

# Supporting Incremental Formalization with the Hyper-Object Substrate

*Frank M. Shipman III*

Department of Computer Science  
Texas A&M University  
College Station, TX 77843-3112  
(409) 862 - 3216  
E-mail: shipman@cs.tamu.edu

*Raymond J. McCall*

College of Architecture and Planning &  
Institute of Cognitive Science  
University of Colorado, Boulder, CO 80309  
(303) 492 - 7042  
E-mail: mccall@phidias.colorado.edu

## ABSTRACT

Computers require formally represented information to perform computations that support users; yet users who have needed such support have often proved to be unable or unwilling to formalize it. To address this problem, this paper introduces an approach called *incremental formalization*, in which, first, users express information informally and then the system aids them in formalizing it. Incremental formalization requires a system architecture that 1) integrates formal and informal representations and 2) supports progressive formalization of information. The system should have both tools to capture naturally available informal information and techniques to suggest possible formalizations of this information. The Hyper-Object Substrate (HOS) was developed to satisfy these requirements. HOS has been applied to a number of problem domains, including network design, archeological site analysis and neuroscience education. Users have been successful in adding informal information and then later formalizing it incrementally with the aid of the system. Our experience with HOS has reaffirmed the need for information spaces to evolve during use and has identified additional considerations in the design and instantiation of systems enabling and supporting incremental formalization.

**KEYWORDS:** Formalization, structure, hypermedia, knowledge-based systems, knowledge representation, knowledge acquisition.

## 1. INTRODUCTION: THE PROBLEMS OF FORMALITY

A goal of modern software design is to devise high-functionality systems that take proactive roles in helping end-users to perform tasks. Unfortunately, progress in developing such proactive systems has been limited by fundamental difficulties in obtaining formalized information from users — i.e., information whose structure and semantics are indicated explicitly. Current limitations of computers, especially the inability to process unrestricted natural language, mean that input generally must be formalized in order for significant computation to be done on the basis of it. More specifically, proactive systems require such formalization both to identify the tasks users are performing and to aid them in these tasks.

The difficulties in obtaining formalized information arise because users who have the relevant information are often unable or unwilling to formalize it. Thus, the information is available but not in a form usable for computation. This paper analyzes this problem and proposes both a conceptual approach and a system architecture for dealing with it. The approach is something we call *incremental formalization*. The architecture involves the use of a knowledge-based hypermedia system as a computational substrate for creating a variety of applications. A prototype called the Hyper-Object Substrate (HOS) instantiates this architecture.

### 1.1 The Need for Formalization

For the purposes of this paper we define formalization as the process of putting information in a symbolic

form that can be processed by computer, or what Chomsky describes as a “formal language” [1956]. According to Chomsky’s definition, computers can only process information that has been formalized, so some degree of formalization is required for computers to be of any use.

In this paper we will describe representations as having various degrees of formality depending on the amount of structure and semantics explicitly represented within the system. Note that while domain knowledge provides one source for semantics, we are not implying it is the only basis. In fact domain-independent systems.

Figure 1 shows a part of our scale of formality. Text has a low degree of formality since it has little structure and does not explicitly indicate correspondences with domain concepts. Frame-based representation languages involving explicit inheritance hierarchies of domain concepts have a high degree of formality. Media such as raster graphics, audio, and video might be considered less formal than text and graphics on such a scale in a domain such as meeting support [Haake et al. 1994; Saund, Moran 1994].

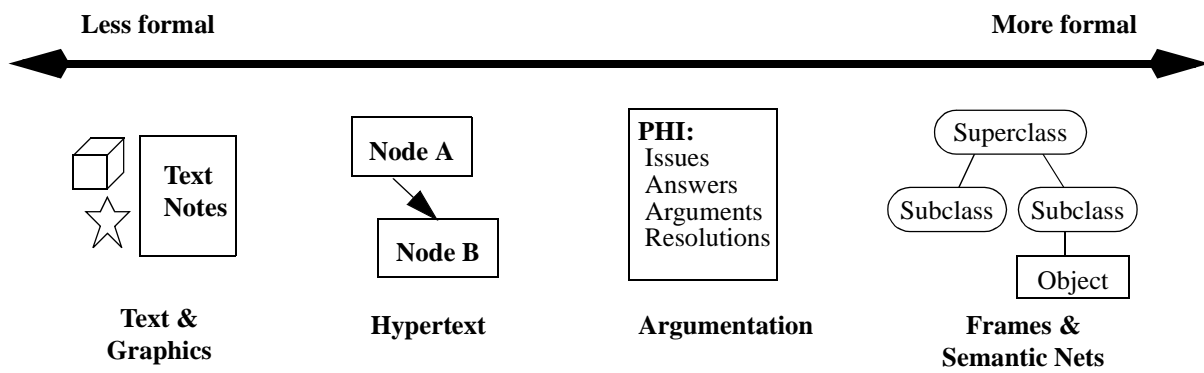


Figure 1: Range of formalisms in domain-oriented systems.

In general, the higher the level of functionality and proactivity needed the greater the level of formalization required. A text editor provides low-level functionality (mostly string manipulation) and requires the very low level of formalization of textual input (an ordered sequence of characters). Knowledge-based systems offer a much higher level of functionality, they are more active in aiding the user, and require a much higher level of formalization of information. Within the representations described, moving information from a less formal to a more formal representation includes the delimiting of “chunks,” as well as the determination of their types, properties (attributes and values) and relationships to other information “chunks.”

The current shift toward work embedded in large-scale distributed networked computing and communication environments dramatically increases the pressure for systems to take a more active role in supporting their users. As an example of this trend, electronic mail readers now frequently support a form of rule-based filtering or sorting based on the electronic mail header. *Domain-oriented systems*, systems which support work in a particular domain, also must add functionality to help the user deal with the growing amount and sources of information available in their domain. In general, these systems need to be able to initiate dialogues with users and adapt to ongoing changes in computational resources and user needs — two central characteristics of what many people call “agents”. Such initiative and adaptability on the part of systems requires formal models that are used to determine when such actions are appropriate and exactly what actions to perform.

