PROVIDING INFORMATION SYSTEMS WITH FULL HYPERMEDIA FUNCTIONALITY

by

Michael Bieber
New Jersey Institute of Technology
Department of Computer and Information Science

Visiting Assistant Professor
Information Systems Department
Leonard N. Stern School of Business

October 1992

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

Working Paper Series

STERN IS-92-29

Providing Information Systems with Full Hypermedia Functionality*

Michael Bieber**
New Jersey Institute of Technology
Department of Computer and Information Science

Abstract

The goal of this research is to provide hypermedia *functionality to all information systems. In this paper I present 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. I then describe a set of minimal requirements for integrating the hypermedia engine. The more sophisticated and cooperative the information system, the higher the level of hypermedia support the engine will provide.*

1: Hypermedia and information systems

I 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, I believe, will be incorporated in the interfaces of all information systems that interact with people. My 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.

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

In §2 I briefly review the concepts of hypermedia and our enhancement, *generalized hypermedia*. Generalized hypermedia is at the heart of my hypermedia engine's architecture. In §3 I introduce two versions of the engine's architecture and describe its internal structure. In §4 I discuss the minimal requirements for hypermedia integration—the commitment information system builders have to make to use my architecture. I conclude in §5 by briefly comparing my work with other current approaches.

2: Hypermedia and generalized hypermedia

Hypertext [3, 14, 45, 49, 50, 59] 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. Figure 1 shows a hypermedia-oriented *interactive document* similar

*Forthcoming in *Proceedings of Hawaii International Conference on Systems Science HICSS-26* (Maui, 1993). An expanded version of this paper is available from the author.

**Author's e-mail address: *bieber@cis.njit.edu*. The author currently is Visiting Assistant Professor of Information Systems at the Stern School, New York University, 1992-93.

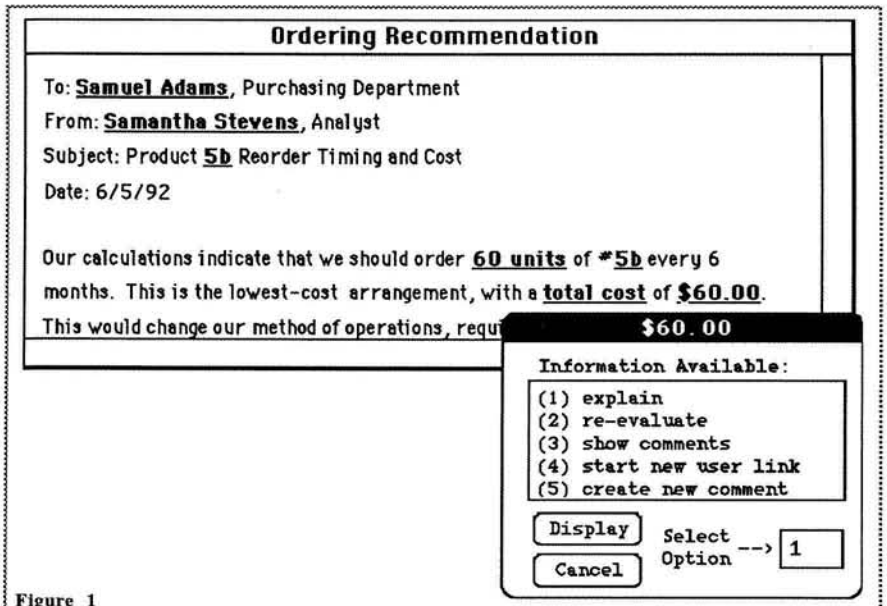

Figure 1
Accessing Application and Hypermedia Functionality in a Hypermedia-Style Interface

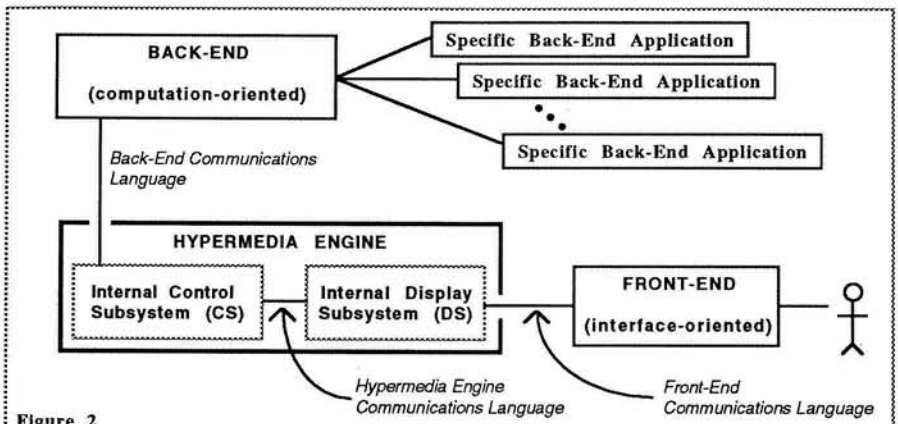

Figure 2
Hypermedia Engine Architecture (Version 1):
Binding Independent Back-End and Front-End Information Systems

