REFLECTIONS ON NOTECARDS: SEVEN ISSUES FOR THE NEXT GENERATION OF HYPERMEDIA SYSTEMS

NoteCards, developed by a team at Xerox PARC, was designed to support the task of transforming a chaotic collection of unrelated thoughts into an integrated, orderly interpretation of ideas and their interconnections. This article presents NoteCards as a foil against which to explore some of the major limitations of the current generation of hypermedia systems, and characterizes the issues that must be addressed in designing the next generation systems.

FRANK G. HALASZ

Hypermedia is a style of building systems for information *representation* and *management* around a network of multi-media nodes connected together by typed links. Such systems have recently become quite popular due to their potential for aiding in the organization and manipulation of irregularly structured information in applications ranging from legal research to software engineering. Moreover, there is something alluring about navigating through a hypermedia network following links from node to node until you find some information of interest. But as the current crop of hypermedia systems moves into more widespread use, the limitations of the hypermedia concept are becoming increasingly apparent. The simple node and link model is just not rich and complete enough to support the information representation, management, and presentation tasks required by many applications. This article presents the NoteCards system as a foil against which to examine some of these limitations in the current generation of hypermedia systems. By examining the major weaknesses in the design of NoteCards, the article characterizes seven critical issues that need to be resolved in designing the next generation of hypermedia.

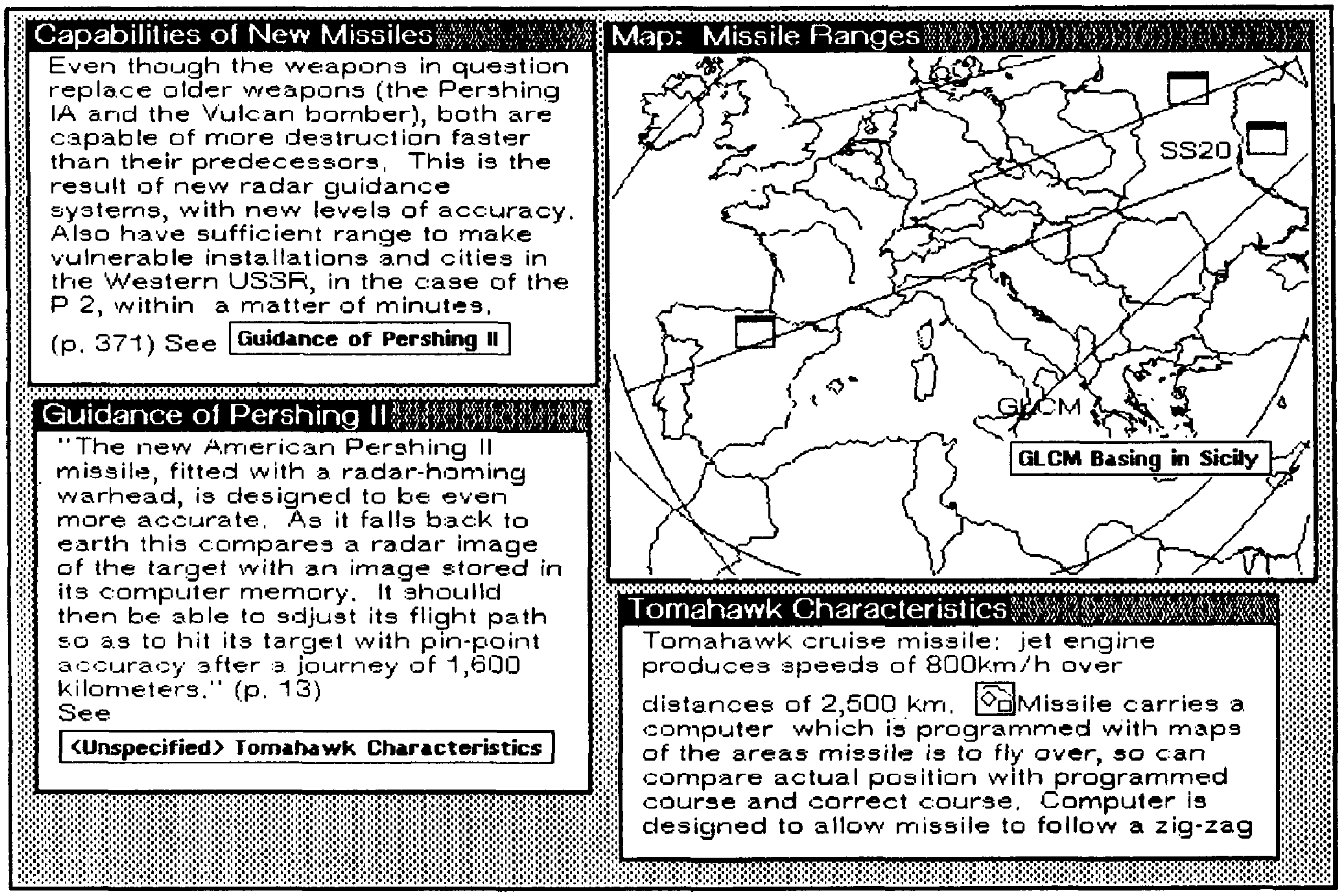
NOTECARDS IN BRIEF

NoteCards is a general hypermedia environment that is fairly typical of the generation of workstation-based systems that is currently moving from the research lab into widespread use (e.g., Intermedia [35] and Neptune [9]). NoteCards was designed to help people work with ideas. Its intended users are authors, researchers, designers, and other intellectual laborers engaged in analyzing information, constructing models, formulating arguments, designing artifacts, and generally processing ideas. The system provides the user with a network of electronic notecards interconnected by typed links. This network serves as a medium in which the user can represent collections of related ideas. It also functions as a structure for organizing, storing, and retrieving information. The system includes facilities for displaying, modifying, manipulating, and navigating through this network. NoteCards was developed by Randall Trigg, Thomas Moran and the present author at Xerox PARC. A more detailed discussion of the system, its design goals, and our experiences with its use can be found in [26].

Four Basic Constructs

NoteCards is implemented within the Xerox Lisp programming environment and is designed around two primitive constructs, *notecards* and *links*. In the basic system, these two primitives are augmented by two specialized types of cards, *browsers* and *fileboxes*, that help the user to manage large networks of cards and links.

- *Notecards.* A notecard is an electronic generalization of the 3×5 paper notecard. Each notecard contains an arbitrary amount of some editable substance such as a piece of text, a structured drawing, or a bitmap image. Each card also has a title. On the screen, cards are displayed using standard Xerox Lisp windows as shown in Figure 1. Every notecard can be edited, that is, retrieved from the database and displayed on the screen in an editor window. There are various types of notecards, differentiated (in part) by the nature of the substance (text or graphics) that they contain. In addition to a set of standard card types, NoteCards includes a facility for adding new card types, ranging from small modifications to existing card types (e.g., text-based forms) to cards based on entirely different substances (e.g., animation cards).
- *Links.* Links are used to interconnect individual notecards into networks or structures of related cards. Each link is a typed, directional connection between a source card and a destination card. The type of a link is a user-chosen label specifying the nature of the relationship being represented. Links are anchored by an icon at a specific location in the substance of their source card, but are anchored to their destination card as a whole. Clicking the mouse in the link icon traverses the link, that is, retrieves the destination card and displays it on the screen. In Figure 1, each of the two cards contains two embedded link icons.
- *Browsers.* A browser is a notecard that contains a structural diagram of a network of notecards. Figure 2 shows a Browser card for a network composed of 8 cards and 8 links. The cards from this network are represented in the browser by their title displayed in a box. The links in the network are represented by edges between the boxed titles. Different dashing styles distinguish different types of links. The diagrams in Browser cards are computed for the user by the system. Once created, browsers function like


Capabilities of New Missiles

Even though the weapons in question replace older weapons (the Pershing IA and the Vulcan bomber), both are capable of more destruction faster than their predecessors. This is the result of new radar guidance systems, with new levels of accuracy. Also have sufficient range to make vulnerable installations and cities in the Western USSR, in the case of the P 2, within a matter of minutes.
(p. 371) See **Guidance of Pershing II**

Guidance of Pershing II

"The new American Pershing II missile, fitted with a radar-homing warhead, is designed to be even more accurate. As it falls back to earth this compares a radar image of the target with an image stored in its computer memory. It shoulld then be able to sdjust its flight path so as to hit its target with pin-point accuracy after a journey of 1,600 kilometers." (p. 13)
See
<Unspecified> Tomahawk Characteristics

Map: Missile Ranges

Tomahawk Characteristics

Tomahawk cruise missile; jet engine produces speeds of 800km/h over distances of 2,500 km. Missile carries a computer which is programmed with maps of the areas missile is to fly over, so can compare actual position with programmed course and correct course. Computer is designed to allow missile to follow a zig-zag

