

**HYPertext-BASED RELATIONSHIP
MANAGEMENT FOR DSS**

by

Tomás Isakowitz

Information Systems Department
Leonard N. Stern School of Business
New York University
New York, NY 10012

and

Edward A. Stohr

Information Systems Department
Leonard N. Stern School of Business
New York University
New York, NY 10012

July 1992

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

Working Paper Series

STERN IS-92-22

Hypertext-Based Relationship Management for DSS

Tomas Isakowitz and Edward A. Stohr
Information Systems Department
Stern School of Business
New York University

July 13, 1992

Abstract

There is a need for integrated access to a wide range of information related to the development and use of DSS in organizations. This information comes in many forms, both formal and informal, and is highly interrelated. To handle this complex information base, we argue that a separate relationship management component should be added to the three traditional components of a DSS (namely, the database, user interface and model management systems). The role of the relationship management component is to relieve DSS application programs of the need to maintain and provide access to the complex set of relationships that can exist between elements in the application domain. We discuss the kinds of information and relationships that arise during the development and use of a DSS, outline the requirements for an independent subsystem to manage this information base, and propose the use of an extended hypertext software system, *H+*, to simultaneously handle relationship management and provide an interesting and useful interface to users.

Contents

1	Introduction	1
2	ROLE OF HYPERTEXT IN DSS	2
2.1	A User-Interface for All classes of User	3
2.2	Hypertext as a Cognitive Aid	3
2.3	Hypertext in Support of Application Programming	4
3	RELATIONSHIPS IN A DSS ENVIRONMENT	5
4	REQUIREMENTS FOR RELATIONSHIP MANAGEMENT	8
4.1	Requirements	9
4.1.1	High Level of Abstraction	9
4.1.2	Rapid authoring	9
4.1.3	Automated establishment and maintenance of relationships	10
4.1.4	Support exploration within the application domain	11
4.1.5	Coordinate computation within the application domain	11
4.1.6	Provide semantics for abstract relationships	12
4.1.7	Support for distributed group work	13
4.1.8	Automated Version Control	14
4.2	Summary of hypertext features for relationship management	14
5	AN ARCHITECTURE FOR RELATIONSHIP MANAGEMENT	15
6	RELATIONSHIP MANAGEMENT THROUGH SCHEMAS	18
6.1	A Language for Schemas	22
6.2	Representing more intricate relationships	25
6.3	A Communications Protocol	26
6.4	An example	27
7	DISCUSSION	30

List of Figures

1	An architecture to support schemas.	16
2	Versioning and sensitivity analysis schema	19
3	The energy production-distribution model.	21
4	From schema to hypertext	23
5	A portion of a report describing a solution to the energy problem.	25
6	The evolution of a schema and the user's view.	28

1 Introduction

The idea of hypertext was first put forward by Vannevar Bush in 1945 [Bus45]. He proposed that the nation's top scientists, recently released from war-related research, turn their energies to the peace-time accumulation of knowledge. He conceived this knowledge as a complex web of concepts and relationships between those concepts. He suggested that scientific knowledge be codified in this form and shared amongst researchers as a way of developing a store of knowledge as a national and international resource. Englebart [Eng83, Eng88] (inventor of the mouse and windows) and Nelson [Nel80] (who first coined the term hypertext) were early proponents of the idea that computers can fundamentally change the way in which information is arranged, displayed and assimilated. The idea that knowledge consists of chunks of related information lies at the heart of hypertext and hypermedia research and development today.

A hypertext document contains chunks (hypertext "nodes") of information that are related by hypertext "links". In contrast to normal text which relies on a primarily linear ordering of ideas (the exceptions being footnotes and references), hypertext nodes can be visited in any order depending upon the whim of the reader and the "links" that were embedded in the text by the author. Hypermedia extends the idea of hypertext to include chunks of information in various media such as graphics, sound and video. Recent advances in technology, and the development of commercial systems such as OWL, MAC system 7, and others [Nie90a] have made hypertext and hypermedia an attractive proposition in many applications. Conklin [Con87] and Nielsen [Nie90a] are excellent surveys of hypertext research and practice that discuss potential applications in many diverse areas.

Despite rapid developments in other knowledge-oriented areas, hypertext has received surprisingly little attention by DSS researchers. Minch [Min90] provides an excellent review of potential applications of hypertext in the DSS arena using a framework with four dimensions: 1) individual (user) factors, 2) problem characteristics, 3) situational and organizational factors and 4) technological factors. Kimbrough, Bieber et al have developed MAX [KPBB90], a hypertext-based system for model management. Max provides a proof of concept of the feasibility of hypertext for DSS. Holsapple, Whinston et al [CHW92, BWD⁺89] have extended the hypertext idea to that of a "hyperknowledge environment" for DSS. This combines ideas from Model Management Systems (MMS), Artificial Intelligence (AI), Database Management Systems (DBMS) and hypertext. A DSS is seen as a mechanism for managing and exploring a complex world of descriptive, procedural, reasoning, linguistic, and presentation knowledge (data, programs, language grammars, rule bases, problem representations and display mechanisms, respectively). It is suggested that hypertext would provide an excellent interface to such a "hyperknowledge environment".

Balasubramanian et al. [BIKM91] suggest the use of hypertext to assist decision making in the context of portfolio management. Bieber [Bie91] and Bieber and Isakowitz [BI91a] use a logic approach to integrate DSSs with a Hypertext front-end. Balasubramanian et al. [BIJS92] describe how hypertext technology

matches the requirements of DSS in two senses (1) as an attractive style of user interface that should help users explore a complex problem space and (2) as a software technology that supports the development of both a model management system (MMS) and specific models. The focus in [BIJS92] is on the elements of a dynamic hypertext that would be required to support these goals.

In summary, hypertext has several advantages in the context of DSS applications: it provides an attractive interface that could reduce cognitive complexity for users and it also automatically handles the connectivity between related DSS elements. Adopting a broad view of DSS, these elements can include any information relevant to the decision process including memos, commentaries and e-mail messages as well as data and models. After a general discussion of the role of hypertext in DSS in section 2, we discuss the kinds of relationships that occur in DSS in section 3, from which we derive the requirements for a dynamic hypertext in section 4. An architecture for a hypertext-based “relationship management system” is presented in section 5. Section 6 describes a schema-based approach to the development of a relationship management system.

2 ROLE OF HYPERTEXT IN DSS

Hypertext has shown its effectiveness in a number of different software domains [Con87, Nie90a, RT88, Fri88]. According to [MS88], the necessary conditions or “golden rules” for the success of hypertext are that the application involves a large body of information that is organized into numerous fragments, the fragments relate to each other, and the user needs to use or view only a small fraction of these fragments at any time. Furthermore, while hypertext has generally been used in the context of information retrieval and presentation, it is also a natural metaphor for expressing problem and knowledge structure [MHRJ91, GPS91, BHW80]. In this paper, we argue that the golden rules apply in the case of DSS applications and that hypertext is a natural medium for expressing and managing DSS system and problem structure. In particular, we believe that a hypertext environment for DSS can help:

(1) provide all classes of user (from model developer to end user) with a uniform and effective interface to all objects that are created during the development, testing, and use of a DSS.

(2) provide cognitive aids to help the user access and view these elements and to travel from one to another in any order.

(3) reduce the tasks that have to be performed (and that have to be programmed into the DSS application).

We now briefly discuss the role of hypertext in helping to achieve each of these objectives.

2.1 A User-Interface for All classes of User

The user-interface is the central mechanism by which decision support is provided: it allows the user to explore the space of problems, techniques and solutions. Hypertext systems generally have sophisticated interface management (windowing) capabilities. Guide [GUI87] for exaple supports multiple scrollable windows. In addition, hypertext systems allow users to “browse” related chunks of information stored on different kinds of media, to add their own annotations to this information, and to create new relationships and “trails” through the “application domain”. Since DSS usage is inherently interactive and exploratory in nature, one would assume that hypertext’s support for such browsing activities would be useful in DSS. Only time will tell whether this supposition is correct; as discussed above, the possibilities of hypertext as an interface to DSS are just now beginning to be explored.

Dynamism and change are at the heart of the DSS concept. The objective of a DSS is to support decision making in semi-structured or unstructured decision situations which cannot be modeled by closed form algorithms. Rather, such situations require that the users be actively engaged in the decision making process [Spr89]. The development of a DSS often requires a prototyping approach with close cooperation between the *model developers* and *end users* [Kee80]. Furthermore, the use of a DSS is often a highly interactive process in which end users (or managers) engage in complex explorations of the model domain during which various assumptions about causative structure and model parameters are varied in an iterative fashion. This calls for a software environment that supports both the evolution and utilization of the DSS.

In this paper, we suggest an environment in which the user interface elements are provided as a given and are specialized and interrelated throughout the development and usage processes. Every design and development object has its own built-in display capabilities and is connected to related design elements via hypertext links. Thus, there is a process of “cascading” the interface down the design hierarchy until it is in place in the specific DSS that the user utilizes. For example, the traditional view of users interacting with a user-interface to generate reports is replaced by one in which users communicate with a number of DSS elements, including reports. Users can *question* the reports about their meaning and change some of the data or model elements to perform sensitivity analysis. To do this, the user communicates with the report, and with the Data and Model base elements associated with the report. This requires that the report itself contain user-interface elements - we propose that these consist of hypertext “buttons” that activate hypertext links to hypertext nodes in the model and data bases.

2.2 Hypertext as a Cognitive Aid

Following [KS92], we believe that there is a need to integrate all classes of information that are relevant to a given decision process. This includes such relatively unstructured and informal items as internal memos,

proposals, e-mail messages and reports, as well as the models and data elements that have been the traditional focus of DSS. It is precisely, this kind of diverse and complex information space that recent forms of hypertext system are designed to support [MO92]. Hypertext systems, offer a built-in facility to record and maintain many of the relationships between elements in the DSS domain together with rapid access to many diverse information sources. That is the good news. The bad news with regard to the use of hypertext systems (at least for users viewing textual material) has been the danger of being “lost in hyperspace”. Users can follow multiple paths in the text and become hopelessly lost (Nielsen [Nie90b]). To combat this, various cognitive aids in the form of “browser“ or “overview” maps showing a graphical model of the hypertext nodes and links and the users’current position and “voyage” to date, have been designed and are being actively researched (Utting and Yankelovich [UY89], Carmel, Crawford and Chen [CCC92]). Since DSS users HAVE to travel extensively between elements of the application domain as they explore the implications of their models, it seems reasonable to assume that they would benefit by the kinds of mapping support that is natural to hypertext systems. Any kind of overview map is probably better than none at all (which is the case in almost all current DSS).