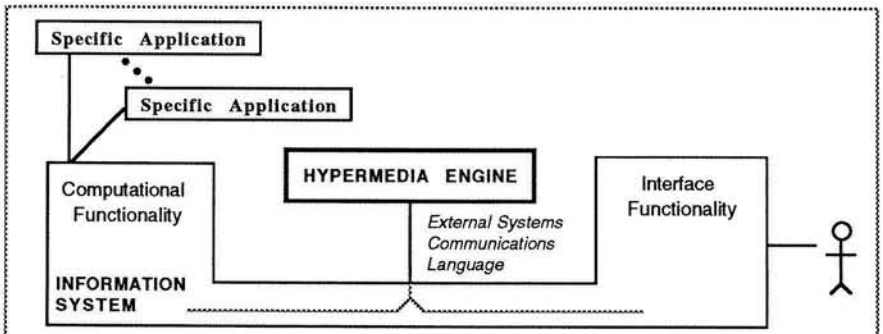

Figure 3
Hypermedia Engine Architecture (Version 2):
Serving an Information System with Adequate Computation and Interface Functionality

to those our Max prototype produces [10, 34]. This document node represents a report generated by a decision support system (DSS) and passed on to the *hypermedia engine* for display. The underlined and boldfaced text strings are link markers, each associated with one or more links. In Figure 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.

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 I 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 [38].)

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 [35], 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 [18, 36, 45, 50, 61] of the hypermedia network. Overviews (often [16]) help alleviate the *network disorientation* [14, 50] associated with hypermedia's nonrestrictive, user-directed access.

Information retrieval-style queries provide an alternative method of forward navigation [17, 20, 63]. 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 [21].

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.

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* [60] or *guided tours* [21, 42]. 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 [9].

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.

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" [44]. Users who want hypermedia functionality often must abandon the software they currently use—an impractical restriction [30, 39]. The first limitation motivated us to develop *generalized hypertext* or *generalized hypermedia* [6, 10, 11]. The second motivated my 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 [23]: 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 §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.

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 [9]. This does not mean that information system builders simply can plug in the 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 or her 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 this 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 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

Figure 2 shows a version of the proposed hypermedia engine's architecture that binds independent back-end and front-end information systems. By *back-end* systems, I mean information systems that primarily provide computation functionality, such as decision support systems, expert systems, intelligent tutoring systems, database management systems, project management systems, etc. By *front-end* systems I 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 have complied with the requirements I discuss in §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 [52]. 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 Figure 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.

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

3.1: An example

I describe the hypermedia engine's architecture through Figure 1's simple text-based example. (My model also supports non-text content and link markers.) Figure 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 hypermedia 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 Figure 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 a document component set for display.

In the following sections I examine different aspects of the hypermedia engine and integrating it into information systems.

3.2: Bridge laws and filters: techniques for automating hypermedia

In this section I 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 documents and commands 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 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" [25, 33, 46] 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 "$\forall$") 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 knowledge base that satisfies the bridge law's conditions.

In Figure 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 I 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.

Together, generalized hypermedia and its bridge laws provide a logic-based *knowledge representation* that enable 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 my research, I shall determine whether a complete set of bridge laws suffices for the engine to perform both structure search and content search [22, 23], and generate a network overview. (Producing an overview for a static hypermedia network is not a trivial task (see, e.g., [61]). No one, as yet, has tackled overviews for virtual environments involving computation, such as my own.)

Several other knowledge representation approaches have appeared in the literature, e.g., Petri nets [57, 58], structured object representation [31], schemata [22, 28, 41], object-oriented hypermodeling [37] and high-level specification languages [55]. Other systems that make use of a knowledge representation include gIBIS [15], Hypermedia-based Argumentation DSS [26], Electronic Working Papers [16], MacWeb [47], IDE [29] and RelType [2]. In future papers I hope to compare implementations using bridge laws and a generalized hypermedia engine with systems using other knowledge representations.