## 1.2 The Need for Explicit Formalization by Users

Many people have thought that by pre-loading the computer with a great deal of (formal) knowledge —

e.g., about the users' application domain — little or no explicit formalization by users would be needed. But this “put-in-all-the-knowledge-at-the-beginning” approach fails in many situations. It is inadequate for domains whose knowledge undergoes rapid change — a characteristic of most high-technology problem domains. Traditional approaches to knowledge acquisition require domain experts and knowledge engineers to collaboratively generate an initial domain representation for the system. This initial representation will typically fail to capture the large amounts of tacit knowledge that expert professionals possess, because practical problem solvers know more than they can say, as described both by Polanyi [1966] and Schoen [1983]. Once in use in the context of actual problem solving, relevant pieces of this tacit knowledge tend to surface as explicit knowledge; and domain experts must then contact knowledge engineers to modify the formal domain representation to reflect their new understanding. While this process can work for the support of limited tasks in limited domains, it will be too expensive for domains where the representation must change frequently.

A large number of the knowledge workers who are now looking to computers to help them deal with the increasing volume and complexity of networked information do not have such easily defined tasks as those traditionally supported by knowledge-based systems. Many of these professionals solve what Rittel characterizes as “wicked” problems [1984], such as analysis and design, where the professionals' understanding or the domain and task change during the course of problem solution. As users acquire new knowledge and as tacit knowledge surfaces in problem solving, it needs to be input so that it can be used. It need not be formalized to be used by humans; but to be used in any way by the machine, some formalization is required.

The machine itself could, in theory, formalize the new input automatically and completely through machine understanding of natural language input. Unfortunately, achieving such understanding still seems a long way off. It has not yet proved possible to put nearly enough knowledge into software to enable input to be formalized without the user explicitly telling the machine a great deal about its formal properties. The proactive assistance in software that users and developers are planning for will therefore inevitably create greater requirements for explicit formalization of input by end users. This is problematic, because it seems that users are unhappy about having to explicitly formalize input.

### **1.3 Problems in Getting Explicit Formalization from Users**

A number of problems have been experienced when systems require extensive formalization of information by their users, as described by Shipman and Marshall [1993]. One is that it takes a great deal of extra time and effort. Often this extra effort is not balanced by a payoff at the time the formalization is required and for the persons who must do the formalization. This discrepancy is noted by Grudin [1994] as a particular challenge for groupware. Even relatively small amounts of this extra time and effort can disrupt the flow of work, causing attention to shift from the task to the system, as described in the case of design in [Fischer et al. 1991].

Why is providing a formal representation of information so problematic for users? The difficulties that users experience are not just interface problems. More effort is required of users in part because formal representations require the explicit statement of information that might have been left implicit or tacit in a less formal representation by contextual knowledge. Formalization requires many extra decisions to divide information into chunks, to type and label these appropriately, to link them with useful relationships and assign properties to them. If users end up focusing their energies on formalization rather than the domain task they are supposed to perform, the use of the computer could lead to the quality of their work actually being decreased rather than enhanced.

Additionally, substantial time and effort are typically required to learn a formalism and to develop skill in representing ideas in it. This requires both talent and interest in formalization that users are unlikely to have.

One well-known difficulty of formalization is that of prematurely imposing structure. Suchman [1987] describes how human problem solving is often a discovery process by which people develop a gradual and progressive understanding of their problems and how to solve them. As new situations are encountered, an expert's understanding of the specific problem — and even the domain — will change. Formalizations based on previous understandings will become outdated and counterproductive. This is often a frustrating experience for users, because the extra time they invest in early formalization ends up being wasted. And they must invest still more effort to get any benefit from formalization.

As users experience the frustrations of formalization, they often become not only reluctant to formalize but actively resistant to efforts to get them to do so. When systems require formalization as part of all input, users resist using those systems, or at least their formal aspects, as seen in experiences with Aquanet [Marshall, Rogers 1992]. Thus, expert systems, which require that all input be highly formalized, fail to get maintained without the efforts of professional knowledge engineers. Argumentation-based hypertext systems for design that emphasize support for authoring — such as Conklin and Burgess Yakemovic's gIBIS [1991] and McCall and colleagues' MIKROPLIS [1983] — typically fall into disuse without strong proponents among the user community [Conklin, Burgess Yakemovic, 1991]. Experiences with general purpose hypertext systems, such as NoteCards and the Virtual Notebook System, show users having difficulty deciding on appropriate node sizes and labels for their documents [Shipman, Marshall 1993]. Hypertext systems, of course, are generally domain independent and require a far lower level of formalization of information, and thus fewer chunking and labelling decisions, than expert systems.

#### **1.4 Analysis of These Problems**

The situation is ironic. On the one hand, users — and software designers — want proactive, high-functionality systems to support their daily work. On the other hand, users resist the formalization of information that is required for such higher-level functionality.

At first glance it might seem that we are facing an intractable dilemma: users demand higher levels of functionality yet are unwilling to pay the inevitable price of increased formalization efforts. But the appearance of dilemma here is illusory. It derives from the mistaken notions that formalization is an all-or-nothing and all-at-once proposition. We argue here that an incremental approach to formalization is both acceptable to users and effective in getting far greater amount of formalized information into systems. With this approach formalization need not be done at the time of information input and exclusively by the user; it can be done during system use and with substantial support from the computer.

In the following section we explain the approach we call incremental formalization. We then propose a software architecture for supporting this approach — an architecture instantiated in a prototype we call the Hyper-Object Substrate (HOS). Following this, we describe experiences in using HOS for several types of applications that benefit from incremental formalization. We conclude by summarizing the benefits of HOS and by identifying issues that arose from our experience with the system.