2.3 Hypertext in Support of Application Programming

With regard to objective (3), important advances have been made over the years as functionality has migrated from the application software itself to independent “software layers” that are used by multiple applications. There are a number of stages in this historic development:

Operating systems relieved the application from having to deal with details of hardware device and file management.

Data base management systems (DBMS) relieved the application of much of the responsibility for logical and physical data management.

User interface management systems (UIMS) relieved the application of many of the chores associated with providing full screen management of text, forms, menus, and graphics. The highly popular “windowing” systems provide a standard interface for accessing such user-interface elements.

In the DSS area, model management systems (MMS) are being designed and developed with the objective of relieving the application of the much of the work associated with the development, maintenance and use of models.

An underlying thesis of this paper is that a fifth reduction in the necessary functionality of user application software is required:

We should relieve the application program of the need to maintain and provide access to the complex set

of relationships that can exist between elements in the application domain of DSS software.

We call this last form of software functionality “relationship management” and envisage an application development environment that provides device management, file management, data management, model management, interface management and relationship management. The extended hypertext, “H+” system described in this paper is designed to provide the latter two capabilities.

In summary, we believe that hypertext can provide an attractive interface to a DSS (objective 1) and that the overview maps that are provided by many hypertext systems can provide beneficial cognitive support for DSS users as they interact with the DSS system and explore the ramifications of their models (objective 2). Finally, with regard to objective 3, we believe that extended hypertext can relieve the application software of many onerous tasks in two areas: (1) interface management, and (2) relationship management -the maintenance of access paths through the complex maze of static and dynamic relationships that constitute the application domain of DSS models. The remainder of this paper is concerned with the potential of hypertext for providing relationship management.

3 RELATIONSHIPS IN A DSS ENVIRONMENT

Table 1 lists some of the data and program elements that must be created, manipulated, and communicated by a DSS system. As discussed later, each of these element types constitutes a potential node class in a hypertext system. Some of the elements are aggregates of other elements; for example, reports can contain formulae, attribute names, attribute values, etc. The elements in the table have been grouped into five “*components*” to show their relationship to the more traditional depiction of a DSS in terms of its component subsystems (Sprague and Carlson [1979], Bonczec, Holsapple and Whinston [1981]). In these traditional depictions, a DSS is thought of as being composed of three subsystems: a database management system (DBMS), a model management system (MMS), and a user interface management system (UIMS). The DBMS handles the data component, the MMS handles the models and problem solvers, while the UIMS handles the interface elements. We distinguish between models and problem solvers: models are either non-procedural statements that exist independently of the problem solver as in the case of linear programming models, or abstract descriptions used for communication and understanding when the model is embodied procedurally in the problem solver code as is the case in most simulation models. Some problem solvers execute standard algorithms and are provided initially with the DSS software; other special purpose programs are built by the model builder.

We have included a *communications* component that comprises the elements through which organization members communicate decisions and other information pertinent to the decision task associated with the DSS [AKN86]. This component has generally been considered to be outside of DSS. However, in our view,

DSS COMPONENT	Elements
DATA	relations, tuples, attributes, values, documentation
MODELS	models, formulae, variables, documentation
PROBLEM SOLVERS	programs, reports, documentation
INTERFACES	screens, menus, graphics, reports, documentation
COMMUNICATIONS	reports, commentaries, memos, e-mail

Table 1: Elements of DSS components to be presented within hypertext nodes.

DSS COMPONENT	PROB.				
	DATA	MODELS	SOLVS.	INTERF.	COMMS.
DATA	DBMS	MMS	MMS	*	*
MODELS		MMS	MMS	*	*
PROBLEM SOLVERS			*	*	*
INTERFACES				UIMS	*
COMMUNICATIONS				*	*

Table 2: Dependencies among DSS components and how these are maintained.

the memos, commentaries, and e-mail records related to the DSS in its organizational context are an integral part of the decision process and should therefore be supported by the DSS [KS92].

An automated DSS should support all of the activities associated with the data elements in table 1. It should also be capable of recording a history of the decisions it assisted in making, the rationale behind these decisions, and an evaluation of the results of implementing the decisions. Subsystems associated with each of the five components in table 1 can store, retrieve and manipulate the relevant elements in their own domain and usually contain their own *documentation* (offline or online) to guide the user. e.g., relationships between data are maintained by the DBMS, relationships between screen elements by the UIMS. However, the relationships between the different subsystems must be patched together by the *DSS software* to provide a coherent system for the user. In fact, some useful relationships (dependencies) are completely ignored by current DSS as shown by the *'s in Table 2.

There is clearly a need for a system to establish and support relationships that span the boundaries of the DSS components - the * in the table. In addition, wherever there is no independent subsystem to handle certain domain elements, a general system for relationship management is needed. For example, in the communications component, facilities to allow users to interrelate reports generated by the system and memos generated by users are not generally an integral part of a DSS. But such interrelationships can be important from the point of view of organizational coordination since many decisions depend on prior decisions and assumptions.

There are compelling reasons for adopting a DSS architecture in which DBMS, MMS, and UIMS subsystems are purchased or developed separately and then combined by additional software into a DSS system

as implied by table 2. These reasons include modularity and the separate historical development and availability of DBMS, MMS and UIMS software. Nevertheless, it is interesting to consider a second strategy in which the DSS software design problem is conceived, *ab initio*, as one in which the DSS has to manage all of the elements in Table 1 and their interrelationships. The major subsystems in such an architecture might be developed more on functional lines to handle expected DSS usage patterns and the separation between data, models, and interface elements that is inherent in current architectures might not be so profound.

In this paper, we suggest a third DSS software development strategy that involves the use of hypertext technology to develop a “relationship management system”. The “H+” system is designed to interrelate the various elements both within and between the traditional DSS subsystems. For example, documentation and user-help can be supported with hypertext links to the appropriate documents within each component, while reports and memos generated within the communications component can be linked to the results of specific runs within the model component.

Different types of relationships must be maintained by a DSS. There are relationships among elements of a decision environment that arise from the structure of those elements. We call these *structural relationships*. Relationships between the classes of DSS system elements (such as models, problem solvers, databases, and reports) that are shown in table 1 are of this kind. Other examples are given by menu and report structures, and the association between software components and their documentation in the form of help screens, etc. Many structural relationships (dependencies) are static and can be postulated a priori. Others, (for example abstract problem structures) rely on domain semantics and can only be determined through interaction with the user.

Process relationships arise through the interaction of model builders and model users with the DSS system. Thus, model builders construct a series of model prototypes that need to be tested and refined over time, while model users build a series of versions of these models and run the models using alternative data values during the model sensitivity analysis phase. In both cases, complex paths are traced that need to be recorded and made available so that back-tracking and browsing for the purposes of revision and learning can take place. The need to maintain process relationships implies that the H+ system should be capable of automatically adding instances of nodes and links to the hypertext network.

In database systems, it is important to distinguish between the the generic structure of the data (as defined by a database “schema”) and realizations or instances of that structure. The same is true in hypertext systems. As discussed in more detail later, most of the structural and process relationships we have mentioned can be constrained to follow a fixed pattern or schema to which new nodes and links are added over time. However, not all such relationships can be extensionally defined and realized through traditional embedded node and link structures. This gives rise to the need for *virtual relationships* that are intentionally determined by the user through what amounts to a querying process. There are two classes of virtual relationship:

(a) Multitudinous instances of relations within a given class of relationship. There may be relationships that can be anticipated as a class but whose instances are either too voluminous to record under normal circumstances and/or too uncertain to determine in advance. e.g. the model builder might well anticipate the need for users to trace backwards from solution results to particular input data values that give rise to those results. Special algorithms for such “impact analysis” can be provided within the DSS for a particular class of model - but this is difficult to do in general.

(b) Multitudinous relationships not all of which are explicitly maintained by the DSS. Thus, there may be unforeseen needs to traverse paths between application domain elements. e.g. the user might need to investigate the reasoning behind a particular model design and might wish to discover the author of the model. While it might be argued that the DSS builder should anticipate this need, there are simply so many possibilities for relationships between components that it may not be worth the DSS developers time to anticipate them all and to program the necessary user access paths.

We emphasize that the information base with which we are concerned is a complex web of relationships between all of the elements in table 1. This web serves to connect hypertext nodes representing the inputs and outputs of the various subsystems (database, MMS, Problem solvers, UIMS, and communications system) as shown in table 2. At the coarsest level of granularity, the hypertext nodes might consist of the actual files/documents that are the inputs and outputs of these systems. These nodes could simply be accessed via read-only or editing windows through which the text is scrolled as in the GUIDE system [GUI87] or NoteCards [HMT87]. In a sense, these nodes are *atomic* because the links do not penetrate the contents of the nodes. Hence, the internal structure of the DSS elements is ignored by the hypertext. At a finer level of granularity, if knowledge of the internal structure or logical content of documents such as problem solver reports and database files is provided, links between elements within these larger data structures could be made directly – either explicitly by planting anchors in the files or dynamically through the use of queries. Thus, every data item in an output report might be a “live” button – clicking the mouse on the item would open up a window through which the value(s) of relevant model parameters could be viewed.

4 REQUIREMENTS FOR RELATIONSHIP MANAGEMENT

The development and use of a DSS may involve multiple human and machine processes and generate many interdependent data and program elements that must be coordinated. The role of a relationship management system (RMS) is to support the establishment, maintenance and use of such relationships. We argue that this can most easily be accomplished by using an extended form of hypertext technology to maintain a dynamically changing “state space” that reflects the structure and history of the decision process.

4.1 Requirements

In this section we discuss eight requirements for relationship management and relate them to the hypertext capabilities that are needed for their support. As it turns out, all of the hypertext capabilities discussed below have been provided in either commercial or experimental hypertext systems, however, as far as we know, no single system as yet contains all the needed features.

4.1.1 High Level of Abstraction

As in [BIJS91, BK92], the RMS component sits between the user interface and the actual DSS application software (see Figure 1). It is not concerned with either the intricacies of user interface displays or the computational and communication processes associated with the DSS and other associated software. The RMS captures and represents generic relationships between elements in the application domain and supports two-way communication between the interface and the application systems. To be applicable in many different DSS applications, the system must represent such relationships in an abstract way and be unconcerned with the detailed contents of the nodes. Several formal models of hypertext engines have been defined that capture the important abstractions in a wide range of existing and future hypertext systems ([HS90, Lan90]. Such models define a high level of hypertext functionality and represent a step towards “open hypertext” in which hypertext document bases can be freely interchanged and hypertext systems become interoperable.