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, 9, 10].

3.3: Internal Control Subsystem (CS)

The hypermedia engine has two major components: the Internal Control Subsystem (CS) and the Internal Display Subsystem (DS).

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, perhaps using a HyTime representation (an SGML-based hypermedia communications standard [48]).

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 [51]), 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 its 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 [4, 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.

Balasubramanian et al. present an alternative system architecture that insulates bridge laws as much as possible from changes to the engine or back-end. Their architecture includes a separate bridge law manager between the hypermedia engine and back-end [1].

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 [19])
- 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. 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 [24].

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 [62] 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., [12]).

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

4: Hypermedia engine/client cooperation and coordination

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 standards I would prefer [30]. As part of my research I am investigating the minimal level of cooperation among front-ends, back-ends and the hypermedia engine. ([30, 40, 54] also investigate a set of various requirements. [28, 27] report on an integration architecture using *state-change messages* that claims to require less coordination among the hypermedia engine and its external systems.)

In [7] I introduced a preliminary set of minimal requirements for client/engine cooperation. Now I augment this set, addressing the interaction between the engine and interface-oriented front-end systems in §4.1, and between the engine and computation-oriented back-ends in §4.2. These apply to information systems from either version of my architecture.

These requirements stem from our own research. I believe, however, that they 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., [37], do not require my degree of generality.)

4.1: The hypermedia engine and front-ends

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 must track the location and identifiers of external objects (i.e., hypermedia link markers), and return the corresponding identifier when a user selects a link marker.

- 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. For example, users may delete, but not modify, back-end object markers. Users may alter a keyword, but the CS will deregister its marker as a keyword and direct the front-end to dehighlight it. 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

  I 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 [55]) 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 that the DS

passes with embedded objects. 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 [53]). Dynamic updating (which requires the front-end to accept external interrupts) may change the display value of hypermedia link markers [28]. 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-ends

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. Even if a back-end declares no bridge laws or keywords, however, 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. This information system builder must be both willing to and capable of developing a set of bridge laws that accurately captures the structure of his system. 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. I hope to remove this restriction by accepting other formats, perhaps through a bridge law editor.

- 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 [32] to infer undeclared keywords.)

  As I demonstrated in §3.1, back-end messages should include dimensional information for objects or 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.

Additional Guidelines: In [7] I 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. (The back-end could incorporate a standard document protocol such as ODA or SGML [13].)
- To assist in validating user responses to back-end queries, the back-end could provide control information for validity checking.

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." [59 pg. 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 [56], and Maurer and Tomek's proposed "core system" [43]). Apple Computer's new operating system, System 7, provides *publish* and *subscribe* capabilities, but these, in themselves, fall far short of full hypermedia functionality. There are hypermedia services that run concurrently with distributed applications in networked environments (e.g., the commercially-available Sun Link Service [53] and PROXHY [30]). We find few methods, however, that externally superimpose hypermedia constructs over an application without adding to its data or knowledge base (e.g., Puttress and Guimaraes' Hypertext Object-oriented Toolkit [54]). When completely developed, my 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 my preliminary architecture I have identified many challenges for hypermedia support of dynamic information systems. I have started developing techniques to address these, which I will implement in an improved prototype soon.

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

Acknowledgment

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. Tomás Isakowitz of New York University and Bob Minch of Boise State University both made invaluable suggestions, as did anonymous reviewers. 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.

References

[1] P. Balasubramanian, T. Isakowitz, H. Johar and E. Stohr, Hyper Model Management Systems, in: Proceedings of HICSS-25 (Kauai, 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 Modeling 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 Technical Report BCCS-91-14 (Nov. 1991).

[9] M. Bieber, Automating Hypermedia for Decision Support, Hypermedia (forthcoming).

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

[11] M. Bieber and S.O. Kimbrough, On the Logic of Generalized Hypertext, Decision Support Systems (forthcoming).

[12] 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 (ECHT) '90 (Cambridge University Press, Versailles, Nov. 1990) 1-12.

[13] 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.

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

[15] 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.

[16] L. De Young, Linking Considered Harmful, in: Proceedings of ECHT'90 238-249.

[17] E.A. Fox, Q.F. Chen and R.K. France, Integrating Search and Retrieval with Hypertext, in [3] 329-355.

[18] 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.

[19] 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.

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

[21] F. Garzotto, L. Mainetti and P. Paolini, Navigation Patterns in Hypermedia Data Bases, in this proceedings.

