure, develop an alternate architecture for information systems not abandoning their interfaces and expand our set of minimal requirements for hypermedia integration.

In section 2 we briefly review the concepts of hypermedia and our enhancement, *generalized hypermedia*. Generalized hypermedia is at the heart of our hypermedia engine's architecture. We also examine the potential role of hypermedia in decision support. In section 3 we introduce two versions of our engine's architecture and describe its internal structure. We illustrate its operation with a detailed example. In section 4 we discuss the minimal requirements for hypermedia integration – the commitment information system builders have to make to use our architecture. We conclude in section 5 by briefly comparing our work with other current approaches.

2. Hypermedia and generalized hypermedia

Hypertext [3,15,49,53,54,67] is the concept of specifying relationships among pieces of information and providing computer-mediated navigation among them. For example, we can automatically *link* a document with a stage in a decision analysis, a keyword with its definition and a calculation with its explanation. *Hypermedia* expands this concept to include media other than text. We refer to the information at either end of the link as *nodes*, and to the entire node and link structure as a hypermedia *network*. We signal the existence of a link from a node by highlighting a portion of the node's display contents, which we call a *link marker*. When a user selects a link marker the system *traverses* this link and displays an appropriate representation of the destination node. Fig. 1 shows a hypermedia-oriented *interactive document* similar to those our Max prototype produces [11,38]. This document node represents a report generated by a decision support system (DSS) and passed on to our *hypermedia engine* for display. The underlined and boldfaced text strings are link markers, each associated with one or more links. In Fig. 1 the user has selected the marker "$60.00" representing the result of a DSS calculation. The hypermedia engine inferred three links associated with this marker's underlying calculation: to a node representing an expert system explanation, to a node representing its dynamic recomputation and to a node containing user comments about it. The two remaining links represent hypermedia engine commands for annotating elements of the DSS. The user navigates through the DSS thus, by selecting some item of interest and traversing a link representing an appropriate DSS (or hypermedia engine) command. The hypermedia engine would support other types of information systems in a similar fashion.

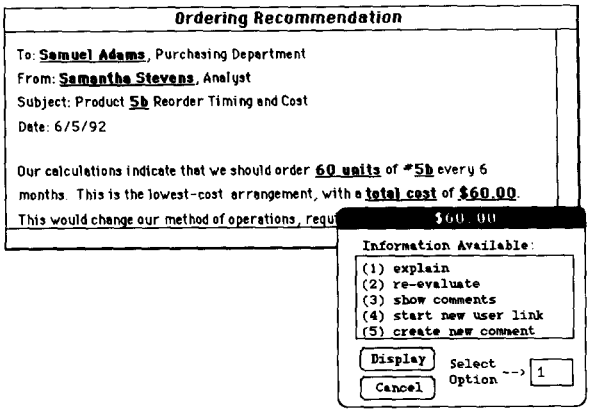


Fig. 1. Accessing application and hypermedia functionality in a hypermedia-style interface.

Hypermedia embodies a methodology of flexible access to information incorporating the notions of navigation, annotation and tailored presentation. Tailoring is inherent in other hypermedia functions, e.g., in customizing the network the user navigates and its annotations. Together, these features constitute what we call "full hypermedia functionality," an ideal level of functionality that few of today's hypermedia systems achieve. (Many systems calling themselves "hypermedia systems," in fact, provide only forward

navigation – i.e., direct manipulation – and perhaps commenting [42,62].)

Users navigate "forward" by selecting an item of interest (a link marker) about which to retrieve comments, annotations, definitions, explanations or any other inferable information. Link markers act as embedded menus [39], giving "context-sensitive" access to an underlying application's knowledge and operations. We have dubbed this the *WYWWYWI ("what you want, when you want it") principle* [5]. Users normally traverse from node to node at the detail level, i.e., with each node occupying a window on the screen. Users also should be able to navigate via (graphical) overviews [21,40,49,54,69] of the hypermedia network. Overviews (often – see [17]) help alleviate the *network disorientation* [15,54] associated with hypermedia's nonrestrictive, user-directed access.

Information retrieval-style queries provide an alternative method of forward navigation [20,23, 72,71]. Queries return a relevant subset of an application's components, which is mapped to a hypermedia representation. Users then can navigate within this tailored subenvironment.

Users can navigate "backwards" as well, returning to prior stages or "screens" in their analysis, i.e., the previously visited computer screens, but in their current state. *Backtracking* is another important weapon against network disorientation. By providing an escape mechanism for returning to familiar territory, backtracking gives users the confidence to explore freely and take "detours."

Annotation comprises features such as user-declared links and comments. Analysts and instructors can use these, for example, to tie specific data, techniques and results together in *trails* [68] or *guided tours* [24,46]. Trails and guided tours both direct and constrain forward navigation. They can document analyses or serve as tutorials, and can be tailored for specific users or tasks. In a DSS, for example, annotations can provide justification for courses of action [10].

Full hypermedia functionality can augment decision support [32]. Consider Simon's *Intelligence* (gathering information) - *Design* (developing alternate solution scenarios) - *Choice* (choosing a solution) framework of decision making [64]. An analyst navigates or *browses* through the documents, models, and data in the problem domain by making queries or by directly selecting items of interest ("drilling down" [49]). If the domain had been explored previously, the hypermedia representation may be tailored to this analyst, or his or her task. Prior users also may have specified annotations or recommended paths. Our analyst's own annotations document the solution scenarios he develops, and connects these to their sources and other supporting information. DSS commands mapped to hypermedia links enable the analyst to evaluate models and data directly through context-sensitive hypermedia navigation. As we see in Fig. 1, this provides a seamless interface for all DSS and hypermedia functions. The analyst can link his chosen solution scenario to documents reporting the decision. He also could package his analysis as a trail or guided tour. Hypermedia thus serves as a documentation and justification tool [43]. In fact, an entire class of hypermedia systems – argumentation systems, e.g., [16,45] – specializes in capturing the decision rationale and deliberations so often unrecorded and quickly forgotten. Through the hypermedia representation, others can explore the analyst's alternatives and conclusions, and can comment upon them. (See [10] for a deeper exploration of hypermedia and decision support and [31,49] for discussions of hypermedia and decision support research issues.)

In summary, hypermedia is a technique for providing direct, context-sensitive access to application data, the commands that manipulate this data, and *metainformation* about the data and commands. Such access should improve the quality and users' understanding of applications and their inputs and outputs, and increase the confidence people have in these. Performance issues aside, we believe that most information systems that interface with people would profit from hypermedia functionality. (The exceptions may be data entry and other transaction processing systems, in which users do not query information. Perhaps even here, validation and other feedback may benefit from a hypermedia representation.)

There are two basic limitations with most of today's "first generation" hypermedia systems. First, they implement a static and explicit model of hypermedia; the nodes, links and link markers must be declared explicitly and be fully enumerated (as opposed to being declared virtually and generated upon demand). Most applications, however, are dynamic and too large to mark up

manually. Imagine a spreadsheet designer having to calculate all what-if analyses in advance. Second, most of today's hypermedia systems are "... *insular monolithic packages* that demand the user disown his or her present computing environment to use the functions of hypertext and hypermedia" [48]. Users who want hypermedia functionality often must abandon the software they currently use – an impractical restriction [34,43]. The first limitation motivated us to develop *generalized hypertext* or *generalized hypermedia* [6,11,12]. The second motivated our hypermedia engine, which will provide hypermedia functionality to an information system's applications. The engine incorporates our dynamic model of generalized hypermedia.