## **2. APPROACH: INCREMENTAL FORMALIZATION**

A number of different approaches can be and have been taken to solving the problem of users' resistance to formalization. One approach is to find a formal representation that is easy for users to work with — a goal of end-user programming languages like those of Eisenberg and Fischer [1994]. A second approach attempts to provide help to the user in learning and using a formal language (e.g., LISP) — a goal of “end-user modifiable” systems as described by Girgensohn [1992]. A third approach — radically different from either of the prior two — attempts to make the system understand the informally represented information — a goal of natural language processing.

The approach presented in this paper synergistically combines aspects of all three of the above-described approaches. Users enter information into the system in an informal representation. The computer then aids the users in formalizing this information by using its current knowledge to suggest partial formalizations of

the informal information. To further formalize the information, the user interacts with the system, which uses various aids to help users to represent information in a relatively intuitive representation scheme.

Incremental formalization aims, first of all, to eliminate the dangers of premature structuring and the cognitive costs associated with formalization that inhibit user input by allowing information to be entered in an informal representation. Secondly, it aims to reduce the burden of formalization by distributing it and making it demand and task driven. By having the system play an active role in the formalization process, incremental formalization helps users recognize and express previously tacit knowledge.

### **2.1 The Process of Incremental Formalization**

Incremental formalization allows information to be entered in an informal representation and to become more formalized over time through a series of small and simple intermediate steps. As formalization proceeds, progressively more of the following are identified: 1) “chunks”, i.e., meaningful units, 2) labels for various chunks 3) relevant attributes, values and relationships, 4) associations among chunks and relevant attributes, values and relationships, 4) constraints, 5) generalizations (both rules and inheritance relationships), and 6) aggregate constructs (composites and part-of relations).

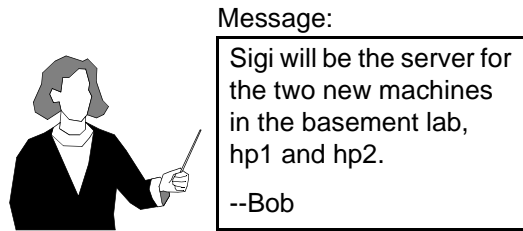
Figure 2 shows an example progression of formalizing information in the domain of network design. Figure 2a shows a textual message added to the system. In Figure 2b, relevant attributes from the domain, such as the locations and machines mentioned in the message, are attached to the message object. In Figure 2c additional chunks (with labels) representing domain entities (specific machines or locations) are formed and associations replace some of the previous textual attribute values for the message. Additionally, attributes representing important characteristics of the domain entities are added. In Figure 2d generalizations of domain concepts, i.e. the laboratory and workstation object classes, are identified and inheritance relationships are added between them and existing objects (indicated by the dark arrows). Not shown in the figure, constraints, such as requirements on what type of domain entity can fill the server attribute for workstations, and aggregate constructs, defining part-whole relationships between workstations and logical networks, could also be added.

There are two, complementary ways in which incremental formalization can proceed. One is the piecewise addition of formal elements — including, chunks, attributes, relationships and classes, as shown in the above example. The second is the progressive refinement of these elements. One way this happens is when chunks are broken into still finer-grained chunks. Another type of refinement takes place when the classes of objects (chunks/nodes) and relationships (links) are progressively specialized. Thus, for example, the link between a pair of objects, x and y, might start as merely “x is related to y.” Later on this relationship might be specialized to “x is a comment on y.” Still later it might become “x is an argument about y,” and still later “x is an argument against y.”

### **2.2 Our Approach to Incremental Formalization**

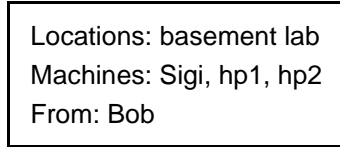
A crucial feature of our approach to incremental formalization is that it is non-destructive. When a formal representation is produced, it supplements rather than replaces the less formal representations — the original message is still part of the information space at the end of the process in our example above. This is because formalization generally results in a loss of information content — e.g., unstated background knowledge and rationale. Keeping the informal knowledge from which formal knowledge is derived helps us to check and revise formal representations, as also found by Hofmann and colleagues [1990]. In addition, different formalizations might be needed for different purposes; so the system will be more flexible in adapting to changes in its use over time if formalization is non-destructive.

The second crucial feature of our approach is a “bootstrapping” strategy for supporting formalization. As explained below, we use the knowledge that has already been formalized to suggest possible formalizations of additional input. Thus, the more information is formalized, the more support users get for further formalization.

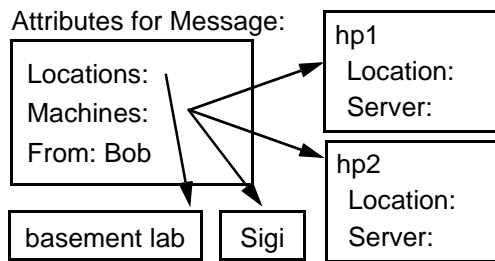


a. Information added in informal representation.

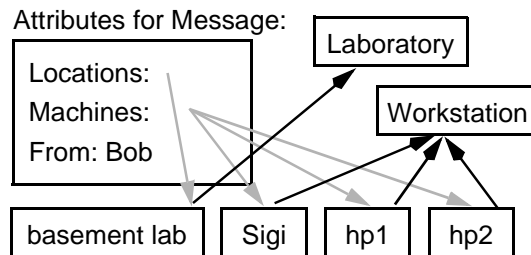
Attributes for Message:



b. Attributes and values attached to message.



c. Textual attributes replaced with associations.



d. Objects are generalized into classes.