4.1.2 Rapid authoring

To support the total decision process as we envisage it, the RMS needs to manage the development and relationship of many diverse documents produced by modelers, data entry clerks, and so on, as well as the ancillary information types that are used directly by end users and are indicated in the “*Communications*” component shown in table 1. Much of this material consists of text and graphics that needs to be created, combined into documents, and interrelated. All hypertext systems provide some form of authoring capability. In addition to the features associated with modern editing systems (spelling checkers and so on), users are able to create buttons and referential links to key pieces of information both within the same document and between different documents. Some existing systems provide “idea processors” that allow documents to be produced *top-down from a starting outline* [HMT87]. Others allow document creation and composition within the framework of a schema describing the classes of relationship that are possible (see below for a more complete discussion.) To help coordination between different DSS users, and to foster joint exploration of the problem space, many hypertext systems also allow users to add their own annotations to hypertext nodes [YMvD85].

4.1.3 Automated establishment and maintenance of relationships

While manual authoring capabilities are essential for the reasons indicated above, the hypertext network required to capture all of the structural and process information in the DSS application domain is so large and complex that it would require significant implementation efforts if implemented manually. We propose to automate the creation and maintenance of the majority of nodes and links in the network by making extensive use of *schemas* to describe how elements in the DSS application domain are to be organized into a hypertext network. Some of these schemas describe the DSS software itself (e.g. hypertext networks for online help, version control, sensitivity analysis, etc.), while other schemas define data and process relationships specific to individual models (e.g. the data structures in a transportation model). We call the totality of schemas used to describe the application domain, the “conceptual model” of that domain.

Similar schemas have been proposed in the area of hypertext authoring. Smith et al [SCGL91] propose the use of *templates* to incorporate pre-defined structure into hypertext documents. Knowledge-oriented hypertext systems use schemas to represent the structure of knowledge within the hypertext node and link structure [GPS91, NN91, KS91, MHRJ91]. Still others use hypertext schemes to represent arguments and reasoning processes [CB89, MHRJ91]. A common feature is the ability to classify elements into a taxonomy of types and to define *schemas* to represent classes of structures. Instances of these schemas are populated as the system evolves.

In the context of creating conventional hypertext, schemas are associated with “authoring-in-the-large” i.e. defining the global structure of the hypertext document - the way the information base is structured and the navigation paths made available to the user. In contrast, authoring-in-the-small involves composing the text, drawing the graphics, and so on, that constitute the nodes. It is useful to restate these concepts in the context of DSS relationship management as we envisage it. Here, authoring-in-the-large, is performed by the tool-smiths and modelers who essentially define the element types in table 1 and the relationship types that are needed to fill-in the gaps in table 2. Authoring-in-the-small is performed by the modeler when new models or simulation programs are written, by the DSS programs or problem solvers that produce results to present to the user, and by managers and others who compose memos, recommendations and proposals and communicate via e-mail.

While the schemas can define important paths through the hypertext network, there will still be a need for users to traverse unanticipated paths and to view and manipulate information of finer granularity than that described by the schemas. Halasz [Hal88] pointed out that the hypertext model should incorporate *virtual structures* which are nodes and links that are not extensionally defined within the system but are generated as needed. Essentially, virtual links are database queries. The system allows the user to use either a keyword scheme or full text search to link to and access related documents (or the internals of documents).

4.1.4 Support exploration within the application domain

A significant part of problem solving involves exploring the problem and solution spaces. For example, a user might want to locate the data used to reach a particular questionable decision presented in a memo by following the chain of documents that led to the decision. The RMS must support the exploration of such trails. Besides the user's natural desire to explore, the system itself needs this ability. For example, to support explanation, the system should be able to keep a log of all steps involved in important decisions and be able to retrace these steps on demand. Such exploratory capabilities are naturally provided by hypertext's navigation mechanisms. The links among nodes (which represent relationships) can easily be selected and traversed by both the user and the system.

Two somewhat related problems have been experienced when users search hypertexts for information. First, hypertext allows users rapid access to information that may or may not be relevant to their particular task producing a tendency to information overload [Nie90b]. How users can combat information overload by bringing structure to large information spaces remains an open question. The second problem, network disorientation, or "getting lost in Hypertext" (Nielsen [Nie90b]) arises because hypertext approaches tend to fragment information; users lose track of where they are, and can't decide either how to return to a previous state or which information nodes they should visit next. This problem has been the subject of much research and a number of mechanisms have been devised to support the user's exploration. These include automatic backtracking, recording user "trails" through the hypertext, allowing the user to place "bookmarks" in the text, providing "land marks" or nodes that are easily recognized, "departure and arrival protocols" that help orient users by reminding them where they were on exit from a node and where they have arrived on entry to a node, and the use of "browser maps" and "graphical browsers". Browser maps are graphical displays of the hypertext network that allow the user to traverse the network by directly selecting links and nodes (Utting and Yankelovich [UY89]). See chapter 5 in [Nie90a] for a discussion of these and other cognitive aids for browsing and search.

4.1.5 Coordinate computation within the application domain

Certain relationships represent information flows. For example, a model, its data (or scenarios) and related solvers are combined in a stream to produce solutions. Hence, the relationships among these elements (the model, the data and the solver) do not simply specify dependencies, they represent parameter passing agreements that determine the information flow required for solver execution. A user should be able to create a "run" relationship and cause the model to execute by pressing a hypertext button associated with the visual representation of the run. Computational hypertext (Halasz [Hal88, Hal91]) can provide this functionality. Halasz differentiates between two kinds of computation: "computation over hypertext" and "computation in hypertext". "Computation over hypertext" takes place when hypertext calls external programs via

special links (DSS Shell [Bie91], PHIDIAS [MBD⁺90]). There are two forms of “computation in hypertext”. In the first, the hypertext itself performs computations to alter node contents using scripting mechanisms (Webtalk [NN91], Schnase and Legett [SL89]). In the second form, the computation imposes “browsing semantics” to control the paths through the hypertext and what is displayed on the screen. For example, when a “run” relationship is accessed, the browsing semantics might also automatically open windows on the screen to display the model equations and data tables associated with the run. The Trellis model [SF89] uses Petri nets to support browsing in this way.

The kind of computation required for relationship management, which we call computation **within** hypertext, is a combination of the above forms. Note that the RMS need not perform complex computations; that is the responsibility of the DSS applications. The RMS however should take the responsibility of coordinating computation by transmitting parameter values between nodes and issuing calls to DSS applications. It is in this sense that we require a combination of Halasz’ two kinds of computation. The computation alternates link traversal (which transmits information) and external program calls.

Links that can transmit information are called *valuation links* [BIKM91], Isakowitz [Isa92] and in Bieber and Isakowitz [BI91b], because they transmit values. Upon activation of a valuation link a propagation process is initiated. Information flows from the activated link to the destination node(s). Valuation links originating in these nodes will, in turn, propagate information to other nodes, and so on. Note the difference between *valuation links* and *action links*. The latter represent calls to external procedures; the former represent transmission of information. For example *active reports* can be implemented using valuation links that establish the flow of information among various nodes. Each active field in an active report is associated with a valuation link that connects the field to a node. When the value at this node changes, the valuation link is activated and the report updated.

Note that in advanced hypertext systems, links are more than just textual cross-references. In general, when a link is activated, a complex series of operations can take place: locate from node, execute inference/computation procedure, locate to node, execute inference/computation procedure, display results.

4.1.6 Provide semantics for abstract relationships

The requirement for a high level of abstraction means that conceptual models of the domain have to ignore details pertaining to domain knowledge. Therefore, relationship management manipulates abstract entities and abstract relationships among these entities. There is however, a correspondence between these abstract representations and their domain counterparts: abstract entities correspond to domain elements, and abstract relationships correspond to domain dependencies. This correspondence needs to be maintained, so that appropriate meaning can be attributed to the abstract entities.

Hypertext documents are usually highlighted by visual “anchors” or “buttons” that indicate that the associated portion of the text is related to other information. The relationships can be of many types that are represented by hypertext links of various types. Usually, the type of a link is indicated by the shape or color of the button or by a short string of text. The commonest link type is one-to-one reference - a single related item of information is displayed in a window on the screen when a button is activated. However, one-to-many and many-to-many links are also supported by some hypertext systems [BBK88].

The following is a minimal list of link types that are needed to support relationship management in the DSS domain.

1. *Is a part of*: elements are grouped into components, which in turn can be grouped into higher level components. This link has a *hierarchical* nature.
2. *Reference*: used to associate semantically related domain elements. For example, a *model variable* and the *variable definition* which contains detailed information about its meaning.
3. *Action*: some relationships, such as the *run* described above, refer to actions in the domain. An *action* link is associated with a specific program in the DSS domain. Activation of the link triggers the execution of the program.
4. *Valuation*: A final relationship type that is important in the DSS arena involves the transmission of information between domain elements via *valuation links* as explained above.

Hypertext can support these constructs through various link types. For example Guide [GUI87] supports the first three link types and [Isa92] the last.

4.1.7 Support for distributed group work

Increasingly, organizations need to coordinate distributed decision networks with ever faster cycle times [Hub84]. A well-known example is the attempt to coordinate marketing, design and manufacturing decisions in the area of operations management. Providing decision support in such dynamic environments requires heavy use of the communications elements listed in figure 1. This is a relatively new area in hypertext research, however at least one system, Hypernet [MO92] provides distributed support for a world-wide hypermedia database linked through the Internet network. Furthermore, Hypernet supports cooperative authoring, editing and browsing of hypermedia documents in both synchronous and asynchronous modes.