In generalized hypermedia we broaden the underlying model of hypermedia components – nodes, links, link markers, etc. – with three of Halasz' proposed extensions to hypermedia [27]: virtual specifications, dynamic computation, and filtering or *tailoring*. We use these to generate a hypermedia representation "on the fly" from basic declarations we call *bridge laws* that describe the internal structure of an information system. As we shall see in section 3.2, bridge laws enable generalized hypermedia to superimpose a hypermedia network on an information system's application, generating all node, link and link marker representations dynamically from the application's original, non-hypermedia data or knowledge base. We should emphasize at this point that both generalized hypermedia and bridge laws are to be applied to an entire information system application *package* (such as a spreadsheet language, a DSS shell or a programming language), not *individual* applications (e.g., a specific worksheet, DSS or program module). Once declared for an entire package, hypermedia functionality will be provided automatically for each of its individual applications. (This is analogous to Garzotto et al.'s *authoring in the large* [25].)

Three aspects combined distinguish generalized hypermedia from other hypermedia approaches: (1) all mapping and computation in generalized hypermedia is dynamic; (2) through bridge laws, generalized hypermedia can provide system-level support to any information system with a well-defined internal structure; and (3) bridge laws map a hypermedia representation without altering an information system's data or knowledge bases. No other approach supports all three criteria [10]. This does not mean that information system builders simply can plug in our hypermedia engine without adjusting their systems. Each builder will have to declare a small set of bridge laws, register the system's communication protocols and add a relatively small number of routines to his system to route information formatted for these bridge laws to the hypermedia engine. This will suffice to provide hypermedia engine support for all specific applications written in his information system. Builders, however, will not have to make their systems or applications "hypermedia-aware" in any way. This is because (1) as mapped representations, nodes, links and link markers do not alter the original, underlying application information and (2) the hypermedia engine maintains all other hypermedia constructs (e.g., comments and trails) in its

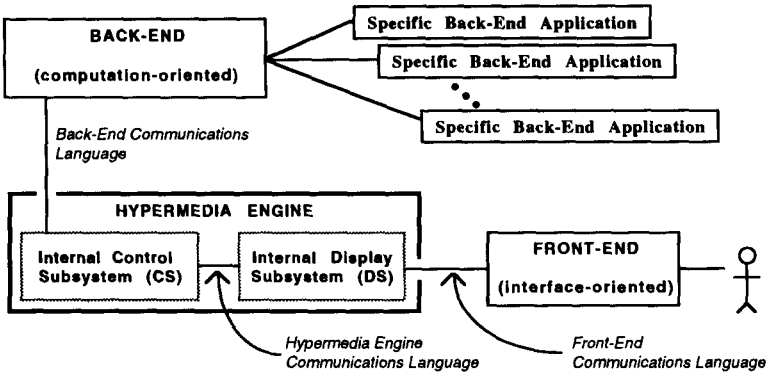


Fig. 2. Hypermedia engine architecture (version 1): This architecture binds independent back-end and front-end information systems.

own knowledge bases separate from its client information systems. The engine adds no hypermedia constructs to its client systems or their applications.

3. The system-level hypermedia engine

Fig. 2 shows a version of our proposed hypermedia engine's architecture that binds independent back-end and front-end information systems. By *back-end* systems, we mean information systems that primarily provide computation functionality, such as DSS, expert systems, intelligent tutoring systems, database management systems, project management systems, etc. By *front-end* systems we mean those that primarily support interface-level functionality such as word processors and graphics packages. Instead of being tightly coupled, the hypermedia engine runs concurrently with – and independent of – the information systems it binds, communicating through external message passing. The engine embeds link markers in messages the back-end passes to the front-end for display and handles requests for back-end functionality or supplementary hypermedia support when a user selects one of these markers. As a result, the user can access a back-end through the interface of his or her choice, which now provides full hypermedia functionality. (This assumes that the front-end and back-end builders comply with the requirements we discuss in section 4.)

This architecture also allows users to access multiple back-end systems at once and incorporate information (linked objects) from different back-ends in a single front-end document [58]. Eventually this architecture will support workgroups of multiple simultaneous users on heterogeneous front-ends.

Many computation-oriented information systems, of course, have high-quality interfaces. Among these are spreadsheets and CAD systems, as well as specific cases of the aforementioned front-end and back-end systems. A second version of the hypermedia engine, shown in Fig. 3, would run concurrently with such systems and manage hypermedia functionality for them. In this architecture, internal communications between the interface and computation modules must be routed through the hypermedia engine.

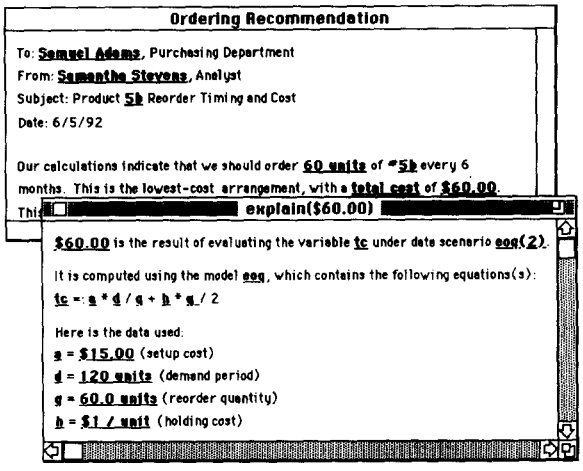


Fig. 3. Hypermedia engine architecture (version 2): This architecture serves information systems that have both adequate interface and computation functionality.

For the rest of this paper we shall use the terms "front-end" and "back-end" to indicate interface-oriented and computation-oriented functionality in both versions of the architecture.

3.1. An overview of the paper's example

We describe the hypermedia engine's architecture through Fig. 1's simple text-based example both here and in section 3.5. (Our model also supports non-text content and link markers.) Fig. 1's interactive document entitled "Ordering Recommendation" started as a message from the DSS back-end. As an illustration, suppose the second sentence of that message had the following format:

'... This is the low-cost arrangement, with a ⟨*variable(tc), "total cost"*⟩ of ⟨*calculation(variable(tc), model(eoq), scenario (eoq(2))), 60, currency(US)*⟩...'

Italicized text within angle brackets denotes a back-end object. The back-end tagged each object with its display value, any relevant formatting information and an internal identifier. The hypermedia engine superimposed a hypermedia structure over the entire message and converted its contents to a *document component set* for display by the front-end. (The document component set contains the message contents after the hypermedia engine has filtered them and embedded hy-

permedia link markers.) As part of the conversion the hypermedia engine added the identifier of the owning back-end, "DSS1," to each object's tag along with a unique hypermedia engine identifier for distinguishing among multiple instances of a back-end object. Assume the corresponding portion of document component set had the following internal format:

'... This is the low-cost arrangement, with a ⟨*[6, DSS1, variable(tc)], value("total cost"), form(text)*⟩ of ⟨*[7, DSS1, calculation(variable(tc), model(eoq), scenario(eoq(2)))], value(60), form-(currency(US))*⟩...'

When the user selected the link marker "$60.00," the hypermedia engine managed the process of gathering all possible links to the underlying object, "calculation(variable(tc), model (eoq), scenario(eoq(2)))," which is owned by the back-end system "DSS1." We see the resulting *link ensemble* representing two back-end commands and three hypermedia engine commands in Fig. 1. Now the user chooses link #1. In traversing this link the hypermedia engine invokes DSS1's explanation generator, which returns the explanation as a message. The engine converts this to the document component set displayed as "explain($60.00)" in Fig. 4.

In the following sections we examine different aspects of the hypermedia engine and then return to this example in further detail.

3.2. Bridge laws and filters: techniques for automating hypermedia

In this section we discuss filters and bridge laws. As part of compiling the document component set, the hypermedia engine must determine the locations (i.e., infer the existence) of link markers in back-end messages. Bridge laws enable this inference. Filters tailor it.