FIGURE 1. Example NoteCards with Embedded Link Icons

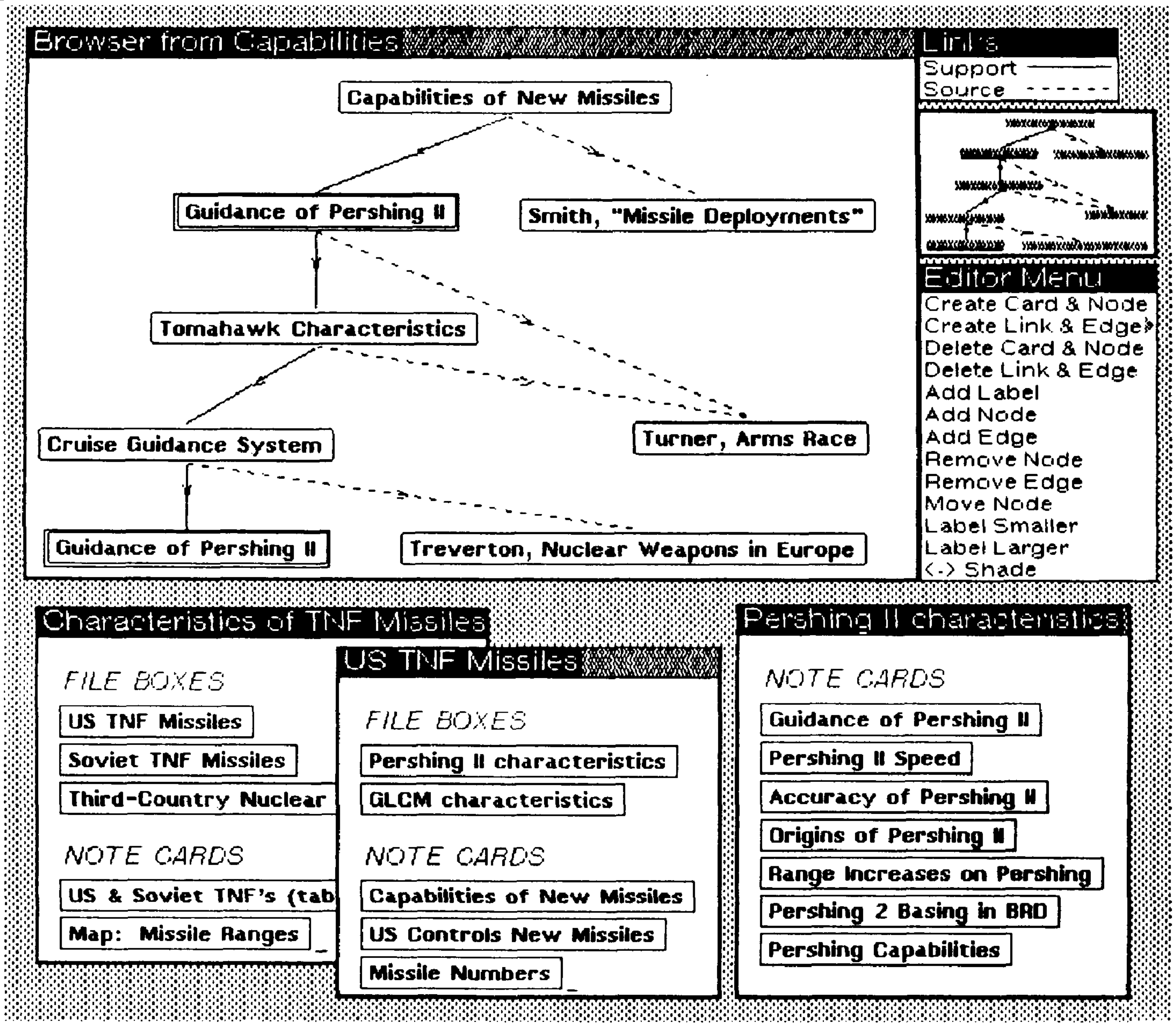


FIGURE 2. Example Browser Card (top) and Filebox Cards

standard notecards. The boxed titles in the browser are in fact icons for traversable links between the browser and the referenced card. Browsers support two levels of editing. First, the user can edit the underlying structure of a network of notecards by carrying out operations on the nodes and edges in the browser. Second, the user can add and delete nodes and edges in the browser diagram without making corresponding changes to the underlying NoteCards structures.

- *Fileboxes.* Fileboxes are specialized cards that can be used to organize or categorize large collections of notecards. They were designed to help users manage large networks of interlinked notecards by encouraging them to use an additional hierarchical category structure for storage and retrieval purposes. A filebox is a card in which other cards, including other fileboxes, can be filed. NoteCards requires that every notecard (including fileboxes) must be filed in one or more fileboxes. Figure 2 shows three fileboxes in addition to the browser.

Accessing Information in NoteCards

Navigation, whereby the user moves through the network by following links from card to card, is the primary means for accessing information in NoteCards. Alternatively, the user can create an overview browser for some subnetwork and traverse the links from the browser to the referenced cards. NoteCards also provides a limited search facility that can locate all cards matching some user-supplied specification (a particular string in the card's title or text, for example).

Tailorability

NoteCards is fully integrated into the Xerox Lisp programming environment. It includes a widely used programmer's interface with over 100 Lisp functions that allow the user to create new types of cards, develop programs that monitor or process a network, integrate Lisp programs (e.g., an animation editor) into the NoteCards environment, and/or integrate NoteCards into another Lisp-based environment (e.g., an expert system). The system also includes a large set of parameters that users can set to tune the exact behavior of the system (e.g., how links are displayed or the default size of notecards). For a more extensive discussion of extensibility and tailorability in NoteCards see [51].

NoteCards in Use

From its inception, the design and development of NoteCards has been driven by the needs of its user community. Currently there are over 70 registered users within Xerox. There are, in addition, an undetermined number of users at various university, government, and industrial sites outside Xerox. This user community has provided invaluable feedback on the strengths and weaknesses of NoteCards as applied to a variety of tasks including document authoring, legal argumentation, development of instructional materials, design of copier parts, and competitive market analysis. Perhaps the most common use of NoteCards is as a database for storing miscellaneous personal information.

A typical example of how the system can be used to support idea structuring and generic authoring is the network created by a history graduate student who used the system to research and write a 25 page paper. Figure 3 shows a browser of the filebox hierarchy created during this project. The author made a habit of keeping this browser on his screen at all times as a way of speeding up the process of filing and accessing cards. This hierarchy was made up of 40 fileboxes and contained 268 (nonfilebox) cards.

The cards in Figure 1 are taken from this hierarchy. In general, cards stored in the hierarchy contain a short (average of about 100 words) quote or paraphrase taken from an article or book. About half of the cards have links embedded in their substance. As a rule, these were "See" or "Unspecified" links and were placed at the end of the card's text preceded by the word "See." There are also a few dozen inter-card links of other types. Further description of this network and the process involved in using it to author a paper can be found in [36] and [26].

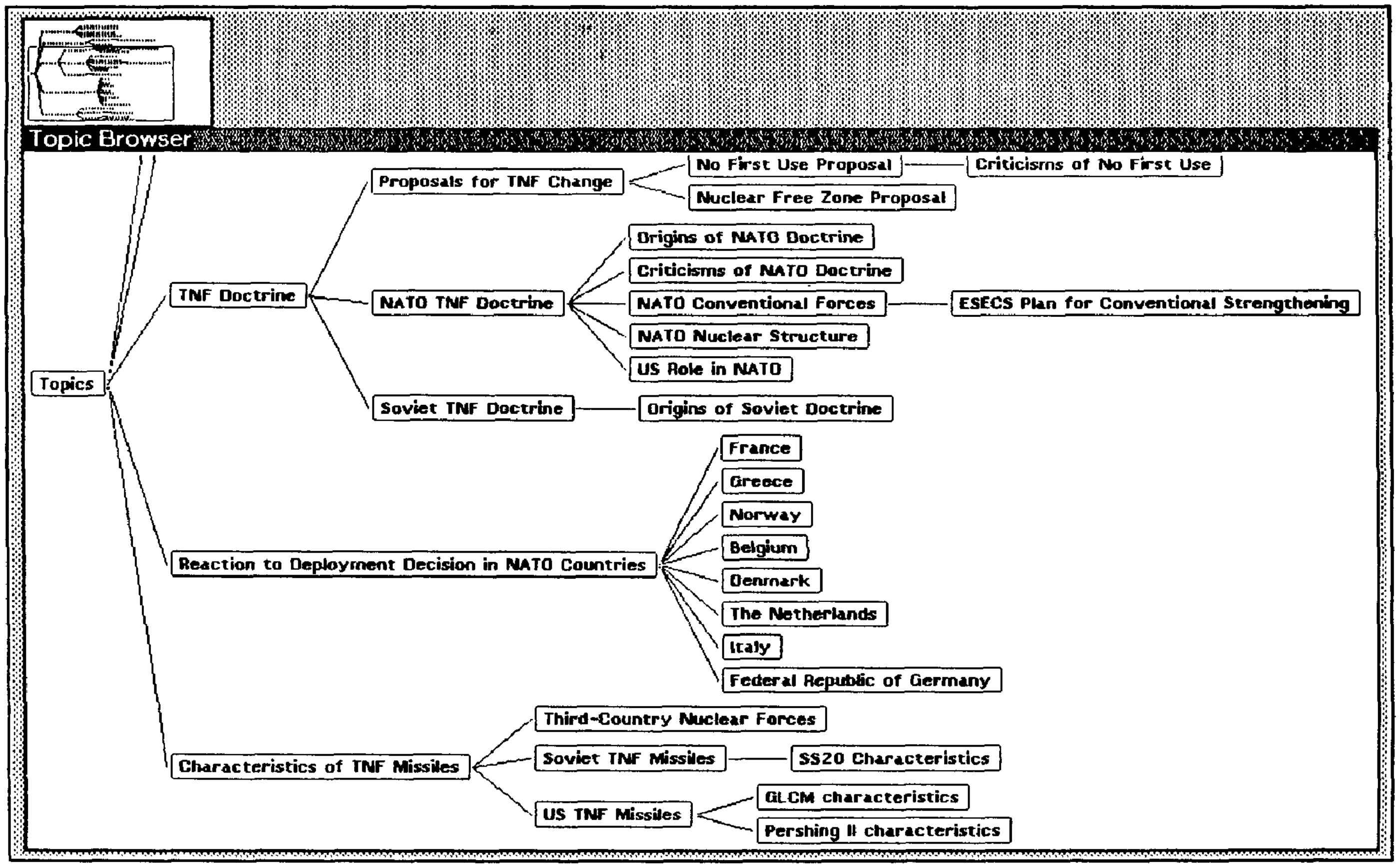


FIGURE 3. Browser of the Filebox Hierarchy from the Public-Policy (NATO-missiles) Notefile

NOTECARDS' NICHE IN THE SPACE OF HYPERMEDIA SYSTEMS

NoteCards is a fairly typical *second* generation hypermedia system. The first generation included systems such as NLS/Augment [11], [12], FRESS [54], and ZOG [33], [40]. These systems were all originally mainframe-based, focused primarily on text nodes, and used display technologies with little or no graphics capabilities. All of these first generation systems, however, included at least some support for medium to large teams of workers sharing a common hypermedia network. An overview of these systems and many others can be found in [6].