1. High Level of Abstraction
 - Independence from application and user interface
2. Rapid Authoring
 - Authoring-in-the-small: node and link creation, editing text and graphics
 - Authoring-in-the-large: schemas
3. Automated Creation and Maintenance of Relationships
 - Schemas, virtual nodes and links
4. Support exploration in the application domain
 - Node and link traversal, query capability, annotation
 - Cognitive aids: backtracking, book marks, browser maps
 - Ability to define browsing semantics
5. Coordination of computation
 - Computation “within” hypertext: valuation links, calls to external applications
6. Provide semantics for abstract relationships
 - Multiple link types
7. Support for distributed group work
 - Distributed hypertext: synchronous and asynchronous access
 - Information sharing: multiple users accessing the same information
 - Individual user views or contexts
8. Version control
 - Automatic logging of node and network versions

Table 3: Relationship management requirements and hypertext features

4.1.8 Automated Version Control

DSS applications run the gamut from simple spreadsheet models to complex multiperson systems that require considerable resources to develop and maintain. Regardless of size and scope, prototyping is the predominant development paradigm. A major objective of model management systems is to provide a software environment that can help coordinate multiple prototypes and the requirements of multiple users. Although the emphasis in hypertext is on authoring and composition of document collections rather than on the development of software, similar problems obtain with regard to version control and support for multiple users. Several hypertext systems provide automatic logging of node and link versions [DS87]. Hypertext systems that support collaboration have developed mechanisms whereby individual users are provided with network “contexts” or “views” which they can explore and manipulate without endangering the integrity of the underlying hypertext database [GB87].

4.2 Summary of hypertext features for relationship management

We have discussed the features required to implement relationship management and shown how existing hypertext-based systems have incorporated these features. Table 3 summarizes this discussion.

While no existing hypertext system contains all the features listed in the table, very general, “open”

	Static hypertext	Dynamic hypertext
hypertext activity	browsing, annotation	computing
links represent	hierarchy and cross-reference	transmission of information
network layout	manual	automatic through <i>schema</i> , virtual nodes and links

Table 4: Comparing static and dynamic hypertext.

hypertext engines are currently under research and development. Table 4 summarizes the main differences between the traditional hypertext, which for lack of a better name we refer to as *static hypertext*, and the dynamic hypertext that is now emerging and is needed to support our concept of relationship management in the DSS domain. Dynamic hypertext has all the properties shown in the static hypertext column plus those listed in its own column. The DSS Shell [BBK88] represents a path-breaking attempt to implement many of these features in the model management arena.

The remainder of this paper describes a schema-based approach to the use of general purpose hypertext software as an integrating mechanism in the DSS domain. Although no commercially available hypertext systems currently provide the dynamic, knowledge-oriented functionality implied by tables 3 and 4, some research systems do approach it, e.g., Aquanet [MHRJ91] and HDM [GPS91].

5 AN ARCHITECTURE FOR RELATIONSHIP MANAGEMENT

Figure 1 depicts a proposal for an extended hypertext system H^+ to implement relationship management. H^+ stands between the user interface which handles presentation functions such as screen layout, and windowing, and the application domain programs which perform computations, update databases, run models, and so on.

A significant portion of the relationships between elements in the application domain are represented in H^+ using *schemas* which are initially input to the *Schema Engine*, *SE*. The hypertext schemas perform a similar function to database schemas. In addition, *SE* is able to interpret the schema and to produce information that is used by H to generate buttons, design window layout, etc. for each node class in the network.

The communication between the DSS and the H^+ is achieved via these schemas. As the decision process evolves, the DSS populates the schema by sending *state-change messages* to *SE*. The *SE* interprets these messages in terms of the objects present in the schemas and sends appropriate messages to the *Hypertext Engine*, *HE* which is responsible for managing hypertext links and nodes, and for browsing operations. Internally, knowledge about the schemas dictates what commands to send to *HE*. *SE* is responsible for transforming

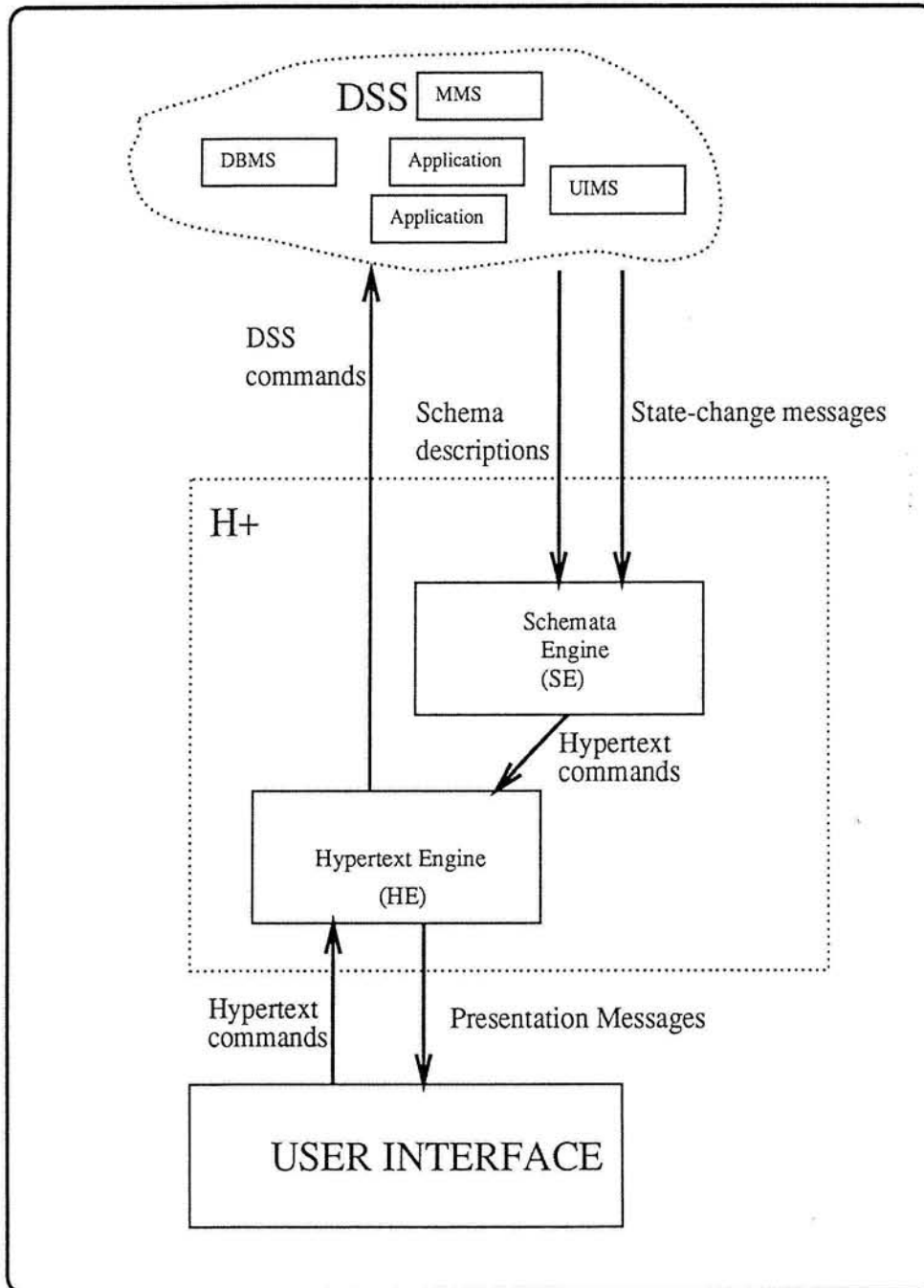


Figure 1: An architecture to support schemas.

knowledge structures, e.g. schemas, into hypertext structures. It is also responsible for understanding *actions* on schemas and interpreting these as hypertext activities. For example, adding a relationship to a schema, results in adding a link to the hypertext.

A mechanism to share contents of nodes (mainly text and figures) between the DSS and H^+ needs to be implemented to support this architecture. For example, both files and pointers can be used. The H^+ system combines the SE and HE into one system and maintains the state of the DSS as a hypertext network with which the user interacts via built-in navigational tools. More formally, if we describe the DSS as a finite state sequential machine, [HS66]:

$$D = \{S, I, O, d, o\}$$

where: S = set of states of the DSS

I = set of user inputs

O = set of outputs from the DSS

$d : S \times I \mapsto S$ is the state transition function

$o : S \mapsto O$ is the output function

Then we need to design H^+ as a state machine:

$$H = \{S', I', d'\}$$

where: S' = the set of hypertext nodes and links

I' = set of input messages from D

$d' : S' \times I' \mapsto S'$ is the hypertext transition function,

Since we only need to consider certain states and actions of D as being of interest to the user, we will assume that S' is isomorphic to a subset of S and I' is isomorphic to a subset of I . S' is represented by the hypertext network topology and the contents of the nodes; I' is the communication protocol language explained below; d' represents the functions (to add nodes and links, traverse the network, etc.) that are associated with the H^+ system.

The remainder of the paper shows how the functionality provided by advanced hypertext systems can be used to augment and broaden the capabilities of model management systems. We concentrate on the flows of information between the application software and H^+ that are shown in figure 1. i.e. on the schema descriptions, state-change messages, and DSS commands. Essentially, we need to demonstrate that the proposed architecture supports the full functionality of the DSS and adds the functionality attributed to relationship management systems in the last section.

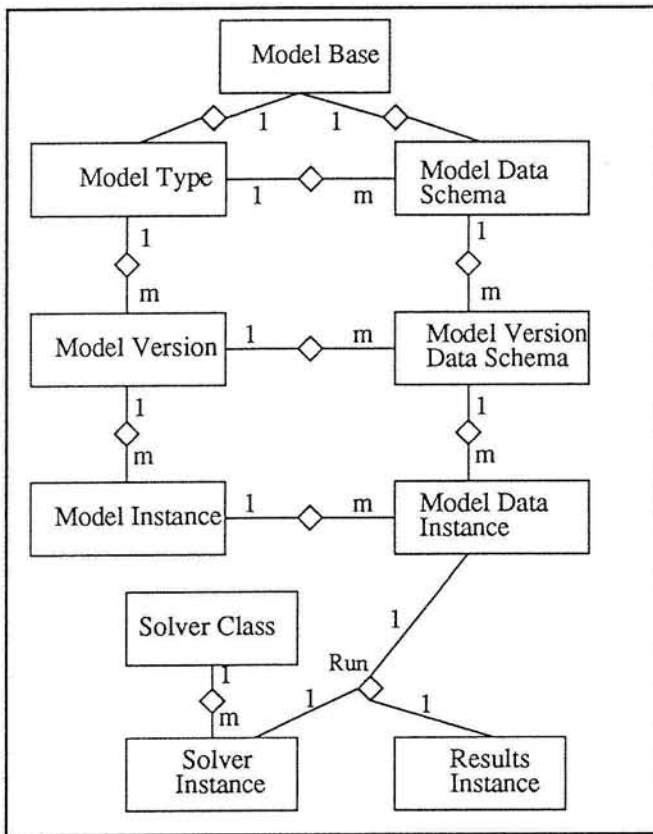
6 RELATIONSHIP MANAGEMENT THROUGH SCHEMAS

At the highest level, the user views the application domain via a hypertext network in which each node represents a DSS application area. Accessing a DSS application node at this level moves the user to a lower layer of hypertext networks that are associated with the application area and so on in a recursive fashion. Thus, each DSS application area is represented as a nested series of hypertext networks that are themselves connected to each other in a larger hypertext network. As work proceeds in the application domain, state change messages are sent to SE which then adds, deletes and modifies node and link instances in the hypertext network that conforms to the pattern defined by the associated schema.