Figure 2: Four steps in an example progression from less formal to more formal representation.

Taking the approach of enabling and supporting incremental formalization has a number of implications for system design. First, a system must integrate informal and formal representations seamlessly enough for information to move easily up in its degree of formality. This influences the representations chosen and the interfaces for interacting with the representations. As a practical matter, for the system to easily acquire informal information, it is useful for the system to become a medium of communication between users, rather than just a place to store information or work individually. Finally, active support for formalization

requires a dialog between the user and the system concerning formalization decisions. This implies that the system must include a model of formalization that can be used to provide the user with advice and an appropriate interface to facilitate such dialogs. The following sections describe one system that meets these requirements and discuss our experiences with its use.

### 3. SUBSTRATE FOR INCREMENTAL FORMALIZATION

To support and set the stage for incremental formalization, we have developed the Hyper-Object Substrate (HOS) [Shipman 1993]. We use the term *substrate* because HOS provides a domain-independent framework which can be used to develop domain-oriented applications. We decided to focus our development on the substrate in order to explore incremental formalization in a number of different domains and at different points in the application development process. In the description of HOS functionality, we will use examples from a system to support computer network design called XNetwork.

HOS combines typical characteristics of hypermedia systems, like hyperlinks, navigation and a document metaphor, with characteristics of knowledge-based systems, such as objects with attributes, relations, inheritance, and agents. HOS can be used to develop single-user or multi-user applications which use knowledge-based support to provide users with information based on their current context.

#### 3.1 Basic System Functionality

Systems like JANUS [Fischer et al. 89] and Concorde [Hofmann et al. 90] provide hypermedia and knowledge-based representations but have separate representations with separate interfaces for working with the information. In contrast, HOS uses a single representation for all information, regardless of its level of formalization, to enable incremental formalization without requiring users to shift contexts within the system. In the example of computer network design, the representation of the computer network being designed, the issue-based discussion of this design, and the annotation of different views of the design have a single interface and may be combined within a single view.

HOS represents all information in the substrate (informal text, semi-formal design rationale, and formal knowledge including objects with attached properties) in a single object store. Regardless of their initial status or underlying structure, objects have an extensible set of attributes and relations that are always accessible through a property sheet. In HOS there is no need to know all of the attributes of an object at creation time. Relationships, including inheritance relationships, are allowed between any pair of objects in the system, and can be added, removed, or edited at any time. For example, should the addition of a new high-speed network printer require it, the network administrator interacting within XNetwork can change the underlying representation to take into account the new type of device available on the network.

HOS includes a number of object types: text-graphic, composite, shell, agent, and page. *Text-graphic objects* are objects which contain a drawing method describing the display of that object. The graphical icon for a computer and textual annotations of this computer are text-graphic objects. A network designer reviewing a design can just point and type to annotate the diagram. A *composite object* is a set of other objects which can have interactive and semantic properties as a set. An example of a composite object in XNetwork is the set of computers, network devices, and cables that make up a subnetwork. *Shell objects* provide an interface to information available through the Unix shell. Thus, a shell object can report the current load on a particular compute server, the items in a print queue, or the disk space status of a fileserver.

*Agent objects* have dynamically computed displays and/or actions based on information in the current object store. An agent object within XNetwork might watch for new computers on one subnetwork that use a particular file server as their primary network server and make suggestions that another file server might create a better design. A *page object* is a resizable finite two-dimensional plane which displays any number of the other types of objects. Text-graphic, composite, shell, and agent objects can be moved or copied between pages and can be displayed in multiple pages at once. Page objects in XNetwork are used to

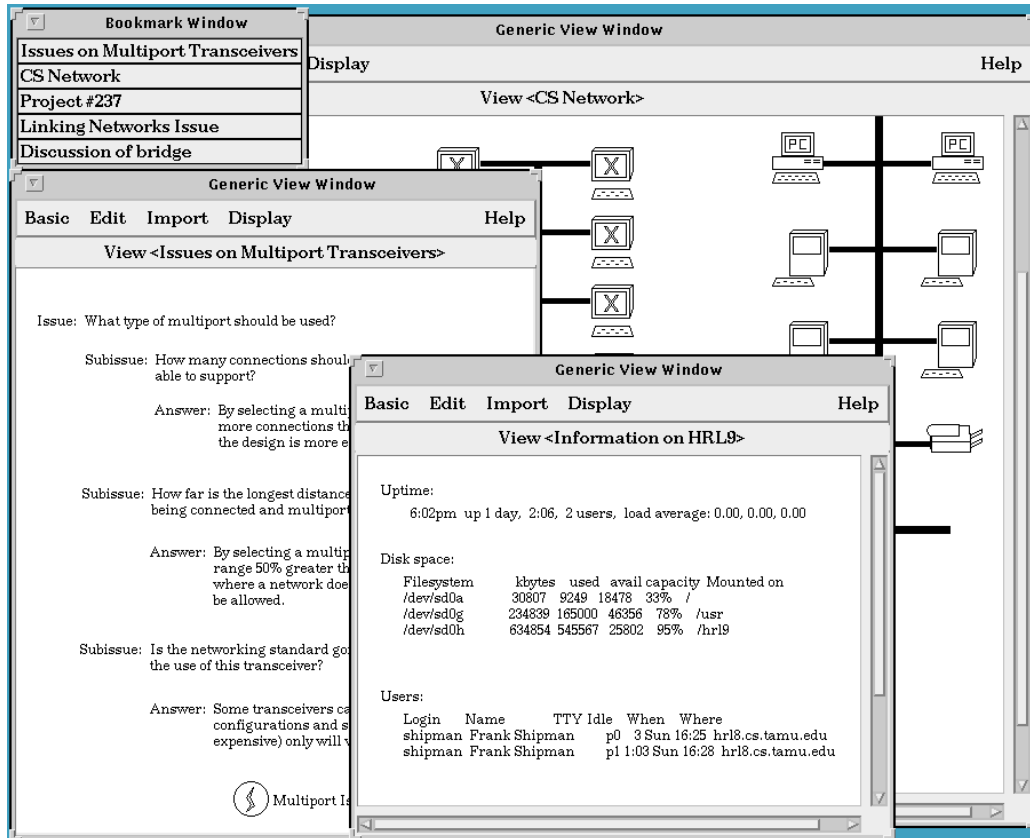


Figure 3: This use of HOS to support network design shows the bookmark list along with three page objects containing text-graphic, shell, and agent objects.

provide perspectives on partial and complete network designs, argumentation-based discussions of particular design issues, and status information of network devices and current network design and administration tasks. Figure 3 shows three page objects presenting information about a particular network.