The hypermedia engine uses filters to customize the user's interaction in many ways. For example, filters can direct:

- which report form or *template* the engine uses to construct a document component set from back-end messages,
- how detailed to make report contents,
- which objects to represent as link markers for the user's current task, and
- which links to prune to avoid overwhelming a novice user.

Through filtering, the hypermedia engine can assume responsibility of managing *mode* or *task* changes, altering the available commands and documents as the user navigates through the back-end. For example, in a project management system the hypermedia engine would use filters to tailor the user's view to his or her current project subtask. For more details see [9], as well as the discussion of "contexts" in [6].

The hypermedia engine uses logical rules called bridge laws to map a hypermedia representation over the components of a back-end system. We adopted the term "bridge law" [29,37,50] because these logical rules serve as a "bridge" or connection between objects defined in the language of the back-end (e.g., models, variables, calculations) and those in that of the hypermedia engine (e.g., nodes, links, link markers). Bridge laws employ *logical quantification*, i.e., they apply to every instance that satisfies the set of conditions specified. Logical quantification (i.e., specifying "for each" or the logical symbol "∀") enables individual laws to map entire classes of back-end objects to hypermedia components; the same bridge law will map every object in the application knowl-

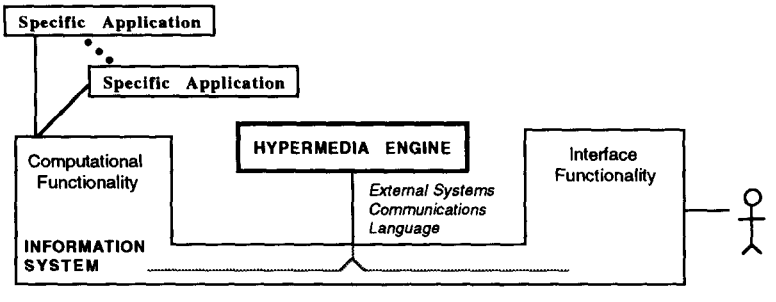


Fig. 4. The document "explain($60.00)": This hypermedia-style interactive document represents the back-end's explanation when the user selects the "explain($60.00)" link in Fig.1.

edge base that satisfies the bridge law's conditions.

In Fig. 1's example, the hypermedia engine used a bridge law similar to the following pseudo version to identify the object "calculation(variable(tc), model(eoq), scenario(eoq(2)))" within the "DSS1" back-end's original message and tag it as a link marker in the document component set.

For each calculation with attribute values satisfying the set of conditions Y and filter settings Z:
map a hypermedia link of type "explain" from the object to the DSS1 explain function, and
map a hypermedia link of type "re-evaluate" from the object to the DSS1 re-evaluate function.

As we shall discuss later, because it is specific to a particular back-end, the back-end's builder would have declared this bridge law. The hypermedia engine maintains its own set of general bridge laws that pertain to all back-ends. For example, the following general bridge law finds objects with comments registered in the hypermedia engine's knowledge bases.

For each object with a user-specified comment that satisfies filter settings Y and access security specifications Z:
map a hypermedia link of type "comment" between the object and its user-declared comment.

The engine uses the following general bridge laws to infer keywords. The first finds keywords that a message's back-end has declared. The second searches for keywords that a user has registered.

For each phrase in the message matching a keyword registered by its back-end that satisfies filter settings Z:
map a hypermedia link of type "keyword" from the phrase to the back-end object it represents.

For each phrase in the message matching a keyword registered by a user that satisfies filter settings Y and access security specifications Z:
map a hypermedia link of type "keyword" from the phrase to the object it represents.

Together, generalized hypermedia and its bridge laws provide a logic-based *knowledge representation* that enables the hypermedia engine to reason about the components (models, data, commands, etc.) of the underlying information systems they map. For example, full hypermedia functionality includes both producing an overview of an application's components, and searching or querying over these components. As part of our research, we shall determine whether a complete set of bridge laws suffices for the engine to perform both structure search and content search [25,27], and generate a network overview. (Producing an overview for a static hypermedia network is not a trivial task (see, e.g., [69]). No one, as yet, has tackled overviews for virtual environments involving computation, which a hypermedia mapping for dynamic information systems would involve.)

Ours is not the only hypermedia knowledge representation. In addition to the argumentation-based hypermedia models mentioned earlier, several other knowledge representation approaches have appeared in the hypertext literature, e.g., Petri nets [65,66], structured object representation [35], schemata [25,32,45], object-oriented hypermodelling [41], HyperSets [57], De Bra et al.'s extensible data model [19] and high-level specification languages [61]. Other systems that make use of a knowledge representation include Hypermedia-based Argumentation DSS [30], Electronic Working Papers [17], MacWeb [51], IDE [33] and RelType [2]. In future papers we hope to compare implementations using bridge laws and a generalized hypermedia engine with systems using other knowledge representations.

The *browsing semantics* of the different systems also will influence this evaluation. The browsing semantics define the dynamic behaviour of a system and are constrained by its underlying knowledge representation [65]. In our model, the hypermedia engine incorporates the browsing semantics and, as we shall see, attempts to integrate them into the front-end's functionality (constrained by the front-end's level of compliance, as we discuss in section 4.1).

The hypermedia engine stores bridge laws and filter settings in knowledge bases belonging to its Internal Control Subsystem. For an in-depth discussion of bridge laws see [6,10,11].

3.3. Internal Control Subsystem (CS)

The hypermedia engine has two major components: the Internal Control Subsystem (CS) and the Internal Display Subsystem (DS). We describe the structure of each next and illustrate their interaction in section 3.5's example.

The CS performs all configuration-independent processing. It handles the communication link between the hypermedia engine and the back-end systems. Back-ends pass messages containing reports, queries and menus. From each message the CS compiles the configuration-independent contents of a document component set or query component set, which the CS passes to the Internal Display Subsystem.

In the future we intend to upgrade our hypermedia engine for a networked, multi-user environment. At that time we shall split the CS into two logical modules, a single global module and multiple local modules. The global module will keep track of information that users on all systems should be able to access. Security permitting, everyone should have access, for example, to public comments, informational links, keyword definitions and documents registered by any user.

Logically, the CS maintains the following knowledge bases, each containing facts and rules for a different domain of inferencing.

- *Hypermedia Knowledge Base* The "Hypermedia KB" contains all types of hypermedia information registered by users including keywords and the nodes they represent; comments, links and other annotations (e.g., bookmarks [56]), and guided tours and other trails. The hypermedia engine maintains these independent of any back-end elements upon which they are based. Back-end systems need no record of the user's hypermedia activities.
- *Back-End Knowledge Base* There is one "Back-End KB" for each back-end system that users can access. The Back-End KB contains network access information for each back-end, as well as its bridge laws, keywords, and any other information necessary to build messages for it and parse its responses. An early version of our TEFA model management system back-end prototype [5] provides an example of supplementary parsing information. TEFA prefixed the display text of its objects with an ampersand. Registering this format would enable the CS to strip the ampersand to make the display less confusing and to reinsert the ampersand in user requests it passes to TEFA.
  We note that [1] presents an alternative system architecture that insulates bridge laws as much as possible from changes to the engine or back-end. This architecture includes a separate bridge law manager between the hypermedia engine and the back-end.
- *Control System Knowledge Base* The "CSKB" contains general parameters and routines for communicating, and for processing messages and responses. Its contents include:
  – default and current settings for the hypermedia engine, including filter settings
  – the functionality behind the hypermedia commands (e.g., querying link markers, creating user-specified links and comments)
  – hypermedia engine bridge laws for mapping user-specified hypermedia elements such as comments to back-end objects
  – standard document templates – forms dictating the general content and layout of documents [6] that the engine uses to create document component sets (similar to *abstract containers* in the Trellis Hypermedia Reference Model [22])
  – standard query templates – forms dictating the general content and layout of queries that the engine uses to create *query component sets*
- *Active Knowledge Base* The hypermedia engine records all back-end and user-declared objects currently displayed on the front-end screen in the "Active KB." The CS uses this for dynamically updating the front-end's display when elements of the back-end, such as a stock price, change. (In a multi-user environment, this would be a global knowledge base representing the displays of all active front-end systems. One function this would facilitate is screen sharing among users on heterogeneous systems.)