The main idea is to use a “generic schema” from which instances of schemas are generated. We provide four examples in this section. We use the entity relationship (ER) formalism [Che76] to represent schemas in this paper, as it is the first, and probably best known example of a language for defining conceptual data models. In the hypertext domain, there are a number of more specialized schema languages that might be better suited (e.g., [GPS91, MHRJ91]). The ER graphical representation of the schema can be displayed as an “overview map” in a window on the user’s screen with the currently active entity set highlighted.

Example 1: Our first example, is a schema that might be input by the toolsmith - the person who builds the DSS system initially. The schema supports model version control and sensitivity analysis. The schema for our example is shown as an Entity-Relationship Diagram (ERD) in Figure 2. More elaborate] models have been proposed in the literature (e.g., Stohr and Tanniru [1979], Muhanna and Pick [1990]) but the model shown in Figure 2 is sufficient for our purposes. The following is a brief explanation of the entities (represented by rectangles) in the figure.

- **Model Type:** This entity class is used to represent generic models (e.g., a transportation model). Each instance could consist of one or more documents containing a template model in algebraic form, plus an explanation of the model, its data elements, and potential applications. Certain generic models would be provided with the system and stored in the “model base” of the DSS.
- **Model Data Schema:** Records the relationships between the data elements in a generic model. For a transportation model, this schema shows the relationships between source supplies, sink demands and route transportation costs. These relationships could be stated in a number of different forms (e.g. as a “structured model” (SM) definition (Geoffrion [Geo87]), or as an entity-relationship diagram (ERD).
- **Model Version:** The model type specialized to the particular user’s problem by the modeler. Assumptions underlying the generic model might be varied (e.g., a capacity constraint might be added to the classical transportation model). The model version is accompanied by explanatory material provided by the modeler.




Key: 1  m one to many relationship

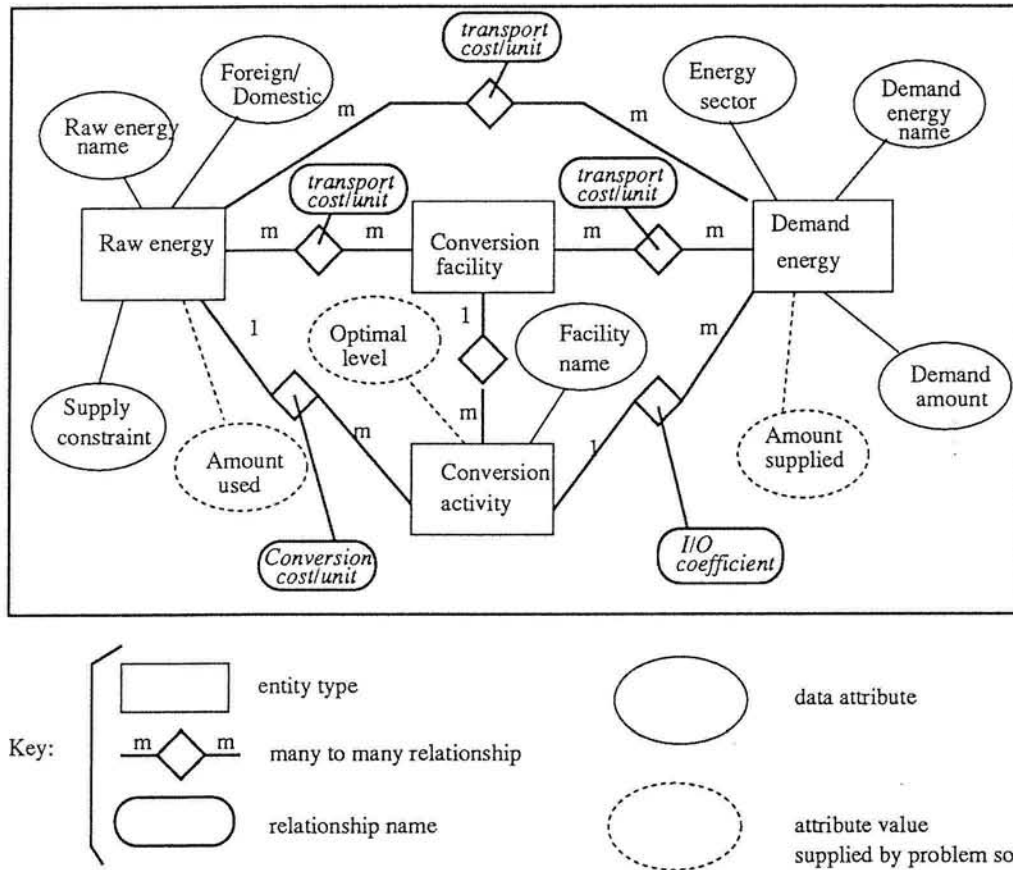
Figure 2: Versioning and sensitivity analysis schema

- **Model Version Data Schema:** The data schema corresponding to model version. It is the same as the parent model type schema but adjusted to match the structural changes in the model version.
- **Model Instance:** The model version is further specialized by modifying data and variable names in the version template to fit the application (i.e. to “bind” the model to its data). Again, explanatory material can be provided by the modeler.
- **Model Data Instance:** The model version schema is specialized to include access paths to the values of the data items that will be used in a run of the model. Since data values can be updated without invalidating a Model Instance (assuming that all data is referred to by name in a Model Instance), there can be a one-to-many relationship between a model instance and the various compatible data instances.
- **Solver Class:** A family of computer programs that is capable of solving a given class of problem. For example, the simplex algorithm, generalized transportation algorithm and specially designed transportation algorithms are all capable of solving problems that can be classified as members of the transportation class.
- **Solver Instance:** Choice of a particular solver for use in a given model run.
- **Results Instance:** A model instance together with a compatible data instance are run with a particular solver instance to produce a result instance (report).

The relationships in the schema in figure 2 are represented by arrows with small diamonds. The “run” relationship is information bearing in the sense that it would have data attributes associated with it that describe the time and other circumstances pertinent to a particular run of a model.

A database developed according to the above model and maintained by the application software (MMS in this case) can provide many of the needed services associated with version control and tracking of trial runs during sensitivity analysis (Stohr and Tanniru [1979]). Our aim in this section is to show how most of the burden of maintaining this database and supporting user access can be shifted from the application software (or MMS) to the relationship management system.

Example 2: The second example is a schema that describes the essential relationships in a production-distribution model (see figure 3). This is a complex problem which can be portrayed as a linear programming problem. In this case, the schema is entered by the model builder. It is, in fact, an instance of the Model Type Schema shown in figure 2. As mentioned above, the schema could be entered in SM or ERD format. In this case, we choose the latter as shown in figure 3. From the schemas in figures 2 and 3, H^+ can automatically provide unique node types for each entity (and information bearing relationship class) containing hypertext “buttons” to allow access along each identified relationship in which the entity participates. It is also desirable



Problem description:

A national energy planning office wishes to determine how much of each of three types of raw energy (oil, gas, coal) to purchase from foreign and/or domestic sources to satisfy the demands of residential, industrial and transportation activities. Raw energy is converted to other forms by conversion facilities such as power plants and refineries. Each such facility converts one form of raw energy to one or more output forms of energy. Transportation costs are involved in bringing the raw energy to conversion facilities and consumers. The goal is to satisfy demand at minimum cost.

Figure 3: The energy production-distribution model.

to have “sibling” buttons in each entity class window to allow access to the “next” and prior instances of that entity. To illustrate, in example 1, a “run” window has (named) buttons leading to the associated model, data, solver and result instances and sibling buttons leading to the next and prior *run* occurrences. We will not discuss navigation semantics or the presentation level in this paper, but one can imagine users accessing a *run* instance node and clicking buttons to open an edit window for the data instance and a viewing window for the results instance. They could then repeatedly click on the sibling button to browse through *runs* instances to compare the results obtained from alternative data values. The button to the solver instance could have a label (the name of the solver) and two targets - one a model selection menu and the other initiating a computation with the selected problem solver.

Note that an instance of the network in example 2 corresponds to an instance of the Model Version Schema node in the network of example 1. As such, it inherits all the links which are defined for model version schemas. The example 2 network can automatically be converted to a relational database schema with one relational table for each entity and each relationship. The Model Data Instance node in example 1 will then be associated with a number of relational tables. During sensitivity analysis, at least conceptually, if not in physical reality, there will be one set of data tables for each trial run.

Example 3: Context Sensitive Help The schema is originally entered by the toolsmith who builds the DSS. It represents a graphical view of the software architecture augmented by nodes representing sections of the code for which help screens are provided. As the DSS executes a module for which a help screen exists, it outputs a state-change message to a history file. If the user presses a help function key, the Context Sensitive Help Hypertext Network is activated, the last state transition message is read from the history file, H+ accesses the indicated hypertext node, and displays the associated help screen.

Example 4: Computer Conferencing The schema is input by a DSS support person or intermediary and represents a hierarchical breakdown of the subject areas important to a given decision process. As modelers and managers interact in the decision process, they can add messages and model output reports to “conversations” under each topic area node. Hypertext facilities allow managers and others to browse the conference, annotate the text as desired, add referential links to create new trails through the network, or even to work cooperatively [MO92].

6.1 A Language for Schemas

Independently of the mechanism utilized to implement the SE, a language is required to express the schemas and the state-change messages. We describe the elements of a primitive language for expressing schemas in this section. Note that in addition to enabling the expression of relationships among nodes, the language has to support the association of relationships to DSS-application functions (see section 4.1.6). This will launch appropriate DSS applications when relationships are activated.