The second and current generation of hypermedia began in the early 1980s with the emergence of various workstation-based, research-oriented systems including NoteCards, Neptune [9], [10] and Intermedia [21], [35]. In addition, KMS [1] is a commercial workstation-based system that grew out of the first-generation ZOG system. These second generation systems are remarkably similar in concept to the first generation systems. However, the workstation technology which underlies them allows for much more advanced user interfaces. In particular, all of these systems support graphics and even animation nodes as well as fully-formatted text nodes. They also make heavy use of graphical overviews of the network structure to aid navigational access. It is interesting to note that these systems are generally designed for single users or small workgroups and hence do not support collaborative work to the same degree as the earlier systems.

These workstation-based, research-oriented systems, were followed a few years later by a number of products and prototype systems running on personal computers. The PC-based systems, which include Guide [5], [25] and Hyperties [43], [44], are more limited in scope and functionality than the workstation-based systems but have many of the same features. Unfortunately, one critical feature not included in these systems is a graphical overview of the network structure to aid the user when navigating in and manipulating the hypermedia.

Table I summarizes the architectural features of the average (and fictional) current generation hypermedia system. This system includes typed nodes connected by labeled, bidirectional links. The nodes are implemented using an extensible type hierarchy, with at least the text and graphics node types being provided with the basic system. The links are labeled but not typed, and are anchored using icons within the contents of both the source and destination nodes. In addition to the standard node and link networks, there is some special support for hierarchical organizations. Information access and structure editing is accomplished using "browsers" containing a graphical map of the network structure. Query-based access is possible, but it is very slow and limited to simple string or keyword matching. The network is stored on a shared relational database or in standard files, but there is no versioning of the stored information. Support for simultaneous multi-user access to the stored information is limited. Finally, there is a programmer's interface that can be used to extend or tailor the system.

TABLE I. Architectural Features of an Average Current Generation Hypermedia System

Feature	Description
Nodes:	Typed (text, graphics, . . .), implemented using a type hierarchy Nodes cannot contain other nodes
Links:	Binary, bidirectional Labeled but not typed Anchors can be whole nodes or points/regions within the node
Overviews:	Browsers containing node/link diagrams of the network Can edit network via browser
Hierarchies:	Special support for hierarchical networks
User Interface:	Multiple windows; mouse/menu driven
Extensibility:	Programmer's interface
Search/query:	Slow, full-text string match
Distribution:	Single-user or multi-user central server with limited concurrency control
Versioning:	None
Storage:	Standard files or relational DBMS

Despite this commonality in basic architecture, the current generation includes systems that are very diverse in their nature and functionality. This diversity can be characterized by partitioning the space of hypermedia systems along three *fundamental dimensions: scope, browsing vs. authoring,* and *target task domain.* Since it is serving as an example against which to assess the entire generation, it is important to understand where NoteCards lies along these three dimensions.

- *Scope.* Hypermedia has been proposed as *the* mechanism for storing and distributing the world's entire literary output [39], as a common information space for teams of programmers on large software projects [19], and as a tool for individuals and small work groups engaged in authoring and idea processing [26]. Although all of these proposals share the notion that information should be organized into networks of nodes and links, they differ radically in scale, e.g., in the sizes of their expected information bases and user populations. This extreme variation in scale implies there will be differences throughout these systems, ranging from underlying storage mechanisms through the user interface to conventions for their use.
- *Browsing versus Authoring.* In systems designed primarily for browsing, the hypermedia network is carefully created by a relatively small number of specialized authors in order to provide an information space to be explored by a large number of more or less casual users. These browsing systems are generally characterized by relatively well-developed tools for information presentation and exploratory browsing. Tools for creating and modifying the network tend to be less evolved. Hypermedia instructional delivery

environments [21] and interactive museum exhibits [43] are examples of such browsing-oriented systems. In systems designed primarily for authoring, the hypermedia network serves as an information structure that users create and continuously modify as part of their ongoing task activities. Hypermedia systems for idea processing [26], document authoring [45], and software development [9] are primary examples. In such systems, the tools for creation and modification of the network are well-developed. Tools for easy browsing and sophisticated information display are present, but tend to be less evolved.

- *Target task domain.* Many hypermedia systems have been designed to support a specific task. For example, WE [45] is an environment designed specifically to support the professional writer. Similarly, the Document Examiner [53] is designed specifically to support the on-line presentation of technical documentation. Other hypermedia systems provide general hypermedia facilities to be used in a variety of applications. Even these generic systems, however, are usually designed with a target task domain in mind. The features and capabilities emphasized in the system often reflect the requirements of this target. Contrast, for example, Intermedia [21] and Neptune [9]. Both are general hypermedia systems. But Neptune was designed to support software engineering and thus emphasizes versioning [10] and node/link attributes. In contrast, Intermedia was designed for multiuser interactive educational applications and thus emphasizes novel interactive displays [20] and annotation facilities.

NoteCards is designed for use by individuals or small work groups. In this respect, NoteCards is similar to PC-based systems like Guide [25] and to single-user workstation systems like WE [45]. Conversely, NoteCards differs significantly along this dimension from global systems like Xanadu [39], as well as systems designed to support larger groups such as ZOG [40] and NLS/Augment [11]. Although its original design places less emphasis on multi-user access, NoteCards is very similar in scope to Neptune [9], Intermedia [35], [21] and Textnet [49].

NoteCards is first and foremost an authoring system designed to provide its users with facilities for creating and modifying hypermedia structures. In this respect, NoteCards is similar to many of the aforementioned systems (Augment, Guide, Intermedia, Neptune, WE) and different from on-line presentation systems such as the Document Examiner [53], Hyperties [43], [44] and interfaces to CD/ROM databases [29]. ZOG has a slightly different flavor, being simultaneously browser- and author-oriented.

NoteCards is a general purpose hypermedia system, but it was originally designed to be used as a tool for idea processing and authoring in a research environment. Its original goals were very similar to those of NLS/Augment, although the actual implementations of the two systems are (on the surface) quite different. Systems such as WE and Guide were also designed to support authoring tasks. Intermedia (education) and Neptune (software engineering) were designed with very different application domains in mind, although both systems were designed in part to support document authoring tasks.

Overall, NoteCards is most similar to Intermedia and Neptune despite the differences in their target application domains. The three are very similar in scope and in the type of facilities they provide. This similarity is reinforced by several factors including a common emphasis on extensibility, similar underlying platforms (i.e., workstations), and a contemporaneous development schedule. Although the present article specifically discusses NoteCards, most of the issues raised are relevant to these two systems and, to a lesser extent, to a large majority of the other current generation systems.

SEVEN ISSUES FOR THE NEXT GENERATION OF HYPERMEDIA SYSTEMS

In the three years since its first release, NoteCards has been observed in use and misuse in a wide range of situations and applications. These observations have provided significant insight into the system's particular strengths as well as its weak points. A brief but balanced assessment of the system is contained in [26]. For expository reasons the present article focuses only on NoteCards weak points, i.e., on the ways in which the system falls short in meeting the needs and preferences of its users.

Many of the NoteCards' problems are specific to its current implementation and could be corrected by limited redesign or reimplementation of the existing system. However, several of its problems reflect fundamental weaknesses in the hypermedia model around which it is built. It is precisely these fundamental weaknesses and the mechanisms for their correction that should form the basis for designing the next generation of hypermedia. The following sections describe seven fundamental limitations evident in NoteCards.

Issue 1: Search and Query in a Hypermedia Network

In some sense, hypermedia is navigational access. The ability to browse around a network by following the links from node to node is a defining feature of hypermedia. It is precisely this ability that makes hypermedia a powerful tool for managing loosely structured information. The NoteCards experience suggests, however, that navigational access by itself is not sufficient. Effective access to information stored in a hypermedia network requires query-based access to complement navigation.

NoteCards is fairly typical in the facilities it provides for navigational access. To retrieve information stored in a network, the typical NoteCards user brings a card onto the screen, examines its contents and links, and then traverses the link that is most likely to move closer to the target information. Fileboxes support such localized link traversal by providing a hierarchical structure in which information is located by recursive descent through an increasingly specific category structure. Such localized link following is augmented by global maps of the network contained in browsers. These maps allow the user to visually scan for and then directly move to areas of the network in which the target information is likely to be found.

Navigational access to information has been adequate, and occasionally even ideal, for a large number of NoteCards applications. These applications can be divided into three basic classes. First, the navigational access has proven sufficient for the small authoring, note taking, and informal knowledge representation tasks that NoteCards was originally designed to support. In these tasks, an individual or a small (2 to 3 person) workgroup is creating and intensively using a relatively small network (50 to 250 cards). Because the network is small and familiar, users have little problem locating information.

A second class of navigational applications are the display-oriented representation tasks in which the network is centered around a single display, usually a browser, used to represent a structure being designed or studied. The goal of these tasks is to create and manipulate this display. In some sense, the network is secondary to the display and is used only to create the structure to be displayed and to hide unimportant details. In these tasks, information access occurs through the central display, with little direct card-to-card navigation. An example of such a network is described in [52].

The third class of navigationally-oriented applications is on-line interactive presentations. In these applications, the network's author often includes in each card navigational instructions to be used by readers of the network. Such guided on-line presentations are discussed in [50]. If no such navigational instructions are included, then the network is generally designed to be explored by the user in a nondirected manner.

In contrast to these navigationally-oriented applications, there are a variety of applications for which NoteCards' reliance on navigational access is problematic. These applications are generally characterized by large, unfamiliar, heterogeneously structured networks. Even in a 500 node single-user network, navigational access can be difficult as the network changes and its structure becomes heterogeneous. In these cases, navigational access is problematic because users tend to get lost while wandering around in the network looking for some target information. Often these users can describe exactly what information they are looking for, but simply cannot find it in the network.

An incremental solution to the navigational problems encountered in NoteCards would be to improve and augment the existing navigation tools. For example, browsers could be made significantly more effective by applying techniques such as fish-eye views [17] and graph flyovers [13]. In addition, new tools such as a voting scheme similar to Synview [31] could be implemented in NoteCards. Although these changes would alleviate some of the navigation problems, they would not eliminate them entirely.

A more fundamental solution is to augment navigation by a query-based access mechanism. With such a mechanism, the user could formulate a query encapsulating a description of the target information and then rely on the system to locate the information in the network. As described above, NoteCards already provides a very limited query/search facility. Unfortunately, this search mechanism is too simple and too poorly implemented to be very useful. For NoteCards to be useful in managing large heterogeneous networks, search and query needs to be elevated to a primary access mechanism on par with navigation.