3.4. Internal Display Subsystem (DS)

The DS has two major responsibilities. First, it translates the configuration-independent document component set for the specific front-end that will display it. Second, it provides whatever "behind the scenes" support its front-end needs

to provide hypermedia functionality. Logically, the DS maintains the following knowledge bases:

- *Session Knowledge Base* The DS stores all user actions and hypermedia engine responses in the "Session KB." From these the DS can tailor a session log for hypermedia-style backtracking and guided tours. The Session KB serves a role similar to that of the *history* component in the Dexter Hypertext Reference Model [28]. Saving the Session KB should enable users to halt a computer session and continue it at a later time. Depending on the detail of user interaction the front-end passes to the DS, the Session KB could support multiple-level *undo* and *redo* functionality [70] for both hypermedia commands and the front-end's own commands. A highly cooperative front-end would pass user actions down to the exact keystroke. This also would enable the DS to serve as a monitoring and experimentation tool for particular front-end and back-end systems and settings. Several researchers have called for such functionality in hypermedia systems (e.g., [13]).
- *Display Knowledge Base* The "Display KB" – analogous to the *session* component in the Dexter model – records all hypermedia objects displayed on the front-end. Depending on the level of hypermedia support the DS must provide, this can include an object's internal identifier, the actual content of the front-end representation and, as we shall explain later, even the object's location within the front-end's windows. The DS uses this to determine what the user has selected and whether the user has permission to alter or delete it. Altering a back-end object's content (e.g., a current stock price or the result of a calculation) can destroy its validity. The DS also uses this knowledge base to map menus and link ensembles to the commands they represent.
- *Front-End Knowledge Base* The "FEKB" contains the information the DS needs to communicate with a specific front-end. In it, the DS maintains protocol formats, current parameter settings and the internal routines for coordinating hypermedia support with the particular front-end. With this knowledge, the DS can

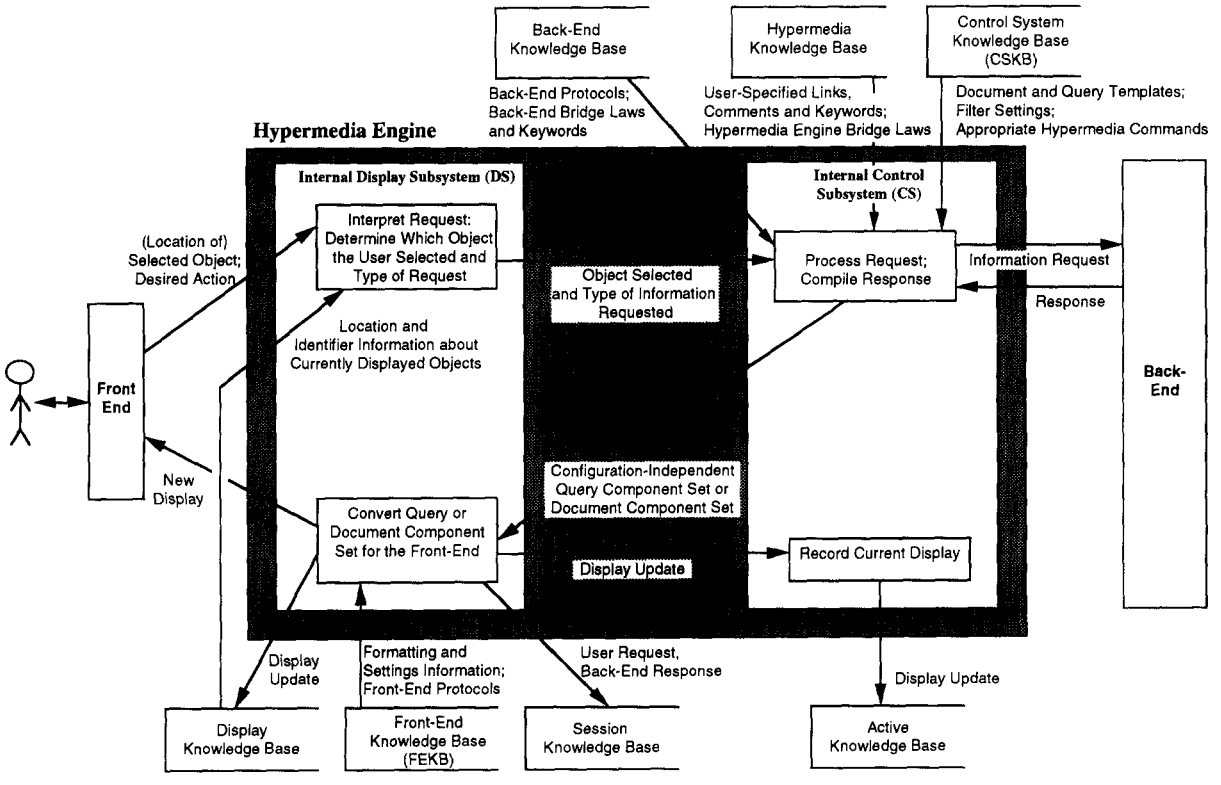


Fig. 5. Informal data flow diagram of the hypermedia engine: This diagrams the processing of a standard request for information about an application object when the user selects the link marker representing it.

translate the configuration-independent document and query component sets the CS passes for display, as well as the user requests the front-end passes.

Having introduced the hypermedia engine's modules and knowledge bases, we now can expand section 3.1's example illustrating a user request.

3.5. Example

Fig. 5 presents an informal data flow diagram depicting how the hypermedia engine compiled both the link ensemble in Fig. 1 for the link marker "$60.00" that the user selected and the interactive document "explain ($60.00)" in Fig. 4 resulting from traversing the "explain" link.

The hypermedia engine expects back-ends and front-ends to support two standard commands for all objects: generating a short description ("What is this?") and generating a list of possible actions ("What can I do with this?"). The user also should be able to select any command within a menu or represented by a link, and request assistance ("What happens if I do this?"). The front-end should provide some mechanism for the user to request each, e.g., a special keystroke combination or menu command. The back-end should provide the respective descriptions, command lists for its objects and assistance for its commands.

Another common action is an implicit request to edit. Because front-ends must ensure the integrity of link marker representations belonging to external systems, when users try to edit a marker's display value the CS must grant permission.

We now expand section 3.1's hypothetical example, following the flow of information illustrated in Fig. 5. Note that Fig. 5's diagram does not cover editing or requests for assistance. This discussion complements, but greatly deepens the illustration in [11].

1. The front-end passes a message to the DS in response to a user action

When the user selects a highlighted text string, such as the "$60.00" in Fig. 1, the front-end sends a message to the DS. The message from a "hypermedia engine-friendly" front-end – one that maintains external objects – will contain both the user's requested action ("What can I do with this?") and the object's internal identifier ("[7, DSS1, calculation(variable(tc), model(eoq), scenario(eoq(2)))]").

The less sophisticated the front-end, the more inferencing the DS must do to manage hypermedia functionality. For example, if the front-end does not maintain external objects then it may be able to pass only the selection's location in coordinates relative to the start of the document. In this case, the Display KB must maintain an up-to-date map of the front-end's documents that records the current location of all hypermedia objects. From this the DS must infer which object the user selected.