Every HOS object may also have a navigational link to a HOS page object. As is typical in page-oriented hypermedia, clicking on an object with a navigational link will cause the linked page to be displayed. Another feature of HOS common to many hypermedia systems is that users may choose pages to be added to a list of bookmarks. Bookmarks provide direct access to pages so marked. Figure 3 includes a list of bookmarks in the upper-left corner providing access to pages the designer commonly uses.

### 3.2 Inheritance in HOS

The inheritance model is another example of how HOS was influenced by the goal of enabling end-users access to the formal representations of HOS applications. Prototype inheritance, discussed in detail by Lieberman [1986], removes the distinction between objects acting as classes and objects acting as instances. HOS's variation of prototype inheritance allows inheritance relations between objects to form generic graphs, that is there can be cycles in the inheritance graph. In short, objects can inherit from any other object, without restriction.

This flexibility facilitates multiple objects within XNetwork to represent the same computer subnetwork. A page object containing the detailed design of the subnetwork, a composite object within a page providing a higher level view of the subnet within the context of a departmental network, and a text-graphic object acting as a label for the subnet on a wide-area network design can share attributes and

relations by being part of a cyclic inheritance graph. In a cycle the objects will inherit all attributes and values from the other two objects. When an attribute or relation is changed for any of the three objects, the page object, the composite object, or the text-graphic object, the change takes effect for all three due to the sharing of attributes and relations.

The system only computes inherited attributes and relations when actually needed, allowing users to change inheritance relations during normal system use and for these changes to have immediate effects. Traversal of the inheritance graph includes marking objects as visited in order to detect cycles. Multiple inheritance conflicts are resolved by choosing the ancestor with the shortest path using a breadth-first search traversal pattern. The lazy evaluation of attributes and relations does create more overhead at run-time but we have not found response time to be a problem in any of the systems developed with HOS. Caching attributes and values locally with objects in the object store is a solution that SELF [Ungar, Smith 1987] and other systems have used to reduce the run-time computation incurred by lazy evaluation.

The use of prototype inheritance, rather than the standard class/instance model, removes the requirement that the user learn about knowledge engineering concepts like classes, instances, hierarchies, or directed acyclic graphs which are normally important to the use of inheritance mechanisms. Users may, and from our experience do, create objects which act as classes but they are not constrained to that model. This allows users the flexibility to use existing objects to represent domain concepts, reducing the complexity and cognitive overhead of formalization.

### **3.3 Use of Formal Information**

The focus of HOS and this paper is on enabling and supporting the formalization of declarative information, represented in the text, graphics, attributes, and relations within the object store. To show why such formalization is of practical benefit, HOS includes a simple agent mechanism, similar to that in OVAL [Malone et al. 1992]. Agent objects, also just called agents, provide the ability for HOS applications to make active use of the information represented as attributes and relations to determine when to take some action.

An agent object consists of a trigger, query, and action and are created with the agent editor shown in Figure 4. Triggers determine when agents evaluate their query to recompute their display or to take some other action. To create an agent the user selects from a menu of triggers including “check every action”, “check when displayed”, and “check when requested by user”. The query looks for objects in the system which have certain attributes and relations. The agent defined in Figure 4 looks for objects with Sigi as their file server and that are part of the office tower subnetwork. Query definition in the agent editor is the same as using the HOS property sheet for defining attributes and relations for objects. The objects, if any, returned by the query are then passed on to the action. Actions are chosen from a menu and include creating a display of the objects found, presenting a message to the user, or adding a bookmark. In the example from Figure 4, the user has selected to present a message. Once this is selected the system asks the user to enter the message to be displayed, in this case a message suggesting that a different file server or different subnetwork might be more appropriate (not shown in the figure.)

This variety of triggers and actions enables different agents to have very different interaction styles. They may collect information like Halasz’s virtual structures [1988], or notify users of the occurrence of certain conditions like JANUS critics [Fischer et al. 1989]. Additionally, agents may be more intrusive, presenting a message in a popup window that the user must attend to, or more subtle, adding an item to the bookmark window which may be ignored by the user.

The agent mechanism in HOS provides a very limited inference engine for building knowledge-based applications. The addition of a more expressive inference mechanism, such as the combination of forward and backward chaining in rule-bases with a general purpose programming language found in commercial expert-system shells like the Knowledge-Engineering Environment (KEE), would increase the potential