There are two broad classes of query/search mechanisms needed in a hypermedia system: *content search* and *structure search.* In content search, all nodes and links in the network are considered as independent entities and are examined individually for a match to the given query. For example, all the nodes containing the string "hyper*" would be a content query. Content search is standard information retrieval applied to a hypermedia information base. Basic techniques for content searches are well known (see [42]). In addition, there are many innovative approaches that could be fruitfully explored in a hypermedia environment. For example, rule-based retrieval schemes (e.g., RUBRIC [34]), connectionist approaches (e.g., [38]), simple statistical techniques (e.g., *n*-gram indexing [32]), and specialized search hardware (e.g., [28]) are all interesting candidates for inclusion in a hypermedia content search engine.

Content search ignores the structure of a hypermedia network. In contrast, structure search specifically examines the hypermedia structure for subnetworks that match a given pattern. For example, the following is a simple structure query: all subnetworks containing two nodes connected by a supports link, where the destination node contains the word "hypertext." This query contains a description of node content (i.e., contains the word "hypertext"). But it also contains a structural description of a subnetwork (i.e., two nodes connected by a "supports" link). A more complicated structure query, involving an indefinite sequence of links, would be something like: a circular structure containing a node that is indirectly linked to itself via an unbroken sequence of "supports" links. This query could be used, for example, to find circular arguments.

The development of a structure search mechanism involves two interrelated subtasks. The first is to design a query language geared toward describing hypermedia network structures. One approach to this would be to develop an analog to regular expressions that would encompass arbitrary network (non-linear) patterns. This pattern language would need to include the standard regular expression operators such as associative grouping, concatenation, alternates, closure, and negation. In addition, the language would need some way to express forking, e.g., an operator with *N* regular expression arguments which states that the next entity in the network being matched must be *N*, not necessarily distinct, structures each of which matches one of the argument expressions. The graphical query language presented in [7] provides a limited start to defining a network query language of this nature.

An important consideration in designing this query language is the need for a simple interface that is accessible to the typical hypermedia user, who is unlikely to be facile with the intricacies of pattern languages. One approach to designing this interface is a network analog of Query-by-Example [55] in which the user could graphically depict the target pattern. Figure 4 shows a simple example that is meant to express the query: find all subnetworks containing an "Issue" node linked to at least two "Position" nodes, each of which has no outgoing links. The design of a graphical interface of this sort is not easy given that operators such as negation (absence of a structure) and closure need to be included in the representation.

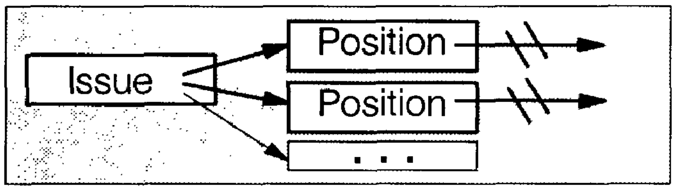


FIGURE 4. Example of a Graphical Expression of the Query: Find All Subnetworks Containing an "Issue" Node Linked to at Least Two "Position" Nodes, Each of Which Has No Outgoing Links

The second major subtask in developing a structure search facility is the implementation of a search engine capable of satisfying the queries expressible in the new language. It is unlikely that efficient search engines can be developed to implement the full pattern matching capabilities suggested by the foregoing discussion. Thus, one of the critical issues for the next generation of hypermedia systems is to define a restricted pattern matching capability that could be easily implemented and yet would satisfy a significant subset of the pattern matching requirements of the average hypermedia user.

Once developed, search and query facilities will be critical components in several aspects of hypermedia systems beyond their basic task of locating information. In particular, queries can be used as a filtering mechanism in the hypermedia interface. In this case, users will specify a query in order to describe the information of interest to them. The interface would then display only those aspects of the network that matched this query, thereby filtering out irrelevant information. The NoteCards browser currently operates in this manner, but only with respect to a very limited set of structure queries. A full-blown query mechanism would allow much more interesting browsers to be constructed. More importantly, the search/query mechanism could be linked much deeper into the interface providing for a pervasive information filtering mechanism that is currently absent in NoteCards. Search and query is also a critical component of the virtual structures mechanism described under Issue 3.

Issue 2: Composites—Augmenting the Basic Node and Link Model

In accordance with the basic hypermedia model, there are only two primitive constructs in NoteCards: cards and links. All other mechanisms in the system, including fileboxes and browsers, are built up from these two constructs. Although this design has been surprisingly successful, experience suggests that it is insufficient. In particular, the basic hypermedia model lacks a composition mechanism, i.e., a way of representing and dealing with groups of nodes and links as unique entities separate from their components.

Figure 5 shows a typical use for composite structures in NoteCards. The figure contains a schematic diagram of a CaseCluster card developed to facilitate the encoding of legal cases into a NoteCards network. When a CaseCluster is created, ten individual cards are created and linked together as shown in the figure. Taken together these ten cards are intended to be a network-based form that, when filled in and expanded, encodes the analysis of a single legal case. Eight of the ten cards are used to contain information about various components of the case. The ninth card is a Browser card showing some important structural relations among a few of the eight component cards. The tenth card contains no content information; it is simply a head card used to gather the other nine cards into an ad hoc grouping. This head card serves an important conceptual function. It is used to represent the legal case (the CaseCluster) as a whole. For example, a user wishing to create a link to a given legal case creates a link to the head card of its CaseCluster. Similarly, properties that are true of the case as a whole are attached to the head card, rather than to each of the component cards.

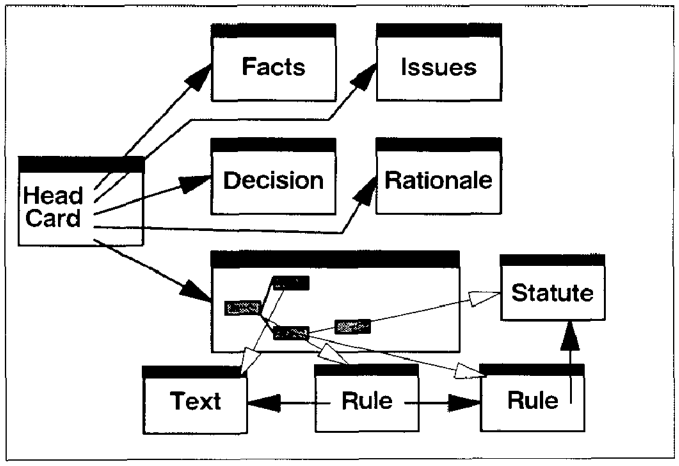


FIGURE 5. CaseCluster "Card" Developed to Facilitate Encoding of Legal Cases in NoteCards

The use of such head cards is fairly common in NoteCards. They are an attempt to utilize existing mechanisms to implement the composition of individual cards and links into higher-level entities. Unfortunately, the head cards are severely limited as a composition mechanism. For example, CaseClusters cannot be treated as single (compound) entities in a browser card. Browsers can display just the head card, in which case the links in and out of the component cards are not displayed. Alternatively, browsers can show all of the cards in a CaseCluster in the normal manner, in which case there is no demarcation that the ten cards in the CaseCluster form a conceptual grouping. In short, browsers are not able to display CaseClusters appropriately, i.e., as single entities that inherit all of the links in and out of their component cards. This limitation and numerous others of a similar nature arise because the system does not understand the user-imposed semantics of the head card. The user intends the head card structure to represent a composition, but the system has no understanding of compositions.

A similar set of problems arises with NoteCards' notion of fileboxes. The filebox concept was designed (in part) to provide some of the characteristics of a composition mechanism with the intent of encouraging hierarchical organizational structures. But the concept is flawed because it fails to take into account the differences between *reference* relations and *inclusion* relations. In particular, an inclusion relation implies a part/whole relationship in which characteristics of and operations on the whole will affect the parts as well. Reference implies a much looser relationship in which the participating entities allude to each other but remain essentially independent. Fileboxes are implemented using standard links, i.e., using reference relationships. But the interface and documentation encourage the user to think of "filing in a filebox" as an inclusion relation. Unlike CaseClusters, this "filing as inclusion" semantics is supported by the system. Unfortunately, this support is only partial. The result is considerable confusion among NoteCards users about the proper use of fileboxes.

An excellent example of this confusion can be seen in the task of writing an organized document (e.g., a technical report) in NoteCards (see [51]). In this task, users typically put the text for each subsection and for each figure into a separate card. All of the cards for a single section are then filed in a filebox. These section fileboxes are filed in the appropriate chapter fileboxes, which in turn are filed in a single filebox representing the document. This scheme is workable. Using the NoteCards document compiler, the user can linearize the network into a single document card containing all of the text and graphics for the document. This document can then be manipulated as a single entity. There is a problem, however, in that the document card is a *separate entity from the source cards stored in the document's filebox hierarchy.* It contains only copies of the text/graphics from these source cards. Changes made to the text/graphics in the document card are not automatically reflected in the corresponding source card.

Another problem with this arrangement is that the user can see the entire document at only one level. Despite the elaborate filebox hierarchy, there is no way to zoom in and out of the document structure, examining its contents at different levels of detail. This capability is commonly found in outline processors and is a critical component in many writing and information organization tasks. As a result, a number of writers have abandoned NoteCards in favor of outline processors for their simple authoring tasks.

The solution to these problems with NoteCards is to add composition as a primitive construct in the basic hypermedia model. Inclusion should be implemented within, as opposed to on top of, all hypermedia systems. Moreover, all aspects of hypermedia should support inclusion (or part-of) relations as a construct distinct from standard (reference) links. Whether or not inclusion relations share a common implementation mechanism with standard links is unimportant, so long as the semantics of inclusion, as opposed to reference, are fully supported.