2. The CS processes the user request

The DS passes the action requested and the identifier of the selected object to the CS. From the identifier the CS can determine the system to which the object belongs. When a back-end owns the object, the CS compiles the appropriate request for the back-end. The CSKB supplies the back-end's protocols and access path. If the hypermedia engine owns the object, such as with user-specified keywords, the CS has all necessary information for the user request in its own knowledge bases. This also applies to hypermedia metainformation about back-end objects. Users may select user-specified comments and links associated with a back-end object and inquire about their creators, modification dates and even comments about these links and comments. The Hypermedia KB contains such metainformation.

We now detail three possible user requests: requests for (a) editing, (b) a short description and (c) a list of relevant commands.

2a. The CS processes the user request: edit link marker

For requests to edit a link marker's display value, the CS does not have to check with the back-end. The CSKB contains hypermedia engine-owned bridge laws controlling editing permission for each type of link marker. For example, users may delete, but not modify, back-end object markers. Users may alter a keyword, but the CS will cancel its marker status as a keyword and direct the front-end to dehighlight it. Users may alter the content of user-specified links without cancelling the marker or deleting its link. The CS approves or rejects the edit request in a message it returns to the DS.

2b. The CS processes the user request: short description (i.e., "What is this?")

The back-end responds to this standard request with a message containing the short description in a format analogous to that of section 3.2. The CS converts this description to a configuration-independent document component set. First the CS infers the appropriate document template for short descriptions from the CSKB. Next it instantiates the template form with the message contents. Then it determines what to represent as link markers as follows.

- To each object in the back-end message the CS applies both back-end bridge laws and hypermedia engine bridge laws for inferring links, one by one until one bridge law succeeds or all fail. If any bridge law succeeds given the current filter settings, then the CS represents that back-end object as a link marker, similar to that in section 3.1. Otherwise the CS passes formatting parameters with the object, but not its identifier. For example, if filtering prevented the CS from inferring Fig. 1's "⟨calculation(variable(tc), model(eoq), scenario(eoq(2))), 60, currency(US))⟩" as a link marker, the CS would have passed it as "⟨value(60), form(currency(US))⟩" in the document component set.
- The CS searches the document's content for keywords registered by users or by the back-end that sent the message. The CS marks each keyword found as a link marker, incorporating the identifier of the node that the keyword represents as part of the marker's identifier.

2c. The CS processes the user request: list of commands (i.e., "What can I do with this?")

The CS often can generate the list of relevant back-end commands directly from the back-end's bridge laws and therefore does not have to communicate with the back-end. For example, the first bridge law in section 3.2 maps the two back-end commands we see in Fig. 1's link ensemble. Resolving complex bridge laws, however, may require internal back-end calculations or information stored in the back-end's own knowledge bases. In this case the CS will have to send a request to the back-end as part of resolving the back-end's bridge laws. (As this may slow processing, we hope that most bridge laws will be specified fully enough to not require back-end processing at run-time.) The CS also processes the selected object using its own general bridge laws. One of the bridge laws in section 3.2 identified a user-specified comment about the selected DSS calculation (i.e., back-end object). The CS represents access to this comment with the third link in Fig. 1. Two other CS bridge laws mapped links corresponding to the hypermedia engine commands "start new user link" and "create new comment."

The CS now formats the link ensemble as a configuration-independent query component set. The CS retrieves the appropriate query template for link ensembles from the CSKB and inserts the five links along with a directive to the DS to include the selected marker's display value as the title. The CS represents each of the five links as a link marker in the query so the user can select any and request assistance (i.e., "What happens if I do this?") for its underlying DSS or hypermedia engine command. Simplistic assistance returns the equivalent of a short description. Sophisticated back-ends provide more sophisticated help.

3. The DS converts the document or query component set for the front end

The DS prepares the document or query for its front-end. It retrieves the protocol the front-end will recognize for documents and queries from the FEKB. The front-end may or may not accept objects embedded in messages and may restrict identifier length. If the front-end does not process dimensional attributes, the DS must pre-format object display representations (e.g., sending "$60.00" instead of ⟨value(60), form(currency(US))⟩). Ideally the front-end will accept a standard document protocol such as ODA or SGML [14], or even a HyTime representation (an SGML-based hypermedia communications standard [52]). Based on the level of front-end support, the DS has to determine whether to represent the query link ensemble (1) as a *dialogue* such as in Fig. 1, into which the user types information, (2) as a document in which users must select a link marker representing one of the commands, or (3) as a menu. The DS may have to sacrifice functionality. For example, Fig. 1's front-end supports query dialogues, but cannot highlight each link as a link marker. Users, therefore, cannot request "What happens if I do this?" assistance for commands directly.

Once converted, the DS passes the document or query component set to the front-end and updates its knowledge bases as shown in Fig. 5. The DS records the user's request and the engine's response in the Session KB to support backtracking, trail construction and undo/redo, etc. The DS records each component set object in the Display KB for interpreting subsequent user requests and for reformatting displays. (The DS includes physical object locations if it must maintain these.) The DS also passes this set of displayed objects to the CS's Active KB to support dynamic updating.

3.6. Our prototype: Max

We have implemented a preliminary text-based prototype of the hypermedia engine inside the Max system, which we developed for the U.S. Coast Guard. Max is a knowledge-based DSS shell [11, 38]. The engine uses a preliminary version of bridge laws to map (1) objects in the DSS knowledge base and reports to hypertext nodes, (2) DSS commands to links, and (3) keywords and objects embedded in DSS messages to link markers. Max's interactive documents and link ensemble queries resemble those in Fig. 1 and Fig. 4.

Max, admittedly, is an "insular, monolithic package," providing its own mandatory front-end. The front-end does not support external objects, so the engine keeps track of its objects' locations within the front-end's windows. The current prototype's front-end and hypertext engine are not entirely independent, neither are the DS and CS entirely separate subsystems. The back-end, however, is completely independent of the engine. It communicates solely through Fig. 2's back-end communications language. Indeed we have developed two separate computation-oriented back-ends for Max, a project management system and TEFA [4,5], a model management system.

4. Hypermedia engine / client cooperation and coordination

The two proposed architectures pose many interesting and unresolved questions concerning cooperation and coordination among a hypermedia engine and independent external information systems. This section discusses several issues our research will address.

The hypermedia engine requires the cooperation of its client front-ends and back-ends. The more sophisticated and coordinated each is, the higher the degree of hypermedia functionality the engine can provide. To provide ubiquitous hypermedia support, however, the engine must accommodate front-ends and back-ends that do not meet the compliance standards we desire. As part of our research we are investigating the minimal level of cooperation among front-ends, back-ends and the hypermedia engine. ([18,34,60,62] also investigate compliance requirements. [32,31] report on an integration architecture using *state-change messages* that claims to require less coordination among the hypermedia engine and its external systems. Marshall also has an enlightening discussion of how compliance affects hypermedia support during document interchange [44]. [8] complements this discussion.)

In [7] we introduced a preliminary set of minimal requirements for client/engine cooperation. Now we augment this set, addressing the interaction between the engine and interface-oriented front-end systems in section 4.1, and between the engine and computation-oriented back-ends in section 4.2. These apply to information systems from either version of our architecture. In section 4.3 we discuss how these requirements impact the integration of existing information systems.

These requirements stem from our own research. We believe, however, that they are universal enough to provide a starting set of general guidelines for all *system-level* approaches to hypermedia integration, including those not employing an external hypermedia engine. (Approaches that integrate hypermedia directly into individual *applications*, e.g. [41], do not require our degree of generality.)