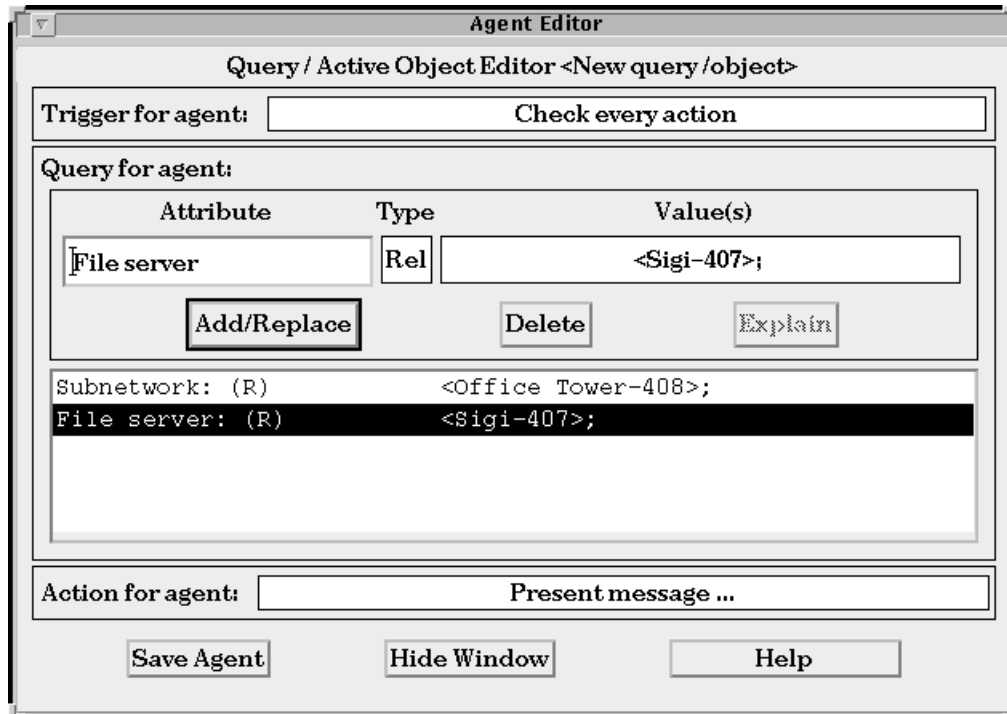


Figure 4: The agent editor in HOS provides users an interface for defining a variety of computational agents which make use of the formal information within the system.

advantage of formalizing information.

#### 4. SUPPORTING INCREMENTAL FORMALIZATION

As the difficulties discussed in section 1 imply, it is not always enough to provide the capability to formalize information. Beyond enabling incremental formalization, HOS includes facilities for importing informal information and actively supporting its subsequent formalization.

##### 4.1 Capturing Communication

For many tasks, the problem is not lack of on-line information but the lack of usefully formalized information. The rapidly increasing volume of electronic mail, USENET News, and Web and other on-line information already provides many people with more information than they can use. Systems that need new formal information can tap into this rich flow and import information for later formalization. Additionally, systems can support what Reeves [1993] calls *embedded communication*, becoming a natural communication channel for collaborating users as their textual communication about an artifact is embedded in a graphical representation of that artifact. HOS uses both techniques to attain informal information.

At its most basic, the user interface for HOS is similar to other page-oriented hypertext systems with an integrated authoring and browsing interface, such as KMS [Akscyn et al. 1988] and the Virtual Notebook System [Shipman et al. 1989]. Unlike the moded interface required for authoring HTML found in Netscape Gold and similar authoring tools, to add a comment to a HOS page, the user simply clicks with the mouse and starts typing.

Users may also import text, electronic mail, and USENET News files into a HOS object store. When importing an electronic mail message or USENET News article HOS parses the header and adds the information as attributes to the newly created object. Figure 5 shows an electronic mail message