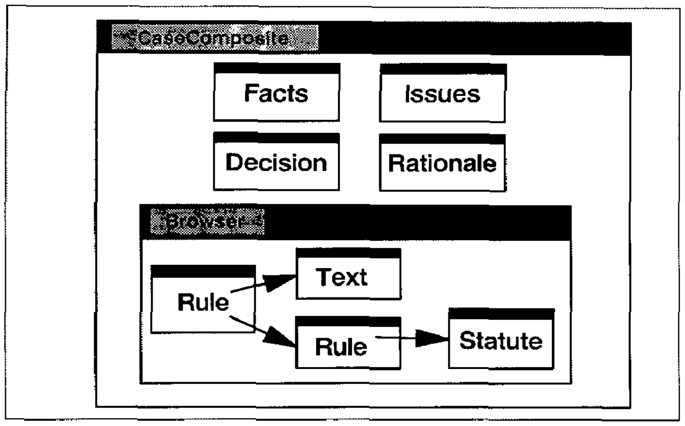


FIGURE 6. Proposed CaseComposite to Replace the CaseCluster shown in Figure 5

Figure 6 shows a schematic example of the CaseCluster redesigned to be a CaseComposite. In this redesign, the head card is replaced by a composite node that directly or indirectly contains all of the components nodes. One of the nodes it directly contains is a browser, which itself is a composite that contains the subnetwork it depicts. Note that this is a significant change from the NoteCards browser which references, but does not contain, the subnetwork it depicts.

The display in Figure 6 is very reminiscent of the displays produced by IGD (Interactive Graphical Documents) [14], [15], a hypermedia system designed to support the creation and presentation of electronic technical manuals. IGD includes a well developed notion of hierarchical node composition. IGD's basic nodes are called pages and serve both as nodes in the hypermedia network and as leaf nodes in a composition hierarchy. Pages contain the textual and graphical content of the system. The interior nodes in the composition hierarchy are called chapters and serve to recursively group pages and chapters into higher level chapters. The design of IGD paid special attention to the issues of displaying composites (chapters), including strategies for using the composition hierarchy to minimize clutter and hide unnecessary details. Interestingly, many of the solutions developed in IGD are redeveloped by Harel [27] in his higraph formalism.

IGD aside, the current generation of hypermedia systems generally lacks adequate composition mechanisms. Designing a composition mechanism appropriate for inclusion in these systems raises a host of interesting questions and issues including, for example:

- Can a given node be included in more than one composite?
- Do links necessarily refer to a node per se or can they refer to a node as it exists within the context of a given composite? If the latter is possible, what does it mean to traverse a link?
- How does one handle versions of composite nodes? Does a new version of an included node necessarily imply a new version of the composite?
- Should composites be implemented using specialized links or is a whole new mechanism necessary?

These issues present a challenging design problem for future hypermedia systems. However, the task is not impossible. NLS/Augment [11] has (of course) already pioneered much of the territory. The goal for the next generation, then, is to pick up where the pioneers left off and to further develop the role of composition in hypermedia systems.

Issue 3: Virtual Structures for Dealing with Changing Information

Hypermedia systems tend to have difficulty with rapidly changing information. This difficulty arises from the essentially static and fragmentary nature of the hypermedia data model. By definition hypermedia encodes information into a collection of independent nodes interconnected into a static network. This network does not change unless it is explicitly edited by the user or some other external agent. In particular, the network cannot reconfigure itself in response to changes in the information it contains. This lack of dynamic mechanisms limits the utility of hypermedia in many task domains.

In NoteCards, the static nature of hypermedia has led to what can be called "the problem of premature organization." NoteCards requires its users to segment their ideas into individual nuggets to be stored away, one per card. Each of these cards then needs to be assigned a title and filed in at least one filebox. Empirical observations [37] have shown that these three seemingly trivial tasks pose significant problems for many users. In particular, a user in the very early stages of working with a particular set of information may not sufficiently understand the content and structure of that information. Knowledge about the critical dimensions of the idea space, the characteristics which distinguish one idea from another, and appropriate naming schemes develops over time as the user becomes familiar with the information. A problem arises because the segmentation, titling, and filing tasks all require the user to have such knowledge up front. As the user's knowledge of the information space evolves, previous organizational commitments (titles and filing categories) become obsolete.

Experienced NoteCards users get around this problem by adopting various strategies to delay the segmentation, titling, and filing of information. To avoid premature segmentation, these users will place the entire idea stream in a single text card. They will go back and review the entire stream before segmenting into separate cards. To avoid premature filing, experienced users file all cards in a single filebox and then use a sketch card to organize their cards into piles based on some judgment of similarity or belongingness. These piles can be easily shifted or rearranged when new information comes in. When the piles are stable, they can be transferred into a filebox structure.

These solutions to the problems of premature organization serve to highlight the difficulties of handling rapidly changing information in NoteCards. Even with these advanced strategies, users' conceptual structures have a tendency to change faster than their corresponding NoteCards structures. The result is that the NoteCards structures are often obsolete with respect to the users' current thinking. To some extent, this situation is unavoidable because it will always be easier, quicker, and less tedious to change one's internal conceptual structures than it will be to update the external representations of these conceptual structures.

NoteCards' difficulties in dealing with changing information could be partially eliminated by improvements in the user interface. Relaxation of the strict titling and filing requirements is an often requested NoteCards enhancement that would certainly help minimize this pressure. Providing less stringent organizational structures such as the similarity piles described previously would also provide a more natural environment for some organizational tasks. Increasing the ease by which structures could be modified (for example, improving browser-based editing) would make it easier for users to track their changing internal structures.

At a more fundamental level, however, the solution to the problem of premature organization is a relaxation of the overly static nature of hypermedia. Specifically, the hypermedia model needs to be augmented with a notion of virtual or dynamically-determined structures. In the current model, nodes and links are extensionally defined, that is nodes and links are defined by specifying the exact identity of their components. In contrast, virtual structures are defined intentionally, that is, by specifying a description of their components. The exact subcomponents of a virtual structure are determined by a search procedure whenever the structure is accessed or instantiated. For example, a virtual composite node might be defined by a specification of the form: a subnetwork containing all nodes created by someone other than me in the last three days. Each time this composite was accessed, its structure and content would be recomputed.

The notion of virtual structures for hypermedia is a direct adaptation of the concept of views (a.k.a. virtual tables) in the world of relational database systems [8]. In a relational database, a view is a table constructed at instantiation time by applying a view definition to data explicitly stored in base (or nonview) tables. The goal (which is not always realized in implementation) is to make view-based tables identical to base tables from a user's perspective. All the same operations should apply, including updates. Although virtual structures in a hypermedia network would be significantly more complex than views in a relational database, the same principle of nondifferentiation at the interface should apply. Any operation possible on a base hypermedia entity should apply to virtual structures as well.

The notion of virtual structures in hypermedia would be possible only in a system that supported a substantial search/query mechanism over the hypermedia network. The definition of the components in virtual structures are in fact queries. Instantiating a virtual structure involves satisfying these queries and constructing a dynamic entity from the results. Although it is not a strict requirement, it would make sense for the query language used for virtual structure descriptions to be the same as the query language used for searches and interface filters.

Virtual structures are a particularly powerful mechanism when combined with the notion of composites. A virtual composite allows the user to create nodes that are dynamically constructed at access time from other nodes, links, and composites that are stored in the network. Such virtual composites are true hypermedia entities and not simply a display of results from a query. Thus, the user can add links, properties or additional static descriptions to a virtual composite. Browsers, for example, could be implemented as virtual composites built from the results of a structure query.

Virtual links are also an intriguing possibility. Such links could, for example, specify their source extensionally and their destination intentionally. Thus one could link from the "ClaimX" node to "the node containing the currently strongest evidence that supports ClaimX." This link could even be created in the absence of any evidence in support of ClaimX. This would effectively create a dangling link with the expectation that an appropriate destination node will appear at some later time. To take the example one step further, one NoteCards user requested conditional links, i.e., links whose specification is of the form: if evidence Q is present, then link from node A to the node containing Q; otherwise link from node A to the node containing P. In general, a simple virtual link is not sufficient to implement arbitrary conditional links. However, depending on the exact semantics of the query processor being used (in particular on how multiple hits are ordered), the specific example above could be satisfied by a virtual link whose source is node A and whose destination is specified by a query of the form: (or (contains Q) (contains P)).

The notion of virtual links in hypermedia has already been explored in ZOG [40] which includes a small set of navigational links that are constructed whenever a node is accessed and displayed. These links connect the displayed node to nodes that the user has recently visited, thereby allowing users the ability to move quickly back from whence they came. In future systems, the notion of virtual links should extend far beyond such simple navigational applications.

Implementing virtual nodes, links, and composites will be a difficult task in the next generation of hypermedia systems, especially when response time is an important factor. Nevertheless, virtual structures will provide these systems with an ability to adapt to changing information in a way that is simply not possible with the current static hypermedia model. Although virtual structures will not replace static structures (since not every relation can be described by a query), they are certain to be critical components in all future hypermedia networks.

Issue 4: Computation in (over) Hypermedia Networks
Hypermedia systems are generally passive storage and retrieval systems. They provide tools for users to define, store, and manipulate a hypermedia network. In service of this goal, they do some processing of the network and the information it contains. For example, most browsers compute the transitive closure defined by a root node and a set of link types. Hypermedia systems, however, do not actively direct the creation or modification of the network or the information contained therein. Unlike expert systems, for example, hypermedia systems do not include inference engines that actively derive new information and enter it into the network.

Although designed as a passive hypermedia system, NoteCards is frequently augmented by active computational engines for particular applications. In one case NoteCards was augmented to function as the delivery vehicle for computer-assisted instruction [41]. In this case, the applications developers implemented a driver that retrieved and interpreted special script cards. These script cards orchestrated the display of other cards containing instructional and test material. Students were expected to answer the test material and their answers were stored in special cards in the network. The driver then used these answers together with the instructions in the script cards to determine what material to display next. In a more advanced version of this application, the driver was a rule-based system that examined a number of cards in the network, including the cards containing the student's previous answers, in order to determine the appropriate material to present.

In the foregoing example there is a clear separation between NoteCards per se and the computational engine embodied in the driver. The computational engine is not integrated into NoteCards. Rather it serves to consume and produce cards and links through the programmer's interface in much the same sense that a (human) user of the system consumes and produces cards and links through the user interface. Aside from the programmatic access to information in the network, NoteCards contains no special support for computational engines of this sort.

It is unclear whether the level of support that NoteCards provides for computational engines is appropriate for future hypermedia systems. Designers of such systems could follow the NoteCards model and continue to support computational engines as separable external entities that create, access, and modify information via the standard programmatic interface. Alternatively, one could design a hypermedia system incorporating a more active computational component that automatically processes (makes inferences from) the information stored in the network. In this case, the hypermedia system would function more like a knowledge-based AI system, both storing and actively processing the information.