While we primarily address compliance and coordination issues in this paper, we must investigate many other issues ranging from the validity of links and node contents (especially when fusing information from multiple, dynamic back-ends [7]), to handling different versions [26,27,55], to supporting cooperative work such as with group DSS and distributed group support systems, etc. We discuss other "non-compliance" issues in [6,7,11].

4.1. The hypermedia engine and front-end compliance

The hypermedia engine provides the front-end and its users with simultaneous access to multiple back-ends. The engine manages hypermedia constructs (e.g., link markers representing user-defined and back-end objects, comments, trails, and overviews) and hypermedia control (e.g., filtering, context-sensitive forward navigation and backtracking). In return the front-end should provide the following functionality.

- Identifying objects in front-end workspaces

Front-ends either must track the location and identifiers of external objects (i.e., hypermedia link markers) or make their up-to-date positions available. In the latter case the DS will have to interpret positions in every type of media the front-end supports (text, graphics, sound, etc.), as well as monitor every editing action that can alter the location of hypermedia markers.

- Front-ends must gain editing permission from the hypermedia engine

Users may alter the display contents of some types of link markers but not others. Users may alter certain types of markers on the condition that the hypermedia engine cancels their marker status. A sophisticated front-end could manage this on behalf of the hypermedia engine, thus speeding interface operations. For most front-ends, however, the hypermedia engine will have to manage editing permission (as in our Max prototype) and the front-end must request this every time the user inserts or deletes.

Copying and pasting provides an additional editing challenge. Whenever it pastes a link marker, the front-end should register the new instance with the DS and obtain a new unique identifier for it.

- Front-ends must provide hypermedia prompts

We expect front-ends to support three standard hypermedia-style requests: a short description of a marker's object, a list of hypermedia and back-end commands applicable to that object, and command assistance. Front-ends should provide some mechanism for users to invoke each of these (e.g., a keystroke combination or a special mouse button).

- Manipulating documents with embedded hypermedia objects

When the front-end saves a document with embedded objects, it could save the objects as well. Otherwise the DS will have to regenerate the link markers when the front-end reopens the document. In any case the front-end must inform the DS when it opens an existing (or new [61]) hypermedia-oriented document so the DS can provide hypermedia support and dynamic updating.

In most information systems users create documents manually. With a hypermedia engine, front-ends must accept the externally-generated documents with embedded objects that the DS passes. The front-end should handle dynamic changes as well. The DS may add additional objects to open documents (e.g., when users create their own comments and links on the front-end workspace [59]). Dynamic updating (which requires the front-end to accept external interrupts) may change the display value of hypermedia link markers [32]. Sophisticated front-ends will accommodate these demands. If not, the hypermedia engine may not be able to provide full hypermedia functionality.

4.2. The hypermedia engine and back-end compliance

The hypermedia engine provides the back-end and its users with access to a variety of front-ends. It manages hypermedia functionality (linking, annotation, backtracking, filtering, network overviews of applications) on behalf of the back-end. In return the back-end should supply the hypermedia engine with specific information about its structure, and its applications' documents and data elements. However, even if a back-end declares no bridge laws or keywords, and passes messages without objects, the hypermedia engine still will provide standard hypermedia functionality (user annotation, backtracking, etc.) In this case the user will not be able to access *back-end* objects or operations in a hypermedia fashion.

- Builders must write bridge laws

The person who knows the back-end the best – the systems programmer who builds or maintains it – should develop its bridge laws. The information system builder must be both willing to and capable of developing a set of bridge laws that accurately captures the structure of his sys-

tem. Once in place the bridge laws should map a hypermedia network to any of the system's specific applications. (Application builders and users need have no knowledge of bridge laws. To them, hypermedia functionality occurs automatically!)

Currently builders must represent bridge laws in predicate logic. We hope to remove this restriction by accepting other formats, perhaps through a bridge law editor.

Each builder must develop his own set of bridge laws. We hope to develop bridge law libraries that map classes of information systems-complete "standard" bridge law sets that handle the models, attributes, data and operations found, e.g., in linear program (LP) packages, relational databases, spreadsheet packages, or rule-based expert system shells. The builder of, say, a new LP package would only have to match the elements in his system to those in the standard LP set. The standard set would provide most of the bridge laws for his system. This would reduce the builder's effort both in determining which kinds of bridge laws would represent his system adequately and in developing these laws.

- Back-ends should embed objects in their messages

The CS cannot infer magically which portions of back-end messages to highlight as link markers. The back-end must mark objects within the messages or provide some content analysis routines for interpreting their messages. The only content analysis the CS automatically performs is keyword search. (An advanced CS could employ, for example, sophisticated content analysis techniques such as lexical affinity [36] to infer undeclared keywords.)

As we demonstrated in section 3.1, back-end messages should include dimensional information for objects or any other content, for which the engine or user might want to alter the display format. For example, a user may wish to change a number's precision.

- Back-ends should support the standard hypermedia engine commands

Just as the front-end should allow users to request short descriptions, command lists and context-sensitive help, back-ends should generate this information on demand.

- Multi-level undo and redo

For the hypermedia engine to support full multiple level undo and redo functionality, the back-end must provide some mechanism for undoing and redoing its operations (e.g., performing a what-if analysis). Otherwise the hypermedia engine can only undo back to the last back-end operation. Back-ends, for example, could return a command with each operation result that would have the effect of restoring the previous back-end state.

Additional guidelines

In [7] we also discussed the following requirements.

- When the back-end message contains a previously-generated report, the hypermedia engine sometimes has trouble locating the positions of the user annotations that were in the previous version. Including the internal structure of each message's content provides additional orientation for the engine.
- To assist in validating user responses to back-end queries, the back-end could provide control information for validity checking.

4.3. The hypermedia engine and existing systems

Builders developing an information system from scratch will find interfacing with the hypermedia engine easier than builders who must retrofit the coordination that the hypermedia engine demands. Builders of existing information systems (assuming they can be located [31]) must reengineer the communications path between the system's interface components and computational components, allowing the hypermedia engine to intercept messages and embed objects. The more loosely coupled and modular an information system is, the simpler hypermedia integration will be.

5. Conclusion

We have yet to see hypermedia availability as a common interface feature. Information system builders wishing to incorporate full hypermedia functionality today must do it themselves. Few new system builders would be willing or able to do this. Fewer builders would put forth the effort to convert existing systems. "A more modest (and practical) goal is to create rules and tools that could be used to allow slightly modified existing

applications to produce data accessible in hypermedia style." [67, p. 81] Certain operating systems, for example, provide system-level hypermedia toolkits for adding hypermedia constructs – nodes, links, markers, etc. – to application data (e.g., the Andrew Toolkit [63], and a recently proposed "core system" [47]). Apple Computer's new operating system, System 7, provides *publish* and *subscribe* capabilities, but these, in themselves, fall far short of full hypermedia functionality.

Several research efforts have succeeded in creating links among primarily non-hypermedia information systems and in facilitating their traversal. The commercially-available Sun Link Service [59], PROXHY [34], Microcosm [18] and Multicard [62] each has a hypermedia engine that executes concurrently with external systems and provides linking support. Each of these four hypermedia systems (ideally) expects client information systems to support link creation and selection by embedding hypermedia calls and handling minimal hypermedia functionality in a manner similar to the toolkit approach. The latter three systems also provide multiple levels of hypermedia support, based on the degree of hypermedia compliance the client non-hypermedia information system provides. Each also provides a limited set of support for client systems that are not hypermedia compatible at all. All four systems, however, concentrate primarily on supporting *front-end* interface-oriented systems, as opposed to *back-end* computation-oriented systems. ([18] and [62] seem to imply that support for back-end systems could be implemented in Microcosm and Multicard, but this neither is stated explicitly nor is their primary focus.)