[22] F. Garzotto, P. Paolini and D. Schwabe, HDM - A Model for the Design of Hypertext Applications, in: Hypertext '91 Proceedings (San Antonio, Dec. 1991) 313-328.

[23] 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.

[24] 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.

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

[26] H. Hua and S.O. Kimbrough, On Hypermedia-Based Argumentation Decision Support Systems, in this proceedings.

[27] T. Isakowitz, Hypermedia in Information Systems and Organizations: A Research Agenda, in this proceedings.

[28] T. Isakowitz and E.A. Stohr, Hypertext-based Relationship Management for DSS, NYU Stern Working Paper IS-92-22 (Jul. 1992).

[29] 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. 1991) 93-104.

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

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

[32] 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.
[33] S.O. Kimbrough, On the Reduction of Genetics to Molecular Biology, in: Philosophy of Science 46 No. 3 (1979) 389-406.
[34] S.O. Kimbrough, C. Pritchett, M. Bieber and H. Bhargava, The Coast Guard's KSS Project, Interfaces 20, No. 6 (1990) 5-16.
[35] L. Koved and B. Shneiderman, Embedded Menus: Selecting Items in Context, Communications of the ACM 29, No. 4 (1986) 312-318.
[36] G.P. Landow, Popular Fallacies About Hypertext, in: D.H. Jonassen and H. Mandl, Eds., Designing Hypermedia for Learning (Springer-Verlag, 1990) 39-59.
[37] D. Lange, Object-Oriented Hypermodeling of Hypertext Supported Information Systems, in this proceedings.
[38] A. Littleford, Artificial Intelligence and Hypermedia, in: [3] 357-378.
[39] 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.
[40] C. Marshall, Standards: A Multi-Tiered Approach to Hypertext Integration: Negotiating Standards for a Heterogenous Application Environment, in: Proceedings of the Hypertext Standardization Workshop, SP500-178 (NIST, Gaithersburg, Jan. 1990) 167-178.
[41] 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.
[42] 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. 1991) 15-42.
[43] H. Maurer and I. Tomek, Broadening the Scope of Hypermedia Principles, Hypermedia 2, No. 3 (1990) 201-220.
[44] 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.
[45] R. Minch, Application and Research Areas for Hypertext in Decision Support Systems, Journal of Management Information Systems 6, No. 3 (1990) 119-138.
[46] E. Nagel, The Structure of Science: Problems in the Logic of Scientific Explanation (Harcourt, Brace & World, Inc., New York, 1961).
[47] J. Nanard and M. Nanard, Using Structured Types to Incorporate Knowledge into Hypertext, in: Hypertext '91 Proceedings (San Antonio, Dec. 1991) 329-343.
[48] 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.
[49] J. Nielsen, Hypertext Bibliography, Hypermedia 1, No. 1 (1989) 74-91.
[50] J. Nielsen, Hypertext and Hypermedia (Academic Press, 1990).
[51] H.V.D. Parunak, Hypermedia Topologies and User Navigation, in: Hypertext '89 Proceedings (Pittsburgh, Nov. 1991) 43-50.
[52] H.V.D. Parunak, Toward Industrial Strength Hypermedia, in [3] 381-395.
[53] A. Pearl, Sun' s Link Service: A Protocol for Open Linking, in: Hypertext '89 Proceedings (Pittsburgh, Nov. 1991) 137-146.
[54] J.J. Puttress and N.M. Guimaraes, The Toolkit Approach to Hypermedia, in: Proceedings of ECHT'90 25-37.
[55] R.N. Robson, Using Hypertext to Locate Reusable Objects, in: Proceedings of HICSS-25 (Kauai, 1992) 549-557.
[56] M. Sherman, W. Hansen, M. McInerny and T. Neuendorfer, Building Hypertext on a Multimedia Toolkit: An Overview of the Andrew Toolkit Hypermedia Facilities, in:: Proceedings of ECHT'90 13-24.
[57] 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.
[58] P.D. Stotts and R. Furuta, Hierarchy, Composition, Scripting Languages, and Translators for Structured Hypertext, in: Proceedings of ECHT'90 180-193.
[59] 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.
[60] 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.
[61] K. Utting, and N. Yankelovich, Context and Orientation in Hypermedia Networks, ACM Transactions on Information Systems 7, No. 1 (1989) 58-84.
[62] A. Van Dam, Hypertext '87: Keynote Address, Communications of the ACM 31, No. 7 (1988) 887-895.
[63] E. Wilson, Links and Structures in Hypertext Databases for Law, in: Proceedings of ECHT'90 194-211.