The integration of hypermedia and AI technology is an interesting direction to explore. In many ways, hypermedia and knowledge-based systems are a natural fit. In particular, at a high level of abstraction, hypermedia systems, frame-based systems [16], and object-based systems [47] present nearly identical data models. Each of these technologies is based around the notion of typed, slotted entities that form a network structure via inter-entity references. The technologies differ in the specific aspects of this basic model that are chosen for further development. Hypermedia systems focus on heavyweight entities (whole documents) and inter-entity references (links). Object-based systems focus on defining the type (class) hierarchy for the entities and the operations (methods) that can be performed on the instances of each type. Frame-based systems focus on issues such as inheritance and defaulting of slot values, as well as on the integration of truth maintenance, inference engines, and rule-based reasoning with the frame representations. It is this latter focus on computational engines that is of most interest in the present context. A merging of concepts from frame-based systems into the design of hypermedia systems would be a sensible way to approach the integration of hypermedia with rule-based, truth-maintenance, and other computational engines. In a more general context, it is clear that a liberal borrowing of ideas from frame-based and object-based technologies would be very beneficial to the advancement of hypermedia systems.

To a great extent the choice between a passive and an active hypermedia system is an efficiency versus generality trade-off. The ultimate functionality for the approaches is approximately the same. However, the distribution of responsibility and effort is different. Computation built into the hypermedia system is likely to be more efficient, especially when that computation involves extensive access to information in the network. In contrast, an external computational engine is less restricted since no commitment to a particular computational engine needs to be made when the system is implemented. Thus, the choice between an active and a passive hypermedia system will be determined largely by the intended applications and the performance needs of these applications. Whichever architecture is chosen, it is clear that hypermedia systems in the next generation will have to accommodate the integration of hypermedia with computation.

Issue 5: Versioning

Versioning is an important feature in hypermedia systems. A good versioning mechanism will allow users to maintain and manipulate a history of changes to their network. It will also allow users to simultaneously explore several alternate configurations for a single network. Unfortunately, NoteCards has no versioning mechanism. Each card and link exists in only one version and is altered in place when modified. As a result, the range of applications that can be easily supported in NoteCards is severely limited. For example, NoteCards has never been used for maintaining software due, in part, to the overwhelming importance of versioning in configuration management. The lack of versioning has had a lesser impact on the authoring, argumentation, and idea processing tasks for which NoteCards was originally designed. Although users in these applications frequently request versioning support, they have been able to make significant progress in its absence.

NoteCards lags behind its cohort systems in this arena. Both Neptune and Intermedia provide some versioning support. Neptune, for example, provides a time-based linear version thread for individual nodes and links. The system also provides a partitioning scheme called contexts [10] that allows users to begin an independent version thread for a given set of nodes. Intermedia, like NoteCards, keeps only a single version of each node. Intermedia does, however, have a notion of alternative named versions of the web, i.e., the set of links that interconnect a collection of documents. This allows each user, project, or application to work with its own network structure over a common set of document nodes.

PIE [22] included a more extensive versioning mechanism than either Neptune or Intermedia. In the PIE model, versioning in a hypermedia network occurs at two levels: the level of individual entities (nodes, links, composites) and the level of changes to the hypermedia network considered as a whole. At the lower level, each entity has its own version history. In PIE the version history was a linear thread, but in the general case it could be a directed acyclic graph. Although the issue was not specifically addressed in PIE, it is important to provide (virtual) entities corresponding to both specific versions of an entity and the differences (deltas) between successive versions. Both the versions and the deltas should be addressable within the system, that is, they should be possible hits for a search. In a software engineering context it should be possible to search for either the version that implements Feature X or the set of changes that implements Feature X. One possibility would be to treat the deltas between successive versions as special hypermedia nodes capable of being annotated and referenced.

Providing a branched version history for all entities in a hypermedia network raises some very difficult issues regarding the semantics of references between entities. In particular, a reference to an entity may refer to a specific version of that entity, to the newest version of that entity, to the newest version of that entity along a specific branch of the version graph, or to the (latest) version of the entity that matches some particular description (query). Which of these reference types is supported is a decision that affects the entire hypermedia system. It is an especially critical decision in the design of links and composites. Moreover, composites raise the related problem of propagating version changes from subcomponents to their composites. For example, a significant change in an individual software module implies the creation of a new version of the system of which that module is a part. In contrast, updating the spelling of a few words in a paragraph may not require the creation of a new version of the document(s) containing that paragraph.

Although maintaining a version thread for each individual entity is necessary, it is not a complete versioning mechanism. In general, users will make coordinated changes to a number of entities in the network at one time. A software developer may implement a new feature by making coordinated changes to a number of separate modules. The developer may then want to collect the resultant individual versions into a single version set for future reference. This set would be a snapshot of the collection of entities at some particular point in time, that in the software domain is often called a release.

An alternative to version sets, is the *layer* mechanism used in PIE. A layer is a coordinated set of changes to one or more entities in the network. For example, all of the changes made to various modules in a software system could be collected into a single layer described as the changes that implement Feature X. The resulting layer could then be applied to the prechange versions of the entities to get the postchange versions.

In a layer-based system, the primary issue is the set of mechanisms available for managing and for composing layers. In PIE, there were constructs called *contexts* that were essentially sequences of layers. The hypermedia network for a given context was determined by applying the first layer to the base network and then applying the next layer to the result and so on through the sequence of layers in the context. New contexts could be created by mixing and matching the layers from other contexts and then producing a hypermedia network by successive applications of the layers in the context. However, not all such constructions made semantic sense. PIE attempted to aid the user in determining which contexts were sensible and which were not.

One important application of the notion of contexts is to provide a collaborative system (see Issue 6) in which each user maintains a private context through which the user interacts with the hypermedia network. Each user's context contains a base layer that is the public hypermedia network. On top of this base are one or more personal layers which alter the base network to provide a personalized view. At any time a user's view can become public by applying the layers in the user's view to the layers in the base system (and informing any collaborators to use this new base network in their contexts).

Although PIE never emerged from the research prototype stage, aspects of the PIE versioning mechanism later appeared in the Loops knowledge base [2] and in Definition Groups [3]. Unfortunately, the lessons of PIE have been largely ignored by the hypermedia community. Although the PIE versioning scheme is not perfect, it is richer and more complete than those provided by most hypermedia systems. One of the critical design issues facing the next generation of systems is a rich versioning model for hypermedia networks. The PIE model seems like an appropriate place to start in resolving this issue.

Issue 6: Support for Collaborative Work

Hypermedia is a natural medium for supporting collaborative work. Creating annotations, maintaining multiple organizations of a single set of materials, and transferring messages between asynchronous users are the kinds of activities that form the basis of any collaborative effort. These are also activities for which hypermedia systems are ideally suited.

Unfortunately, NoteCards was originally designed as a single user system. Collaboration was seen as occurring outside of the system, through face-to-face conversations, electronic mail, or hardcopy documents. In practice, however, this has rarely been the case. Most idea processing and information management tasks are inherently collaborative, with groups of two to ten people working on a common project. As detailed in [50], NoteCards users have to work hard to overcome the system's limitations in such collaborative environments.

Other members of the current generation fare slightly better than NoteCards in this arena. KMS, Intermedia, and Neptune all provide simultaneous multiuser access to their hypermedia networks. Although this access is supported by some degree of concurrency control, none of these systems includes the kind of sophisticated concurrency management necessary for efficient collaborative work. For example, no system includes a mechanism for notifying users of important actions being taken by other users. More importantly, none of the systems provides support for collaboration beyond the simple mechanics of sharing data. The social interactions involved in sharing a hypermedia network are simply not supported.

The next generation of hypermedia needs to drastically improve support for collaborative work in these two disparate but interrelated areas: the mechanics of simultaneous multiuser access to a common network, and the social interactions involved in collaboratively using a shared network. Supporting the mechanics of multiuser access involves extending the standard technologies for shared databases (e.g., transactions, concurrency control, and change notification) to handle the special requirements engendered by hypermedia. With respect to transactions, hypermedia systems clearly need to support long to very long transactions. For example, if a network contains large documents, a user may require anywhere from a few minutes to several days to make a single update to a node. This update should be atomic, independent of the duration of the update. Techniques for handling such long transactions are areas of active study in the database literature [18] and should be incorporated into hypermedia systems.

A closely related issue is concurrency control in hypermedia networks. Classical hard locking techniques for concurrency control are generally inappropriate for multiuser hypermedia. Standard read/write locks tend to be overly restrictive, preventing multiple users from working on the same node or link even when it is appropriate for them to do so. For example, it is often appropriate for one person to be updating a node while another is annotating it by creating links. With a simple read/write locking protocol, the write lock obtained by the updator would prevent simultaneous link creation by the annotator. The solution to this particular problem is to provide locking at a finer grain of activity: updating the contents of a node should be locked independently of creating/updating links into and out of the node. On a more general level, collaborative use of hypermedia networks requires softer locking protocols such as those discussed in [24]. These protocols allow a greater degree of concurrent work on a single entity (node or link) at the risk of increasing the number of conflicting operations on that entity. The impact of the increase in conflicts is minimized by conflict resolution schemes that are less drastic than simply aborting one of the conflicting operations. For example, in a hypermedia network with a sophisticated versioning model, a conflicting update could simply create a branch in the version tree for the affected node (or link) and then notify the users that there is a conflict to be resolved offline.

Notification of interested parties when certain critical events occur is another mechanism of special importance to collaborative hypermedia. In general, users should be notified when an important event has occurred on any node, link, or composite of interest to them. This notification should take place as early as possible—notification should occur when a user signals an intention to update a node (by obtaining a write lock on that node) rather than when the updated node is actually entered into the network [48]. Such early notification ensures that collaborative users can detect possible conflicts as early as possible and modify their actions accordingly. Thus, notification is a partial solution to the increase in conflicts brought about by soft locks. More importantly, notification is an automatic mechanism that helps multiple users coordinate their ongoing work in a shared hypermedia network.