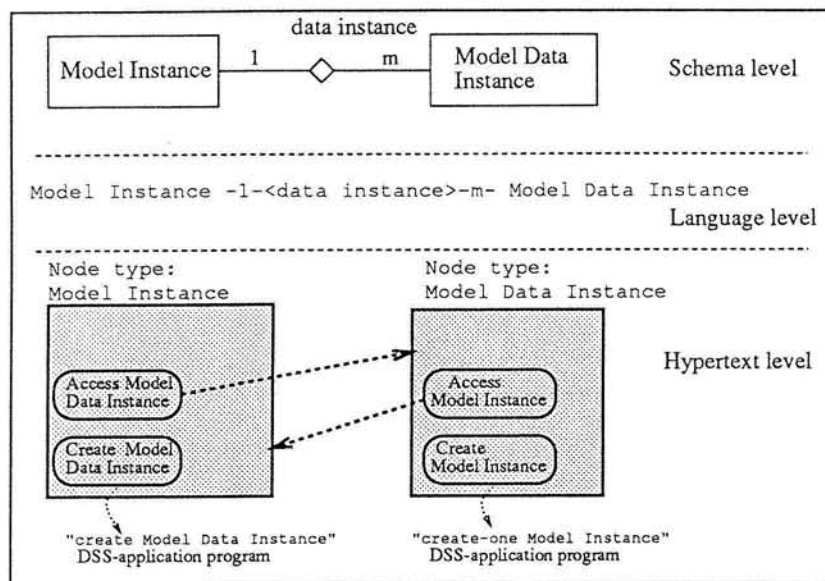


Figure 4: From schema to hypertext

Figure 4 illustrates how a relationship migrates from a graphical schema to a hypertext structure. At the **schema level**, the graphical schemas express relationships among domain elements. At the **language level** the same relationships are described syntactically. Finally, the **hypertext level** represents the relationships with generic constructs consisting of nodes, buttons and links. The last step, from language to hypertext, is performed by the SE.

As the figure shows, the SE deduces that nodes of type “**Model Instance**” have two buttons: “**Create Model Data Instance**” and “**Access Model Data Instance**”. The first button is used to create new **data instances**. It is associated with an action link that executes the “**create Model Data Instance**” routine of the DSS. The second button is used to access these “**Model Data Instances**”. Similarly the SE deduces that nodes of type “**Model Data Instance**” have the two buttons shown in the figure. Note that the “**Create Model Data Instance**” and the “**Create Model Instance**” buttons are associated with different DSS programs. This is because the declaration of the relationship **data instance** restricts to one the number of “**Model Instances**” that can exist for each “**Model Data Instance**”. In this example, the constraint is enforced by launching a special DSS routine called “**create-once**” that ensures that only one such “**model instance**” will ever be created. This constraint can be enforced at the SE or DSS levels. Here we chose to do it at the application level. The DSS routine “**create**” however, allows many data instances to be created for the same “**Model Instance**”.

Although this example concerns only one relationship, the general case, where a schema contains multiple relationships, is handled by iterating the process for each relationship in the schema. More generally, given a relationship:

$$A-i--<L>--j--B$$

Buttons in node A	Link name	Link type	Semantics
$j = 1$			
access B	$L \mapsto B$	reference	browse
create B	$X(L \mapsto B)$	action	DSS application call <code>create-once-B</code>
$j = m$			
access B	$M(L \mapsto B)$	multiple endpoint reference	browse
create B	$X(L \mapsto B)$	action	DSS application call <code>create-B</code>

Table 5: Hypertext functionality associated with schema relationships.

table 5 dictates the buttons and links to be implanted in nodes of type A . By reversing the roles of A and B , the table also indicates the buttons and links for nodes of type B . A link $L \mapsto B$ is a referential link to a node of type B ; and $M(L \mapsto B)$ stands for a link with multiple endpoints of type B . Links named $X(-)$ are *action links* that issue calls to external applications. Note the difference between “create” buttons for $j = 1$ and $j = n$. As explained above, in the first case the routine “`create-once- B` ” is launched, while “`create- B` ” is used in the second case. To illustrate how the table works let us consider the relationship

Model Instance --1--<data instance>--m-- Model Data Instance

from figure 4. According to the table (using $j = m$), every node of type “Model Instance” will have two buttons:

1. A “Create Model Data Instance” button that enables the creation of nodes of type “Model Data Instance”. This button is associated with an action link labeled “ $X(\text{data instance} \mapsto \text{Model Data Instance})$ ” which upon activation, issues a call to the external DSS application “`create-Model Data Instance`”.
2. An “Access Model Data Instance” button that provides access to all “Model Data Instance” nodes that are “data instances” of the current one. This button is associated with a link labeled “ $M(\text{data instance} \mapsto \text{Model Data Instance})$ ” that has multiple end-points: one per each data instance of this node.

To find out the buttons and links to be associated with “Model Data Instance” nodes, we reverse the relationship:

Model Data Instance --m--<data instance>--1-- Model Instance

The table is now used with $j = 1$. It indicates two buttons: an “Access Model Instance” button associated with a *one-destination* reference link; and a “Create Model Instance” button associated with the “`create-once Model Instance`” DSS application.

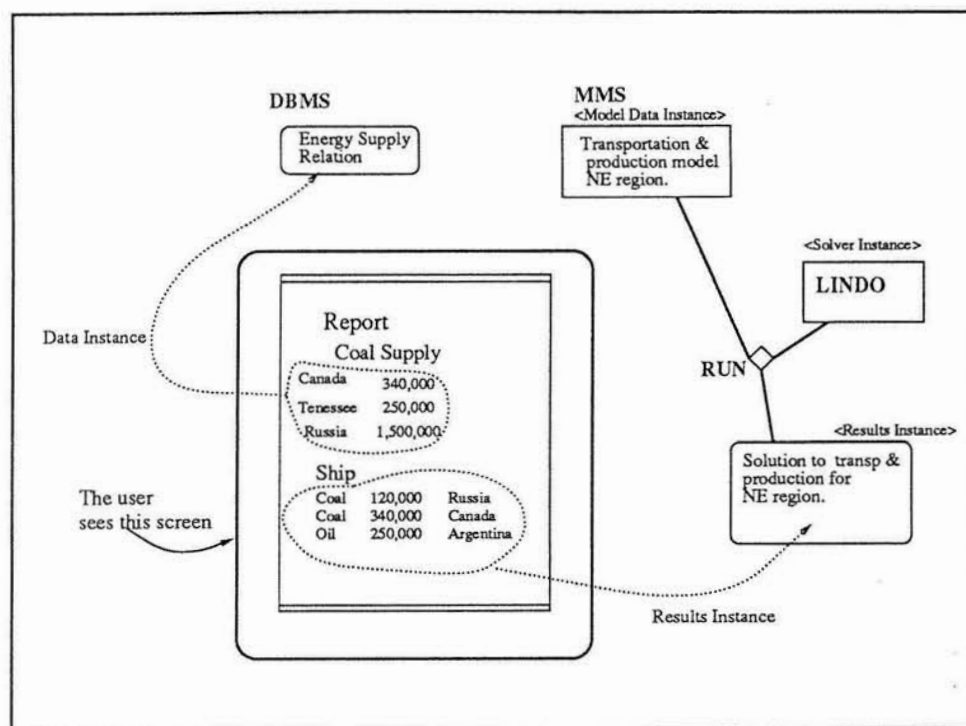


Figure 5: A portion of a report describing a solution to the energy problem.

We have presented a simple language to syntactically represent schemas, and we have explained how to *compile* it into hypertext constructs. Although this is not meant to be a complete design, it demonstrates the feasibility of our approach. The discussion also exposed one of the roles the schema engine (SE) plays in the overall architecture. It acts as a *compiler* translating schemas into hypertext constructs (also called “schemas” in Aquanet and HDM). A second role of the SE, upon which we will elaborate in sections 6.3 and 6.4, relates to the propagation of DSS events to the hypertext.

6.2 Representing more intricate relationships

The kinds of relationships we have represented in the previous sections are of the *referential* kind: one object referring to another one. A different kind of relationship is *inclusion*: several objects being part of another one. For example, a report contains numbers obtained from computations and data from the Data Base. Hence *computed values* and *data* are objects of the DSS-application domain which are part of another object: the report. Figure 5 illustrates this. The report shown is a solution to the previously mentioned energy problem. The portion of the report shown contains the input data for the supply of coal, which originates in the Data Base; and the computed solution obtained from a specific execution of the model. The figure shows that the elements within the report can themselves participate in relationships.

The issues that arise from such rich interaction albeit complex, are manageable within knowledge-oriented

hypertext systems such as HDM or Aquanet. From the decision making perspective, such composite objects provide the functionality to *drill-down* elements of the process. A report can be broken into pieces and each piece analyzed in greater detail. The support of such capabilities only requires the availability of schemas describing the structure of reports. In this way, H^+ is able to present and manage information at various granularities. A link might associate an entire report to another node, as well as portions of the report to portions of other nodes. It all depends upon the kinds of schemas supplied to H^+ . This ability to parametrize information is another of the powerful aspects of the schema approach to RMS.

6.3 A Communications Protocol

A complex sequence of state transitions takes place in the application domain as a user develops a model by testing multiple versions and runs these versions using multiple model instances and data instances. By carefully tracking these state transitions it is possible to construct a realization or instance of the schema in figure 2. Our strategy is to have the schemas engine (SE) system (rather than the DSS) dynamically build and maintain a replica of the actual network of relationships that are generated by the user interaction. Every time an event or state transition occurs, the DSS simply passes a descriptive message describing the event to the SE. The SE interprets this message using the schema and the past history of events, to decide the next addition, if any to its network. As state transitions occur in the application domain, the DSS sends messages to H^+ according to the following format:

```
Current-operation :=
[Create-object|Modify_object| Delete_object|....]
State (= object type from schema)
User-supplied object instance name
[<attribute name= attribute value>,...]
```

Any of the lines of the message might be a procedure instead of a value. For example, instead of supplying a name for an object, the system might specify that the user should be prompted for one. For example, the following message originating in the MMS prompts the SE to create a new model instance named "Energy T&D for NJ". It also resolves the "model-version" relationship by specifying the name of the model of which this is an instance. The contents, the actual text and graphics, are passed with a pointer.

```
CREATE
MODEL INSTANCE
'Energy T&D for NJ'
[Model-version = TRANSP+DISTR MODEL
```

Contents = <ptr to contents>]

Communication is realized as follows.

- **Establishing the communication.**

The toolsmith, who first builds the DSS system, and later model builders, who develop models using the DSS, provide the H^+ system with schemas of the application domain.