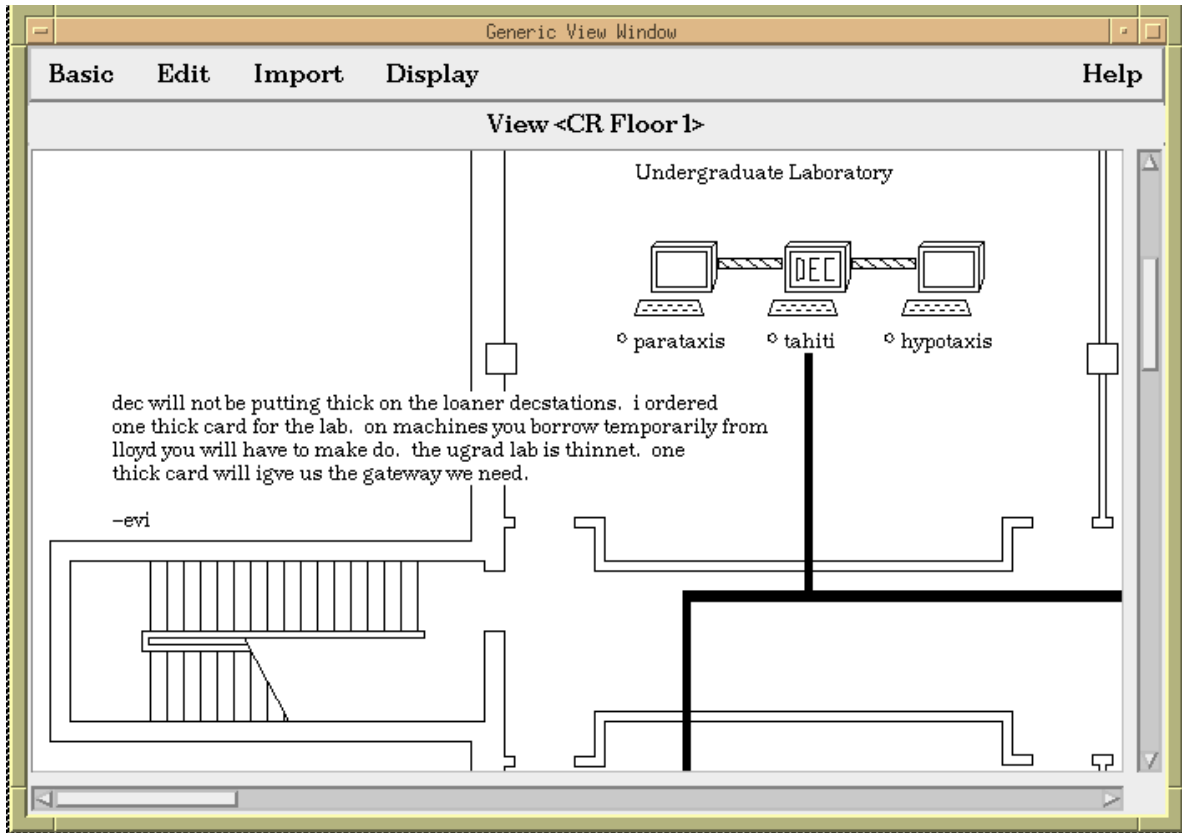


Figure 5: Electronic mail message imported next to the network being discussed

concerning network design in a page containing the graphically represented design being discussed. The graphical objects contain the formal information about the network devices, such as the configuration information for workstations and the properties of the network cables. It is the integration of the informal and formal knowledge in a single representation, as opposed to JANUS's separation of textual information from the domain knowledge [Fischer et al. 1989], that enables embedded communication to lead to incremental formalization [Reeves, Shipman 1992].

#### 4.2 Suggesting Formalizations of Informal Information

We have explored a simple mechanism to support formalization that makes suggestions for formalizations based on textual patterns within the information already in the system. This mechanism suggests the addition or modification of attributes and relations. Suggestions are based on the current state of the information space, thus accepting the suggestion may lead to further suggestions.

The text-analysis algorithm for determining suggestions is deliberately simple, using a type of string search rather than natural language processing techniques. Figure 6 diagrams this process for one of the suggestions based on the text in Figure 5. The system creates a lexicon by collecting the names and synonyms of the objects in the information space. These are currently explicitly defined by the user. When the property sheet for an object is opened, the system looks for lexicon items within both the object's textual content and textual values of attributes. For example, in Figure 6 the string "lloyd" is found to be in the lexicon. When a reference is found, a rule base is used to determine what attribute or relation is suggested to the user based on characteristics of the object possibly being referenced. In the example, the object with the synonym "Lloyd" inherits the "type: person" attribute indicating the object represents a person. The rule from Figure 6 indicates that a potential reference to a person leads to a suggestion that the

object referenced be added to the “People involved” relation for the textual object. The user interacts with the information in the object store, either directly or indirectly changing the elements in the lexicon and the object hierarchies. The rule-base used to map potential references to suggestions for attributes and relations cannot currently be changed by the user.

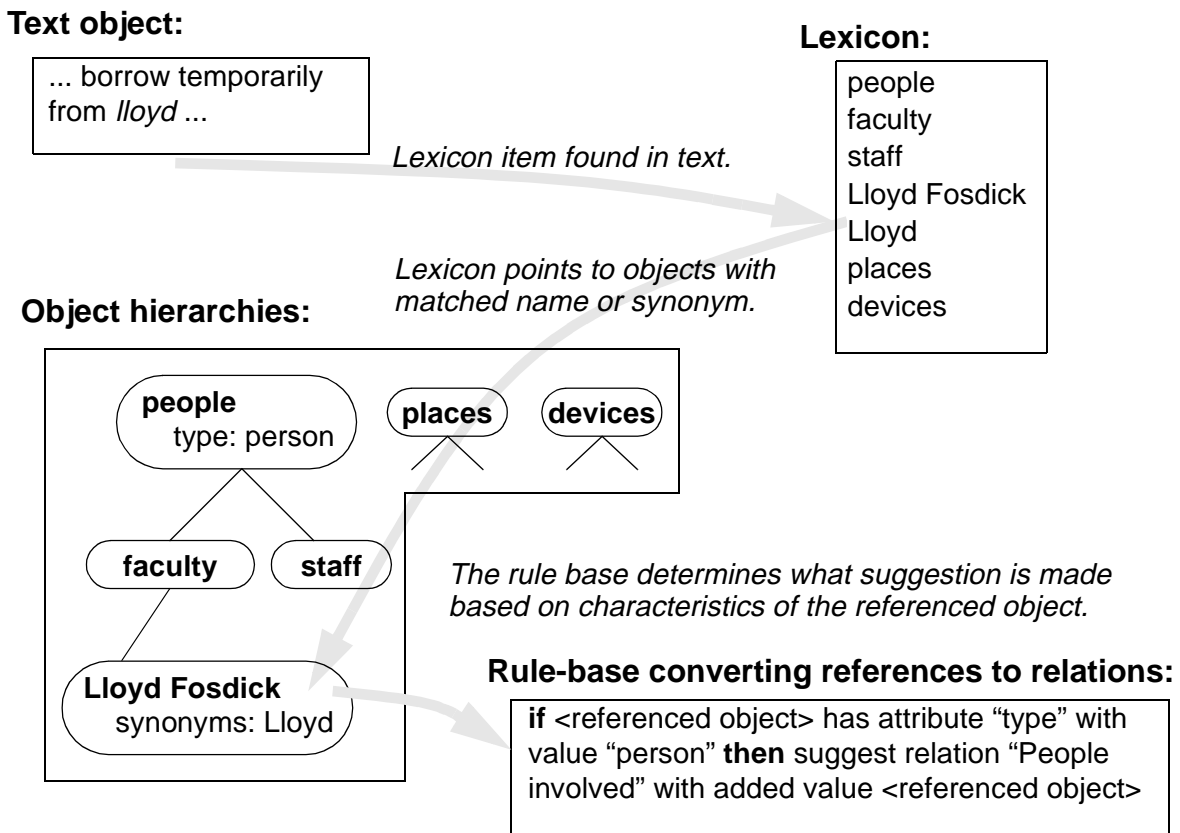


Figure 6: Diagram of the steps involved in determining what formalizations are suggested by HOS.