Our research, in contrast, emphasizes the mapping of hypermedia representations to the back-end computational aspects of information systems. We find few methods that externally superimpose hypermedia constructs over an application without adding to its data or knowledge base (e.g., the proposed Relationship Management System [32]). When completely developed, our hypermedia engine will provide full hypermedia functionality to dynamically changing applications while running concurrently with them and mapping a hypermedia representation that does not alter them.

Through our preliminary architecture we have identified many challenges for hypermedia support of dynamic information systems. We have started developing techniques to address these, which we soon shall implement in an improved prototype.

Hypermedia should be a widely implemented paradigm for information management and presentation. We invite information system developers, and challenge both information system and hypermedia researchers, to join us and make this goal a reality.

References

[1] P. Balasubramanian, T. Isakowitz, H. Johar and E. Stohr, Hyper Model Management Systems, in: Proceedings of the Twenty-fifth Hawaii International Conference on System Sciences, Volume III (Kauai, Jan. 1992) 462–472.

[2] D. Barman, RelType: Relaxed Typing for Object-Oriented Hypermedia Representations, in: Object-Oriented Programming in AI: Workshop Notes from the Ninth Annual National Conference on Artificial Intelligence (Anaheim, 1991).

[3] E. Berk and J. Devlin, Eds., Hypertext/Hypermedia Handbook (Intertext Publications/McGraw-Hill Publishing Co., Inc., New York, 1991).

[4] H.K. Bhargava, A Logic Model for Model Management, Ph.D. dissertation (University of Pennsylvania, Philadelphia, PA 19104, 1990).

[5] H. Bhargava, M. Bieber and S.O. Kimbrough, 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, 1988) 179–192.

[6] M. Bieber, Generalized Hypertext in a Knowledge-based DSS Shell Environment, Ph.D. dissertation (University of Pennsylvania, Philadelphia, PA 19104, 1990).

[7] M. Bieber, Issues in Modelling a "Dynamic" Hypertext Interface for Non-Hypertext Information Systems, in: Hypertext '91 Proceedings (San Antonio, Dec. 1991) 203–218.

[8] M. Bieber, On Merging Hypertext into Dynamic, Non-Hypertext Systems, Boston College Computer Science Department Technical Report BCCS-91-14 (Chestnut Hill, MA 02167, Nov. 1991).

[9] M. Bieber, Template-Driven Hypertext: A Methodology for Integrating a Hypertext Interface into Information Systems, Boston College Computer Science Department Technical Report (Chestnut Hill, MA 02167, 1991).

[10] M. Bieber, Automating Hypermedia for Decision Support, Hypermedia 4, No. 2 (1992) 83–110.

[11] M. Bieber and S.O. Kimbrough, On Generalizing the Concept of Hypertext, Management Information Systems Quarterly 16, No. 1 (1992) 77–93.

[12] M. Bieber and S.O. Kimbrough, On the Logic of Generalized Hypertext, Decision Support Systems 11, No. 2 (1994) 241–257.

[13] P. Brown, Assessing the Quality of Hypertext Documents, in: A. Rizk, N. Streitz and J. André, Eds., Hypertext: Concepts, Systems and Applications, Proceedings of European Conference on Hypertext '90 (Cambridge University Press, Versailles, Nov. 1990) 1–12.

[14] F. Cole and H. Brown, Standards: What Can Hypertext Learn From Paper Documents?, in: Proceedings of the Hypertext Standardization Workshop, SP500-178 (NIST, Gaithersburg, Jan. 1990) 59–70.

[15] E.J. Conklin, Hypertext: a Survey and Introduction, IEEE Computer 20, No. 9 (1987) 17–41.

[16] E.J. Conklin, and M.L. Begeman, gIBIS: A Tool for All Reasons, Journal of the American Society for Information Science 40, No. 3 (1989) 200–213.

[17] L. De Young, Linking Considered Harmful, in: Proceedings of European Conference on Hypertext (ECHT) '90 (Cambridge University Press, Versailles, Nov. 1990) 238–249.

[18] H. Davis, W. Hall, I. Heath, G. Hill and R. Wilkins, Towards an Integrated Information Environment with Open Hypermedia Systems, in: Proceedings of the ACM Conference on Hypertext (Milan, Nov. 1992) 181–190.

[19] P. De Bra, G. Houben and Y. Kornatzky, An Extensible Data Model for Hyperdocuments, in: Proceedings of the ACM Conference on Hypertext (Milan, Nov. 1992) 222–231.

[20] E.A. Fox, Q.F. Chen and R.K. France, Integrating Search and Retrieval with Hypertext, in: E. Berk and J. Devlin, Eds., Hypertext/Hypermedia Handbook (Intertext Publications/McGraw-Hill Publishing Co., Inc., New York, 1991) 329–355.

[21] M.E. Frisse, S.B. Cousins and S. Hassan, WALT: A Research Environment for Medical Hypertext, in: Hypertext '91 Proceedings (San Antonio, Dec. 1991) 389–394.

[22] R. Furuta and P.D. Stotts, The Trellis Hypertext Reference Model, in: Proceedings of the Hypertext Standardization Workshop, SP500-178 (NIST, Gaithersburg, Jan. 1990) 83–94.

[23] L. Gallagher, R. Futura and P.D. Stotts, Increasing the Power of Hypertext Search with Relational Queries, Hypermedia 2, No. 1 (1990) 1–14.

[24] F. Garzotto, L. Mainetti and P. Paolini, Navigation Patterns in Hypermedia Data Bases, in: Proceedings of Twenty-Sixth Annual Hawaii International Conference on System Science (Wailea, Jan. 1993).

[25] F. Garzotto, P. Paolini and D. Schwabe, HDM – A Model-Based Approach to Hypertext Application Design, ACM Transactions on Information Systems 11, No. 1 (1993) 1–26.

[26] A. Haake, CoVer: A Contextual Version Server for Hypertext Applications, in: Proceedings of the ACM Conference on Hypertext (Milan, Nov. 1992) 43–52.

[27] F.G. Halasz, Reflections on NoteCards: Seven Issues for the Next Generation of Hypermedia Systems, Communications of the ACM 31, No. 7 (1988) 836–855.

[28] F. Halasz and M. Schwartz, The Dexter Hypertext Reference Model, in: Proceedings of the Hypertext Standardization Workshop, SP500-178 (NIST, Gaithersburg, Jan. 1990) 95–134.

[29] J. Haugeland, The Nature and Plausibility of Cognitivism, in: John Haugeland, Ed., Mind Design: Philosophy, Psychology, Artificial Intelligence (MIT Press, Cambridge, 1981).

[30] H. Hua and S.O. Kimbrough, On Hypermedia-Based Argumentation Decision Support Systems, in: Proceedings of Twenty-Sixth Annual Hawaii International Conference on System Science (Wailea, Jan. 1993).

[31] T. Isakowitz, Hypermedia, Information Systems and Organizations: A Research Agenda, in: Proceedings of the Twenty-sixth Hawaii International Conference on System Sciences (Wailea, Jan. 1993).

[32] T. Isakowitz and E.A. Stohr, Hypertext-based Relationship Management for DSS, New York University Information Systems Department Working Paper IS-92-22 (New York, NY 10012, Jul. 1992).

[33] D.S. Jordan, D.M. Russell, A.S. Jensen and R.A. Rogers, Facilitating the Development of Representations in Hypertext with IDE, in: Hypertext '89 Proceedings (Pittsburgh, Nov. 1989) 93–104.

[34] C. Kacmar and J. Leggett, PROXHY: A Process-Oriented Extensible Hypertext Architecture, ACM Transactions on Information Systems 9, No. 4 (1991) 399–419.

[35] H. Kaindl and M. Snaprud, Hypertext and Structured Object Representation: A Unifying View, in: Hypertext '91 Proceedings (San Antonio, Dec. 1991) 313–328.