In the area of support for the social interactions involved in collaborative use of a shared network, the critical notion is *mutual intelligibility* [50]. In a collaborative effort, each participant must have some degree of understanding of the actions and intentions of any collaborators. In the hypermedia context, collaborative users need to understand not only the content of the collaborative network but also the accepted procedures to be used to modify that network. Trigg, Suchman, and Halasz [50] describe three kinds of collaborative activities that occur within a shared hypermedia network. Substantive activities constitute the work at hand, the writing of the coauthored paper. Annotative activities involve annotating the substantive work using comments, critiques, questions, etc. Procedural activities involve discussions, decisions, and other actions focusing on the use of the network and the procedures for collaboration. Writing a coauthored paper is a substantive activity. Commenting on your coauthor's draft is an annotative activity. Deciding how to distinguish additions and deletions as the draft is passed back and forth is a procedural activity. Trigg et al. suggest that existing hypermedia systems focus primarily on supporting substantive and annotative activities. Explicit support for procedural activities is very rare.

In examining collaborative use of NoteCards, Trigg et al discovered that even without explicit support, users engaged in procedural activities including maintaining history cards detailing changes made during each session, creating and maintaining a bulletin board for posting messages between asynchronous collaborators, and engaging in explicit discussions (within a card) about proper conventions for carrying out specific tasks in the network. An example of the latter was a discussion about how to use font styles to distinguish the specific contributions of the various collaborators. All of these procedural activities were supported in NoteCards using ad hoc structures and labor-intensive procedures. Clearly, much of this work could be directly supported by the system. Operations such as maintaining change histories and message posting areas, tracking and displaying the specific contributions made by each individual, and recording decisions about usage conventions could easily be automated or semiautomated within a hypermedia system.

At a more abstract level, achieving mutual intelligibility in hypermedia-based collaborations will require the development of a rhetoric of hypermedia. This rhetoric will provide guidelines and conventions for creating hypermedia networks that will be understandable by others who share the rhetoric. Interestingly, the development of these conventions is a crucial issue that can only be solved by accumulated experience in using hypermedia systems in real world tasks.

Based on his several years experience in developing courseware materials for the Intermedia system, Landow [30] has developed the beginnings of just such a rhetoric. Of particular interest is Landow's observation that materials developed for nonlinear presentation in a hypermedia system require special discourse techniques to accommodate the process of link traversal. These techniques, which Landow calls the *rhetoric of arrival* and the *rhetoric of departure*, are needed to provide the reader with information about the relationship being indicated by the link. The departure rhetoric has to convey information about why the link should be followed, while the arrival rhetoric needs to convey information about how the node just arrived at relates to the context from which the user departed. These new rhetorical devices, and many more like them, are needed to help users deal with the difficult issues engendered by the ability to create documents that no longer have a fixed linear ordering. As our experience with hypermedia increases, many more such devices should evolve.

The next generation of hypermedia needs to focus equally on improvements in the technologies for shared databases and on improvements in the support for the social interactions involved in the collaborative use of a shared network. Both are necessary if hypermedia is to realize its potential as an ideal vehicle for supporting collaborative work.

Issue 7: Extensibility and Tailorability

The broad applicability of hypermedia systems stems from the inherent flexibility of the basic hypermedia model. The hypermedia model provides a set of very general abstractions, nodes, links, and composites. The user is responsible for properly applying these abstractions to the task at hand. In this sense, a hypermedia system is just a tool for creating, manipulating, and displaying nodes of information embedded into a network structure. To the system, all nodes and links are essentially the same: objects to be stored, retrieved, displayed, interconnected, etc. To the user, nodes and links are filled with meaningful contents and organized into meaningful structures. The system cannot operate directly on this meaning. It simply provides a collection of generic tools that can be used to manipulate networks in meaningful ways.

This generic nature of hypermedia systems is both a blessing and a curse. It is a blessing because it allows hypermedia to be useful in a wide variety of task domains and user populations. It is a curse because generic hypermedia is not particularly well suited to any specific task or style of use. Thus, hypermedia users are faced with a tool that is clearly useful but not particularly well adapted to the specific task at hand. In the NoteCards user community, for example, the most frequent complaint is the lack of a strategy manual with examples showing successful uses of the system in specific tasks. The problem is that each new NoteCards user is faced with a significant database design task. The user has a familiar collection of information that must be translated into some NoteCards structure. This representation task is not always straightforward since the familiar structure of the information may be very different from the cards, links, and fileboxes provided by the system. For example, in a legal application the user might have entities such as cases, briefs, evidence, citations, etc. that need to be filed and interconnected in accordance with a standard set of legal relationships. To the casual user it is not obvious how to do this. Should each case be a text card or a filebox? Should evidence be stored in the same card as the case or in a separate card connected to the case by a link? What type of link? In large database systems, such design problems are usually assigned to a professional database designer. In hypermedia systems, these problems are frequently left in the laps of individual users.

To offset the difficulties caused by their generic nature, most hypermedia systems are designed to be extensible and tailorable. The expectation is that users will extend the system with new functionality or tailor the existing functionality to better match the exact requirements of their application. Users with legal applications might tailor the system's user interface to their specific task by replacing the generic nodes and links with the appropriately specialized entities such as cases, briefs, and citations. The CaseCluster shown in Figure 5 is an example of such a tailored entity.

In NoteCards, as well as in systems such as Intermedia and Neptune, the primary mechanism for extensibility and tailorability is a programmer's interface which provides programmatic access to all of the system's functionality including the mechanisms for extending the system's hierarchy of node and link types. In practice, the NoteCards programmer's interface has been very successful. In addition to the many users who have done minor tailoring of the NoteCards interface, there have been several major new systems built on top of NoteCards using the programmer's interface [25], [51]. Trigg, Moran, and Halasz [51] describe in some detail the extensible and tailorable nature of NoteCards, including examples of its use, problems with its implementation, and prospects for its improvement.

Despite its success, the NoteCards programmer's interface failed to meet one of its explicit design goals: that minor changes to NoteCards should be achievable with a small amount of work by casual, nonprogramming users. In its current form, the programmer's interface provides significant functionality only to those who have a fair degree of expertise in programming and system implementation. Unfortunately, this is true of the tailoring interfaces in most of the current generation of hypermedia systems as well. The challenge, then, is to design mechanisms for the small tailorability of hypermedia systems. The goal is to make it easy for the large majority of nonprogramming users to make small changes to the system with a minimal amount of effort. This goal should be achieved without interfering with the current facilities that allow expert programmers to make major changes or extensions to the system.

GNU Emacs [46] and Apple Hypercard/Hypertalk [23] are two systems worthy of study. Neither is a true hypermedia system (although both include some hypermedia features), but both have a kind of extensibility and tailorability that should be standard in future hypermedia systems. The critical characteristic of both is that they are built around an interpreter for a fully-functional programming language that is specially designed for the kind of objects and operations commonly handled by the system. Aside from a basic kernal, all of the system's functionality is implemented using this programming language. Moreover, the interface to the language/interpreter has a kind of scalability. Simple things can be done using single commands or even by direct manipulation. Yet it is also possible to write large and complex programs to make major changes or extensions. In GNU Emacs, this language is a dialect of Lisp with extensions to handle the common operations on Emacs objects such as files, buffers, and windows. Emacs includes mechanisms for incorporating new or amended Lisp functions into a running system, for evaluating single Lisp expressions from the user interface, and for creating "programs" by enacting the keystrokes that would normally be used to implement the desired functionality. The result of these mechanisms is a highly adaptive system that can be tailored by users with a wide variety of expertise and requirements.

The goal of the next generation of hypermedia should be to develop systems with extension and tailoring facilities analogous to those currently found in GNU Emacs and Hypercard. Without these facilities, many potential users will discover that the problems of dealing with an overly generic system outweigh the many benefits of using hypermedia.

CONCLUDING REMARKS

The recent surge in the popularity of hypermedia systems has been accompanied by considerable hype about the tremendous power and functionality inherent in hypermedia. Our experiences with NoteCards suggest that there is indeed reason to be excited about the prospects of using hyermedia networks as a basic information management and representation technology for a wide variety of applications. But the seven unresolved issues discussed in this article should serve as a cautionary note. There is a great deal of difficult design and implementation to be done before hypermedia systems can achieve their potential. The seven issues suggest that the current model needs to be extended beyond simple nodes and links to include compositions, queries, versioning, computation, and many other features. These seven issues thus represent an agenda for the designers and implementors of the next generation of hypermedia systems.

Acknowledgments. I would like to thank Randy Trigg and Tom Moran of Xerox PARC and Jeff Conklin, Michael Begeman, and Eric Gullichsen of MCC. The ideas presented in this paper were developed during many discussions with these folks.