- **DSS initiated events.**

The results of an activity of a DSS application are propagated to the user. This involves the following steps:

a-1 The DSS causes events (state transitions) to occur in the application domain.

a-2 Messages from the DSS to the H^+ system record these state transitions.

a-3 The H^+ system uses the conceptual model of the application domain (a collection of schemas) plus the messages sent by the DSS to construct and maintain a network of the state transitions that have occurred to date.

a-4 The schemas allow the H^+ system to infer virtual relationships as required by the user or the DSS.

a-5 The H^+ system directs the user-interface (UI) to present appropriate aspects of the current application state to the user.

- **User initiated events.**

b-1 The user interacts with the user-interface by selecting, clicking or typing text.

b-2 The user-interface informs H^+ about the user actions.

b-3 H^+ either handles the request itself – if it does not require involvement of the DSS – or crafts an appropriate message to the corresponding DSS application.

b-4 The DSS application carries out the request of the user.

Notice that user initiated events do not directly impact the conceptual model because it is a reflection of the application domain, not of the user's perception of it. Only the DSS applications manipulate the schemas via state-change messages.

6.4 An example

We illustrate the proposed communication protocol using the energy production-distribution problem.

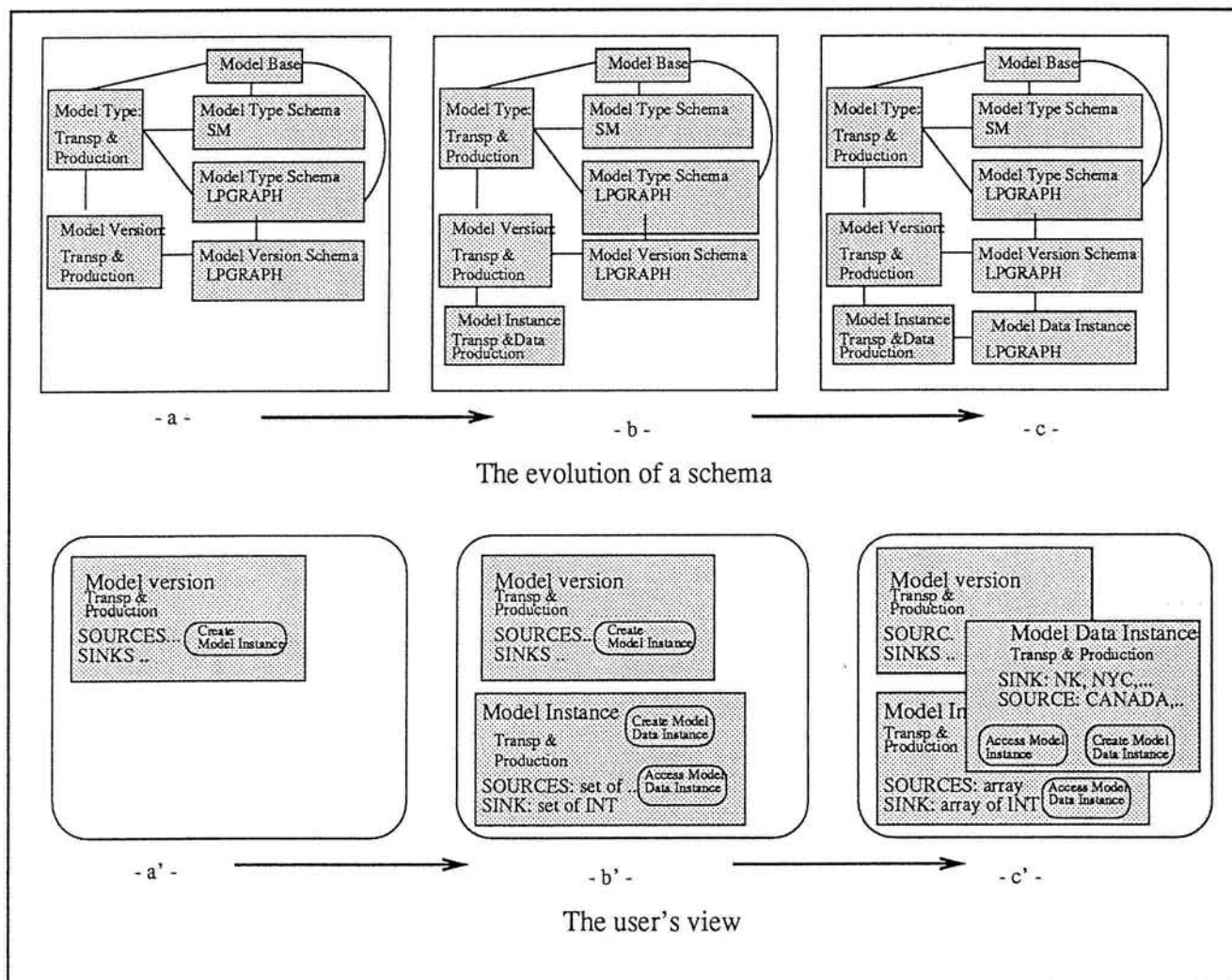


Figure 6: The evolution of a schema and the user's view.

The transportation-distribution model is developed top-down. Let us assume that through interaction with the user, the DSS has evolved into the state depicted in figure 6-a. Two schemas have been defined for generic models, one using structured modeling [Geo87] and the other expressed in a graphical language such as LPGRAPH [MMS89]. The *model version* used is identical to the *model type*, i.e. no additional constraints imposed. The user is viewing the screen as shown in 6-a'. The steps to create a *model instance* are illustrated next.

1. The user initiates the request by clicking on the *Create Model Instance* button on the LPGRAPH version. This button is deduced from the Schema by the SE. The user-interface is in charge of identifying the button activated. It sends a message to the HE system indicating which button has been activated, e.g., *button 09567 activated*.
2. H^+ retrieves all links originating in *button 09567*, in this case only one link exists. This is an *action* link associated with the message:

```
CREATE
MODEL INSTANCE
<PROMPT THE USER FOR A MODEL INSTANCE NAME>
[Model-version = TRANSP+DISTR MODEL]
```

3. H^+ first prompts, via the user interface, for the name of the model instance to be created e.g., *"Energy T&D for NJ"* and then sends the complete message to the DSS model management application:

```
CREATE
MODEL INSTANCE
'Energy T&D for NJ'
[Model-version = TRANSP+DISTR MODEL]
```

Now the Model Management System (a DSS-application) reacts to the message by creating an object to represent the *Energy T&D for NJ* model instance of the distribution and transportation problem. It then proceeds to

4. send a message to H^+ :

```
CREATE
MODEL INSTANCE NODE
'Energy T&D for NJ'
[Model-version = TRANSP+DISTR MODEL]
```

Contents = <ptr to contents for model instance node>]

5. SE augments the schema as shown in figure 6-*b*.
6. HE requests the user interface to present to the user the updated network. This is shown in figure 6-*b'*. Note that the buttons `Create Model Data Instance`, `Access Model Data Instance`, `Create Model Instance` and `Access Model Instance` are automatically generated.

If the user now clicks on the *Create Data Instance*, s/he will trigger another chain of events which will result in the screen as shown in figure 6- *c'*. The schema engine assists in resolving the *Model Version Schema* link by presenting the user with the choice between the structured modeling and the graphical representations of the model - the latter one was chosen in this case.

Note that some links are handled entirely by H^+ while others require the DSS application to become active. For example accessing the *model version schema* from a given *model data instance* is entirely handled by H^+ . However, activating the *RUN* link to generate a *results instance* requires a call to a specific solver. The difference lies in that the first activity is part of relationship management and thus is taken care of by H^+ ; while the second activity requires a call to an external application.

7 DISCUSSION

We have argued that there is a need for an integrated approach to managing and providing access to the complex set of interrelated items of information in the application domain of a DSS. To provide this functionality, we proposed that a fourth subsystem, specialized to “relationship management”, be added to the traditional three components of a DSS (the database, user interface and model management systems). The role of RMS is to help DSS application programs create and maintain *structural relationships*, that represent the organization of domain elements, and *process relationships*, that represent the evolution of the DSS. We identified eight requirements for relationship management:

1. High level of abstraction.
2. Rapid authoring.
3. Automated creation and maintenance of relationships.
4. Exploration in the application domain.
5. Coordination of computation.
6. Provide adequate semantics for abstract relationships.

7. Support exploration of the application domain.
8. Support for distributed group work.
9. Version control.

and related these to the capabilities of a number of existing hypertext systems. We proposed an extended form of hypertext engine $H+$ to provide both the full functionality of relationship management and an attractive user interface based on the hypertext paradigm. At the core of $H+$ lie a knowledge oriented hypertext and a schema engine. The DSS and $H+$ communicate using a) schemas, that describe the domain of application, and b) state-change messages, that describe the outcomes of various DSS processes. The role of the schema engine is to facilitate the coordination of activities between the DSS applications (in the background) and the hypertext front-end.

In future research, we will investigate a number related issues:

- A formal treatment of relationship management, including a *calculus of relationships*.
- An implementation of a prototype system for relationship management using existing hypertext software.
- Integrating existing DSSs into new relationship management systems.
- An evaluation of the relationship management approach.

We believe that the concept of relationship management can play an important role in making DSS more functional, more intuitive, and easier to use.