[36] S.M. Kaplan and Y.S. Maarek, Incremental Maintenance of Semantic Links in Dynamically Changing Hypertext Systems, Interacting with Computers 2, No. 3 (1990) 337–366.

[37] S.O. Kimbrough, On the Reduction of Genetics to Molecular Biology, in: Philosophy of Science 46 No. 3 (1979) 389–406.

[38] S.O. Kimbrough, C. Pritchett, M. Bieber and H. Bhargava, The Coast Guard's KSS Project, Interfaces 20, No. 6 (1990) 5–16.

[39] L. Koved and B. Shneiderman, Embedded Menus: Selecting Items in Context, Communications of the ACM 29, No. 4 (1986) 312–318.

[40] G.P. Landow, Popular Fallacies About Hypertext, in: D.H. Jonassen and H. Mandl, Eds., Designing Hypermedia for Learning (Springer-Verlag, 1990) 39–59.

[41] D. Lange, Object-Oriented Hypermodelling of Hypertext Supported Information Systems, in: Proceedings of Twenty-Sixth Annual Hawaii International Conference on System Science (HICSS) (Wailea, Jan. 1993).

[42] A. Littleford, Artificial Intelligence and Hypermedia, in: E. Berk and J. Devlin, Eds., Hypertext/Hypermedia Handbook (Intertext Publications/McGraw-Hill Publishing Co., Inc., New York, 1991) 357–378.

[43] K.C. Malcolm, S.E. Poltrock and D. Schuler, Industrial Strength Hypermedia: Requirements for a Large Engineering Enterprise, in: Hypertext '91 Proceedings (San Antonio, Dec. 1991) 13–24.

[44] C. Marshall, Standards: A Multi-Tiered Approach to Hypertext Integration: Negotiating Standards for a Heterogeneous Application Environment, in: Proceedings of the Hypertext Standardization Workshop, SP500-178 (NIST, Gaithersburg, Jan. 1990) 167–178.

[45] C.C. Marshall, F.G. Halasz, R.A. Rogers and W.C. Janssen Jr., Aquanet: A Hypertext Tool to Hold Your Knowledge in Place, in: Hypertext '91 Proceedings (San Antonio, Dec. 1991) 261–275.

[46] C.C. Marshall and P.M. Irish, Guided Tours and On-Line Presentations: How Authors Make Existing Hypertext Intelligible for Readers, in: Hypertext '89 Proceedings (Pittsburgh, Nov. 1989) 15–42.

[47] H. Maurer and I. Tomek, Broadening the Scope of Hypermedia Principles, Hypermedia 2, No. 3 (1990) 201–220.

[48] N. Meyrowitz, The Missing Link: Why We're All Doing Hypertext Wrong, in: E. Barrett, Ed., The Society of Text: Hypertext, Hypermedia, and the Social Construction of Information (MIT Press, Cambridge, 1989) 107–114.

[49] R. Minch, Application and Research Areas for Hypertext in Decision Support Systems, Journal of Management Information Systems 6, No. 3 (1990) 119–138.

[50] E. Nagel, The Structure of Science: Problems in the Logic of Scientific Explanation (Harcourt, Brace and World, Inc., New York, 1961).

[51] J. Nanard and M. Nanard, Using Structured Types to Incorporate Knowledge into Hypertext, in: Hypertext '91 Proceedings (San Antonio, Dec. 1991) 329–343.

[52] S. Newcomb, N. Kipp and V. Newcomb, The "HyTime" Hypermedia/Time-based Document Structuring Language, Communications of the ACM 34, No. 11 (1991) 67–83.

[53] J. Nielsen, Hypertext Bibliography, Hypermedia 1, No. 1 (1989) 74–91.

[54] J. Nielsen, Hypertext and Hypermedia (Academic Press, 1990).

[55] K. Østerbye, Structural and Cognitive Problems in Providing Version Control for Hypertext, in: Proceedings of the ACM Conference on Hypertext (Milan, Nov. 1992) 33–42.

[56] H.V.D. Parunak, Hypermedia Topologies and User Navigation, in: Hypertext '89 Proceedings (Pittsburgh, Nov. 1989) 43–50.

[57] H.V.D. Parunak, Don't Link Me In: Set Based Hypermedia for Taxonomic Reasoning, in: Hypertext '91 Proceedings (San Antonio, Dec. 1991) 233–242.

[58] H.V.D. Parunak, Toward Industrial Strength Hypermedia, in: E. Berk and J. Devlin, Eds., Hypertext/Hypermedia Handbook (Intertext Publications/McGraw-Hill Publishing Co., Inc., New York, 1991) 381–395.

[59] A. Pearl, Sun's Link Service: A Protocol for Open Linking, in: Hypertext '89 Proceedings (Pittsburgh, Nov. 1989) 137–146.

[60] J.J. Puttress and N.M. Guimaraes, The Toolkit Approach to Hypermedia, in: A. Rizk, N. Streitz and J. André, Eds., Hypertext: Concepts, Systems and Applications, Proceedings of European Conference on Hypertext '90 (Cambridge University Press, Versailles, Nov. 1990) 25–37.

[61] R.N. Robson, Using Hypertext to Locate Reusable Objects, in: Proceedings of Twenty-fifth Annual Hawaii International Conference on System Science (HICSS) (Kauai, Jan. 1992) 549–557.

[62] A. Rizk and L. Sauter, Multicard: An Open Hypermedia System, in: Proceedings of the ACM Conference on Hypertext (Milan, Nov. 1992) 4–10.

[63] M. Sherman, W. Hansen, M. McInerny and T. Neuendorfer, Building Hypertext on a Multimedia Toolkit: An Overview of the Andrew Toolkit Hypermedia Facilities, in: A. Rizk, N. Streitz and J. André, Eds., Hypertext: Concepts, Systems and Applications, Proceedings of European Conference on Hypertext '90 (Cambridge University Press, Versailles, Nov. 1990) 13–24.

[64] H. Simon, The New Science of Management Decision (Harper and Row, New York 1977).

[65] P.D. Stotts and R. Furuta, Petri-net-based Hypertext: Document Structure with Browsing Semantics, ACM Transactions on Information Systems 7, No. 1 (1989) 3–29.

[66] P.D. Stotts and R. Furuta, Hierarchy, Composition, Scripting Languages, and Translators for Structured Hypertext, in: A. Rizk, N. Streitz and J. André, Eds., Hypertext: Concepts, Systems and Applications, Proceedings of European Conference on Hypertext '90, (Cambridge University Press, Versailles, Nov. 1990) 180–193.

[67] I. Tomek, S. Khan, T. Müldner, M. Nassar, G. Novak and P. Proszynski, Hypermedia–Introduction and Survey, Journal of Microcomputer Applications 14, No. 2 (1991) 63–103.

[68] R.H. Trigg and M. Weiser, Textnet: A Network-Based Approach to Text Handling, ACM Transactions on Office Information Systems 4, No. 1 (1986) 1–23.

[69] K. Utting, and N. Yankelovich, Context and Orientation in Hypermedia Networks, ACM Transactions on Information Systems 7, No. 1 (1989) 58–84.

[70] A. Van Dam, Hypertext '87: Keynote Address, Communications of the ACM 31, No. 7 (1988) 887–895.

[71] J.A. Waterworth and M.H. Chignell, A Model for Information Exploration, Hypermedia 3, No. 1 (1991) 35–58.

[72] E. Wilson, Links and Structures in Hypertext Databases for Law, in: A. Rizk, N. Streitz and J. André, Eds., Hypertext: Concepts, Systems and Applications, Proceedings of European Conference on Hypertext '90 (Cambridge University Press, Versailles, Nov. 1990) 194–211.