REFERENCES

1. Akscyn, R., McCracken, D.L., and Yoder, E. KMS: A distributed hypertext for sharing knowledge in organizations. *Commun. ACM 31*, 7 (July 1988), 820–835.
2. Bobrow, D.G., and Stefik, M. *The Loops Manual*. Intelligent Systems Laboratory, Xerox Palo Alto Research Center, Palo Alto, Calif., 1983.
3. Bobrow, D.G., Fogelsong, D.S., and Miller, M.S. Definition groups: Making sources into first-class objects. In *Research Directions in Object-Oriented Programming*, B. Shriver and P. Wegener, Eds. MIT Press, Cambridge, Mass., 1987, pp. 129–146.
4. Brachman, R.J., and Schmolze, J.G. An overview of the KL-ONE knowledge representation system. *Cognitive Science 9*, 2 (1985), 171–216.
5. Brown, P.J. Turning ideas into products: The Guide system. In *Proceedings of the Hypertext '87 Workshop* (University of North Carolina at Chapel Hill, Nov.). ACM, New York, 1987.
6. Conklin, J. Hypertext: A survey and introduction. *IEEE Computer 20*, 9 (1987), 17–41.
7. Cruz, I.F., Mendelzon, A.O., and Wood, P.T. A graphical query language supporting recursion. In *Proceedings of the ACM SIGMOD Annual Conference* (San Francisco, Calif., May). ACM, New York, 1987, pp. 323–330.
8. Date, C.J. *An Introduction to Database Systems vol. 1*. Addison-Wesley, Reading, Mass., 1981.
9. Delisle, N., and Schwartz, M. Neptune: A hypertext system for CAD applications. In *Proceedings of ACM SIGMOD '86* (Washington, D.C., May 28–30). ACM, New York, 1986, pp. 132–142.
10. Delisle, N., and Schwartz, M. Contexts—a partitioning concept for hypertext. In *Proceedings of the Conference on Computer-Supported Cooperative Work* (Austin, Texas, Dec. 3–5). 1986, pp. 147–153.
11. Englebart, D.C., and English, W. A research center for augmenting human intellect. In *Proceedings of 1968 FJCC* (San Francisco, Calif., Dec. 9–11). AFIPS Press, Montvale, N.J., 1968, pp. 395–410.
12. Englebart, D.C. Authorship provisions in Augment. In *Proceedings of the IEEE COMPCON* (San Francisco, Calif., Spring). IEEE, New York, 1984, 465–472.
13. Fairchild, K.F., Poltrock, S.E., and Furnas, G.W. SemNet: Three-dimensional graphic representations of large knowledge bases. In *Cognitive Science and its Applications for Human-Computer Interaction*, R. Guindon, Ed. Lawrence Erlbaum Associates, Hillsdale, N.J., 1988.
14. Feiner, S., Nagy, S., and van Dam, A. An experimental system for creating and presenting interactive graphical documents. *ACM Trans. Graphics 1*, 1 (1982), 59–77.
15. Feiner, S. Seeing the forest for the trees: Hierarchical display of hypertext structure. In *Proceedings of the Conference on Office Information Systems* (Palo Alto, Calif., Mar.). ACM, New York, 1988, pp. 205–212.
16. Fikes, R., and Kehler, T. The role of frame-based representation in reasoning. *Commun. ACM 28*, 9 (1985), 904–920.
17. Furnas, G.W. Generalized fish-eye views. In *Proceedings of the 1986 ACM Conference of Human Factors in Computing Systems (CHI '86)* (Boston, Mass., Apr. 13–17). ACM, New York, 1986, pp. 16–23.
18. Garcia-Molina, H., and Salem, K. Sagas. In *Proceedings of the ACM SIGMOD Annual Conference* (San Francisco, Calif., May). ACM, New York, 1987, pp. 249–259.
19. Garg, P.K., and Scacchi, W. A hypertext system to manage software life cycle documents. In *Proceedings of The 21st Hawaii International Conference on Systems Science* (Honolulu, Hawaii, Jan.), 1987.
20. Garrett, L.N., and Smith, K.E. Building a time-line editor from prefab parts: The architecture of an object-oriented application. In *Proceedings of the Conference on Object-oriented Programming Systems, Languages, and Applications (OOPSLA '86)* (Portland, Oregon, Sept. 29–Oct. 2). *ACM SIGPLAN Not. 21*, 11 (1986), pp. 202–213.
21. Garrett, L.N., Smith, K.E., and Myrowitz, N. Intermedia: Issues, strategies, and tactics in the design of a hypermedia document system. In *Proceedings of the Conference on Computer-Supported Cooperative Work* (Austin, Texas, Dec. 3–5). 1986, pp. 163–174.
22. Goldstein, I., and Bobrow, D. A layered approach to software design. In *Interactive Programming Environments*, D. Barstow, H. Shrobe, and E. Sandewall, Eds. McGraw-Hill, New York, 1987, pp. 387–413.
23. Goodman, D. *The Complete HyperCard Handbook*. Bantam Books, New York, 1987.
24. Greif, I., and Sunil, S. Data sharing in group work. *ACM Trans. Office Info. Sys. 5*, 2 (1987), 187–211.

25. Guide Users Manual. Owl International, Bellevue, Wash., 1986.
26. Halasz, F.G., Moran, T.P., and Trigg, R.H. NoteCards in a Nutshell. In *Proceedings of the 1987 ACM Conference of Human Factors in Computer Systems (CHI+GI '87)* (Toronto, Ontario, Apr. 5–9). 1987, pp. 45–52.
27. Harel, D. On visual formalisms. *Commun. ACM 31*, 5 (May 1988), 514–530.
28. Hollaar, L.A. The Utah text retrieval project. *Info. Tech.: Res. Dev. 2* (1983), 155–168.
29. Lambert, S., and Ropiequet, S., Eds. *The New Papyrus.* Microsoft Press, Redmond, Wash., 1986.
30. Landow, G.P. Relationally encoded links and the rhetoric of hypertext. In *Proceedings of the Hypertext '87 Workshop* (University of North Carolina at Chapel Hill, Nov.). 1987.
31. Lowe, D. SYNVIEW: The design of a system for cooperative structuring of information. In *Proceedings of the Conference on Computer-Supported Cooperative Work* (Austin, Texas, Dec. 3–5). 1986, pp. 376–386.
32. Mah, C.P., and D'Amore, R.J. Complete statistical indexing of text by overlapping word fragments. *ACM SIGIR Forum 17*, 3 (Winter-Spring 1983), 6–16.
33. McCracken, D.L., and Akscyn, R. Experience with the ZOG human-computer interface system. *Int. J. Man-Machine Studies 21*, 2 (1984), 293–310.
34. McCune, B.P., Tong, R.M., Dean, J.S., and Shapiro, D.G. RUBRIC: A system for rule-based information retrieval. *IEEE Trans. Softw. Eng. SE-11*, 9 (1985), 939–945.
35. Meyrowitz, N. Intermedia: The architecture and construction of an object-oriented hypermedia system and applications framework. In *Proceedings of the Conference on Object-oriented Programming Systems, Languages, and Applications (OOPSLA '86)* (Portland, Oregon, Sept. 29–Oct. 2). *ACM SIGPLAN Not. 21*, 11 (1986), pp. 186–201.
36. Monty, M.L., and Moran, T.P. A longitudinal study of authoring using NoteCards. *ACM SIGCHI Bulletin 18*, 2 (Oct. 1986), 59–60.
37. Monty, M.L. Temporal context and memory for notes stored in the computer. *ACM SIGCHI Bulletin 18*, 2 (Oct. 1986), 50–51.
38. Mozer, M.C. *Inductive Information Retrieval Using Parallel Distributed Computation.* Tech. Rep. C-015. Institute for Cognitive Science, University of California, San Diego, Calif., 1984.
39. Nelson, T.H. *Literary Machines.* T.H. Nelson, Swarthmore, Penn., 1981.
40. Robertson, G., McCracken, D., and Newell, A. The ZOG approach to man-machine communication. *Int. J. Man-Machine Studies 14* (1981), 461–488.
41. Russell, D.M., Moran, T.P., and Jordan, D.S. The instructional design environment. In *Intelligent Tutoring Systems: Lessons Learned,* J. Psotka, L.D. Massey, & S.A. Muter, Eds. Lawrence Erlbaum Associates, Hillsdale, N.J., 1987.
42. Salton, G., and McGill, M.J. *Introduction to Modern Information Retrieval.* McGraw-Hill, New York, 1983.
43. Shneiderman, B. User interface design for the Hyperties electronic encyclopedia. In *Proceedings of the Hypertext '87 Workshop* (University of North Carolina at Chapel Hill, Nov.). 1987.
44. Shneiderman, B. User interface design and evaluation for an electronic encyclopedia. Tech. Rep. CS-TR-1819. Dept. of Computer Science, University of Maryland, College Park, Maryland, Mar. 1987.
45. Smith, J.B., Weiss, S.F., Ferguson, G.J., Bolter, J.D., Lansman, M., and Bea, D.V. *WE: A writing environment for professionals.* Tech. Rep. 86-025. Dept. of Computer Science, University of North Carolina, Chapel Hill, N.C., Aug. 1986.
46. Stallman, R.M. *GNU Emacs Manual.* Free Software Foundation, Cambridge, Mass., 1985.
47. Stefik, M., and Bobrow, D. Object-oriented programming: Themes and variations. *AI Magazine 6*, 4 (1986), 40–62.
48. Stefik, M., Foster, G., Bobrow, D.G., Kahn, K., Lanning, S., and Suchman, L. Beyond the chalkboard: Computer support for collaboration and problem solving in meetings. *Commun. ACM 30*, 1 (1987), 32–47.
49. Trigg, R.H., and Weiser, M. TEXTNET: A network-based approach to text handling. *ACM Trans. Office Info. Sys. 4*, 1 (Jan. 1986), 1–23.
50. Trigg, R., Suchman, L., and Halasz, F. Supporting collaboration in NoteCards. In *Proceedings of the Conference on Computer-Supported Cooperative Work* (Austin, Texas, Dec. 3–5). 1986, pp. 147–153.
51. Trigg, R.H., Moran, T.P., and Halasz, F.G. Tailorability in NoteCards. In *Proceedings of Interact '87 2nd IFIP Conference on Human-Computer Interaction* (Stuttgart, West Germany, Aug.), Bullinger, H.J., and Shackel, B. (Eds.). North-Holland, Amsterdam, 1987.
52. VanLehn, K. *Theory reform caused by an argumentation tool.* Tech. Rep. Xerox Palo Alto Research Center, Palo Alto, Calif., ISL-11. 1985.
53. Walker, J. Document Examiner: Delivery interface to hypertext documents. In *Proceedings of the Hypertext '87 Workshop* (University of North Carolina at Chapel Hill, Nov.). 1987.
54. Yankelovich, N., Meyrowitz, N., and van Dam, A. Reading and writing the electronic book. *IEEE Computer 18*, 10 (1985), 15–30.
55. Zloof, M.M. Design aspects of the Query-by-example data base management language. In *Databases: Improving Usability and Responsiveness.* B. Schneiderman, Ed. Academic Press, New York, 1978.

CR Categories and Subject Descriptors: H.1.2 [**Models and Principles**]: User/Machine Systems—*human factors*; H.2.4 [**Database Management**]: Systems—*distributed systems, query processing*; H.4.2 [**Information Systems Applications**]: Types of Systems; I.7.m [**Text Processing**]: Miscellaneous
General Terms: Design, Experimentation, Human Factors
Additional Key Words and Phrases: Collaborative work, hypermedia, hypertext

Author's Present Address: Frank G. Halasz, Microelectronics and Computer Technology Corp. (MCC), 3500 West Balcones Center Dr., Austin, TX 78759-6509.