References

- [AKN86] Lynda M. Applegate, Benn R. Konsynski, and Jay F. Nunamaker. Model Management Systems: Design for Decision Support. *Decision Support Systems*, 2:81–91, 1986.
- [BBK88] Hemant Bhargava, Michael Bieber, and Steven O. Kimbrough. Oona, Max and the WYWWYWI Principle: Generalized Hypertext and Model Management in a Symbolic Programming Environment. In Janice I. DeGross and Margarethe H. Olson, editors, *Proceedings of the Ninth ICIS*, pages 179–192, 1988.
- [BHW80] R. H. Bonczek, Clyde Holsapple, and Andrew B. Whinston. Future Directions for Developing Decision Support Systems. *Decision Sciences*, 11(2):616–631, October 1980.
- [BI91a] Michael P. Bieber and Tomás Isakowitz. Bridge Laws in Hypertext, a Logic Modeling Approach. Working paper IS-91-17, Center for Research in Information Systems, New York University, Information systems Department, New York, NY 10003, 1991.
- [BI91b] Michael P. Bieber and Tomás Isakowitz. Valuation Links: Formally Extending the Computational Power of Hypertext. Working paper IS-91-11, Center for Research in Information Systems, New York University, Information systems Department, New York, NY 10003, 1991.
- [Bie91] Michael P. Bieber. Issues in Modeling a *Dynamic* Hypertext Interface. In *Hypertext' 91 Proceedings*, pages 203–218, San Antonio, Texas, December 1991. ACM, ACM Press.
- [BIJS91] P. R. Balasubramanian, Tomás Isakowitz, Hardeep Johar, and Edward A. Stohr. Hyper Model Management Systems. Working paper IS-91-16, Center for Research in Information Systems, New York University, Information systems Department, New York, NY 10003, 1991.
- [BIJS92] P. R. Balasubramanian, Tomás Isakowitz, Hardeep Johar, and Edward A. Stohr. Hyper Model Management Systems. In Jay F. Nunamaker, editor, *Proceedings of the 25th Hawaii International Conference on System Sciences, Volume III*, pages 462–472, Kauai, HI, January 1992. IEEE Computer Society Press.
- [BIKM91] P. R. Balasubramanian, Tomás Isakowitz, Rob Kauffman, and Raghav K. Madhavan. Exploiting Hypertext Valuation Links for Business Decision Making: A Portfolio Management Illustration. In Roy Freedman, editor, *Proceedings of the First International Conference on Artificial Intelligence Applications on Wall Street*, pages 312–318, New York, NY, October 1991. (forthcoming).
- [BK92] Michael P. Bieber and Steven O. Kimbrough. On Generalizing the Concept of Hypertext. *Management Information Systems Quarterly*, 16(1):77–93, March 1992.
- [Bus45] Vannevar Bush. As we may think. *Atlantic Monthly*, pages 101–108, July 1945.
- [BWD⁺89] Robert Blanning, Andrew Whinston, Vasant Dhar, Clyde Holsapple, Mathias Jarke, Stephen Kimbrough, Javier Lerch, and Michael Prietula. Precis of Model Management and the Language of Thought Hypothesis. In Edward A. Stohr, editor, *Proceedings ISDP-89*, 1989.
- [CB89] Jeff Conklin and Michael L. Begeman. gIBIS: A Tool for All Reasons. *Journal of the American Society for Information Science*, 20(3):200–213, 1989.
- [CCC92] Erran Carmel, S. Crawford, and H. Chen. Browsing in Hypertext: A Cognitive Study. *IEEE Transactions on Systems, Man and Cybernetics*, Forthcoming, 1992.
- [Che76] P. P. Chen. The Entity Relationship Model: Toward a Unified View of Data. *ACM Transactions on Database Systems*, 1(1):9–36, 1976.
- [CHW92] A. Chang, Clyde W. Holsapple, and Andrew B. Whinston. The Hyperknowledge Environment of Model Management Systems. In Edward A. Stohr and Ben R. Konsynski, editors, *Information and Decision Processes*. IEEE Computer Society Press, 1992.
- [Con87] Jeff Conklin. Hypertext: An Introduction and Survey. *IEEE Computer*, 20(9):17–41, September 1987.
- [DS87] Norman Delisle and Mayer Schwartz. Contexts-A Partitioning Concept For Hypertext. *ACM Transactions on Office Information Systems*, 5(2):168–186, 1987.

- [Eng83] Douglas C. Engelbart. *Vistas in Information Handling*. Spartan Books, London, 1983.
- [Eng88] Douglas C. Engelbart. The Augmentation System Framework. In S. Ambron and K. Hooper, editors, *Interactive Multimedia*. Microsoft Press, Redmond, Washington, 1988.
- [Fri88] Mark E. Frisse. Searching For Information In A Hypertext Medical Handbook. *Communications of the ACM*, 31(7):880–886, July 1988.
- [GB87] Ira P. Goldstein and Daniel G. Bobrow. A Layered Approach to Software Design. In D. Barstow, H. Schrobe, and E. Sandewall, editors, *Interactive Programming Environments*, pages 387–413. McGraw-Hill, New York, 1987.
- [Geo87] Arthur M. Geoffrion. An Introduction To Structured Modeling. *Management Science*, 33(5):547–588, May 1987.
- [GPS91] Franca Garzotto, Paolo Paolini, and Daniel Schwabe. HDM - A Model for the Design of Hypertext Applications. In *Hypertext'91 Proceedings*, pages 313–328, San Antonio, Texas, December 1991. ACM, ACM Press.
- [GUI87] GUIDE. *Guide User's Manual*. Owl International Inc., 1428 NE 21 St., Bellevue, WA 98007, (206) 747-3203, 1987.
- [Hal88] Frank G. Halasz. Reflections on Notecards: Seven Issues for the Next Generation of Hypermedia Systems. *Communications of the ACM*, 31(7):836–852, 1988.
- [Hal91] Frank G. Halasz. Seven Issues: Revisited. Hypertext'91 Keynote Talk, Hypertext'91 Conference, December 1991.
- [HMT87] Frank G. Halasz, T.P. Moran, and Randy H. Trigg. Notecards in a Nutshell. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*, 1987.
- [HS66] J. Hartmanis and R. E. Stearns. *Algebraic Structure Theory of Sequential Machines*. Prentice Hall, 1966.
- [HS90] Frank G. Halasz and Meyer Schwartz. The Dexter Hypertext Reference Model. In Judi Moline, Dan Beningni, and Jean Baronas, editors, *Proceedings of the Hypertext Standardization Workshop*, pages 95–133, Gaithersburg, MD 20899, March 1990. National Institute of Standards and Technology, NIST special publication 500-178.
- [Hub84] George P. Huber. The Nature And Design Of Post-Industrial Organizations. *Management Science*, 30(8):928–951, August 1984.
- [Isa92] Tomás Isakowitz. MALUAR - A Computational Hypertext Environment. Working paper, Center for Research in Information Systems, New York University, Information systems Department, New York, NY 10003, 1992.
- [Kee80] Peter G. W. Keen. Decision Support Systems: Translating Useful Models into Usable Technologies. *Sloan Management Review*, 21(3):33–44, Spring 1980.
- [KPBB90] Steven O. Kimbrough, Clark W. Pritchett, Michael P. Bieber, and Hemant K. Bhargava. The Coast Guard's KSS Project. *Interfaces*, 20(6):5–16, November/December 1990.
- [KS91] Harmann Kaindl and Mikael Snaprud. Hypertext and Structured Object Representation: A Unifying View. In *Hypertext'91 Proceedings*, pages 354–358, San Antonio, Texas, December 1991. ACM, ACM Press.
- [KS92] Ben Konsynski and Edward A. Stohr. Decision Processes: An Organizational View. In Edward A. Stohr and Ben Konsynski, editors, *Information and Decision Processes*. IEEE Computer Society Press, 1992.
- [Lan90] Danny B. Lange. A Formal Model of Hypertext. In Judi Moline, Dan Beningni, and Jean Baronas, editors, *Proceedings of the Hypertext Standardization Workshop*, pages 145–166, Gaithersburg, MD 20899, March 1990. National Institute of Standards and Technology, NIST special publication 500-178.

- [MBD⁺90] Raymond J. McCall, Patrick R. Bennett, Peter S. D'Oronzio, Jonathan L. Ostwald, Frank M. Shipman III, and Nathan F. Wallace. PHIDIAS: Integrating CAD Graphics into Dynamic Hypertext. In A. Rizk, N. Streitz, and J. André, editors, *Proceedings of the European Conference on Hypertext*, pages 212–223, France, November 1990. INRIA, Cambridge University Press.
- [MHRJ91] Catherine C. Marshall, Frank G. Halasz, Russell A. Rogers, and William C. Janssen Jr. Aquanet: a hypertext tool to hold your knowledge in place. In *Hypertext'91 Proceedings*, pages 261–275, San Antonio, Texas, December 1991. ACM, ACM Press.
- [Min90] R.P. Minch. Applications and Research Areas for Hypertext in Decision Support Systems. *Journal of Management Information Systems*, 6(3):119–138, Winter 1989-90.
- [MMS89] P. Ma, F.H. Murphy, and E.A. Stohr. A graphics interface for linear programming. *CACM*, 32(8):996–1012, May 1989.
- [MO92] Nenad Marovac and Larry Osburn. Hypernet. Working paper, Computer Science Department, San Diego University, 1992.
- [MS88] G. Marchionini and Ben Shneiderman. Finding Facts vs. Browsing Knowledge in Hypertext Systems. *IEEE Computer*, pages 70–80, January 1988.
- [Nel80] Theodor H. Nelson. Replacing the Printed Word: A Complete Literary System. In S. H. Lavington, editor, *IFIP Proceedings*, pages 1013–1023. North Holland, 1980.
- [Nie90a] Jakob Nielsen. *HyperText & HyperMedia*. Academic Press, 1990.
- [Nie90b] Jakob Nielsen. Through Hypertext. *Communications of the ACM*, 33(3):297–310, March 1990.
- [NN91] Jocelyne Nanard and Marc Nanard. Using Structured Types to incorporate Knowledge in Hypertext. In *Hypertext'91 Proceedings*, pages 329–342, San Antonio, Texas, December 1991. ACM, ACM Press.
- [RT88] Darrell R. Raymond and Frank WM. Tompa. Hypertext And The Oxford English Dictionary. *Communications of the ACM*, 31(7):871–879, July 1988.
- [SCGL91] Karen Smith-Catlin, L. Nancy Garrett, and Julie A. Launhardt. Hypermedia Templates: An Author's Tool. In *Hypertext'91 Proceedings*, pages 147–160, San Antonio, Texas, December 1991. ACM, ACM Press.
- [SF89] P. David Stotts and Richard Furuta. Petri net based Hypertext: Document Structure with Browsing Semantics. *ACM Transactions on Information Systems*, 7(1), January 1989.
- [SL89] John L. Schnase and John J. Leggett. Computational Hypertext in Biological Modelling. In *Hypertext '89 Conference Proceedings*, pages 181–197, Hypertext Research Lab, Dept. of Computer Science, Texas A&M University, College Station, TX 77843, November 1989.
- [Spr89] Ralph H. Sprague. A Framework for the Development of Decision Support Systems. In Ralph H. Sprague and Hugh J. Watson, editors, *Decision Support systems, Putting Theory into Practice*, chapter 1, pages 9–35. Prentice Hall, second edition, 1989.
- [UY89] Kenneth Utting and Nicole Yankelovich. Context and Orientation in Hypermedia Networks. *ACM Transactions on Information Systems*, 7(1):58–84, January 1989.
- [YMvD85] Nicole Yankelovich, Norman Meyrowitz, and Andries van Dam. Reading and Writing the Electronic Book. *IEEE Computer*, October, 1985.