Figure 7 shows the property sheet for the electronic mail message from Figure 5 just after it was imported. In this case four suggestions (the top four attributes listed) have been provided. Three are based on possible references within the body of the text to people, places, and devices in the object store and are produced as explained above. The fourth suggestion, for the “From” attribute, is a modification of an attribute created by parsing the electronic mail header. In this case the system suggests replacing the textual value “Evi Nemeth <evi>” with a relationship to the object named “Evi Nemeth”.

Suggestions can be accepted as is, modified and accepted, deleted, or just ignored. If the attribute being edited is one suggested by HOS the “Explain” button becomes active, as it is in Figure 7. HOS provides an explanation of why each suggestion was made, providing users with rationale about the formalization process and about the specific formalization being suggested. These explanations are generated by filling in information from the lexicon item, the rule base, and the object believed to be referenced into a textual template.

The suggestion mechanism in HOS is domain-independent but requires domain-dependent information to produce suggestions. The “people involved”, “places involved”, and “devices involved” suggestions in Figure 7 require the important people, places and things in a domain to be objects in the system. In this

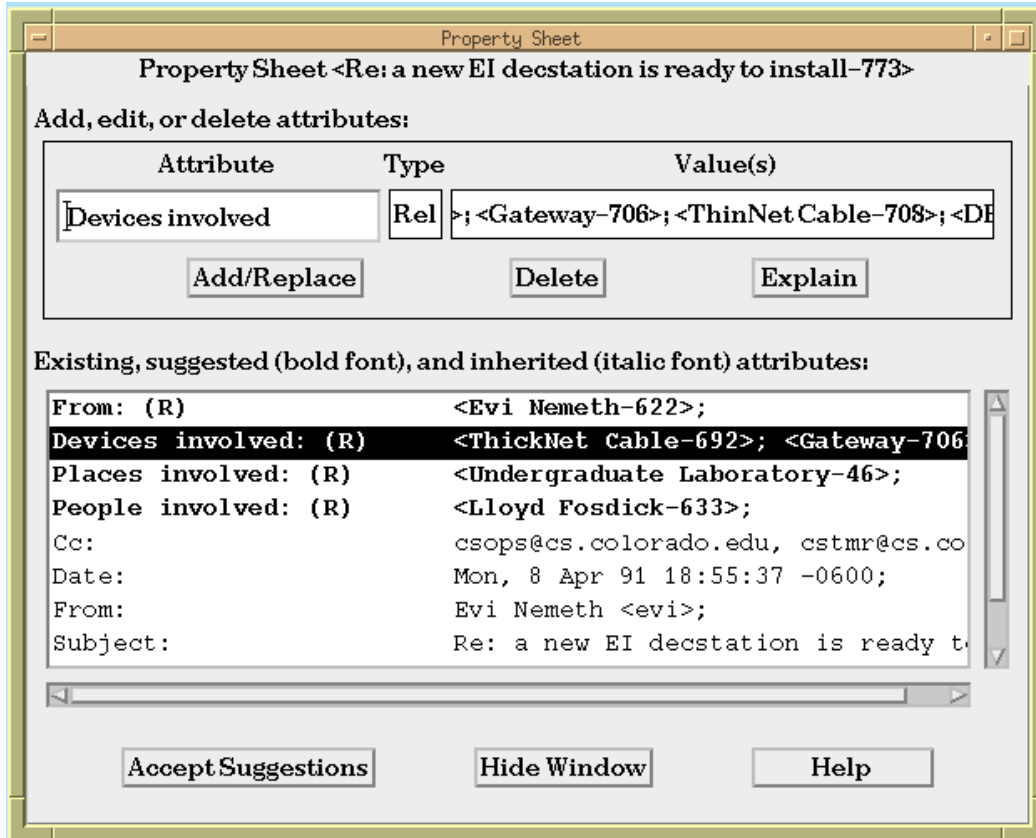


Figure 7: Suggested attributes in a property sheet

case the “things” suggestion has been specialized to devices for the domain of network design. People, places, and things represent the who, where, and what involved in a piece of text. These are likely candidates for formalization across domains. Likewise, the suggestion that textual attribute values be changed to relations to other objects (such as the “From” relation in Figure 7) uses the same type of domain-specific information and the same domain-independent search algorithm.

## 5. APPLICATIONS

HOS has been used to create a variety of domain-oriented applications and information spaces. Additionally, it has been applied to more traditional hypermedia tasks including trip reports, discussions of current literature, and the development of the outline for a dissertation. Experience with these informal tasks suggests that HOS’s standard hypermedia authoring/browsing abilities are not hindered by functionality concerned with more formal representations. As hypertext authoring is performed through the direct manipulation of objects in a WYSIWYG browser/editor, the users did not need to use the property sheet or other interface components associated with the system’s formal representations. The knowledge-based system functionality is invisible to users who do not need it.

Enabling users to easily author less formal information is only one (necessary) part of incremental formalization. We will now describe our experiences with the use of HOS for applications combining knowledge-based and hypermedia functionality. First, we discuss experiences with the use of HOS for graduate-level class projects in the domains of archeological site analysis and neuroscience education. Following this we will further describe its use in the creation of XNetwork, a knowledge-based design environment supporting collaborative network design. The formalized information acquired in these examples, in the form of attributes and relations, is used to provide greater domain-oriented or task-

oriented support.

### 5.1 Semester-Long Graduate Student Projects

HOS was used for two semester projects in a graduate-level knowledge systems class. Each project was the work of a single student and took place over a period of two to three months. The goal of the class projects was to create a knowledge-based system for a domain and task of the student's choice.

One class project used HOS to build the Archeological Site Analysis Environment (ASAE), a tool to aid teams of experts in different topics sharing and analyzing information concerning the "dig" site. The purpose of ASAE, as described in the student's project report, is "to handle the information overload, to link the archeological team members, and to make historical and scientific background knowledge more accessible and useful."

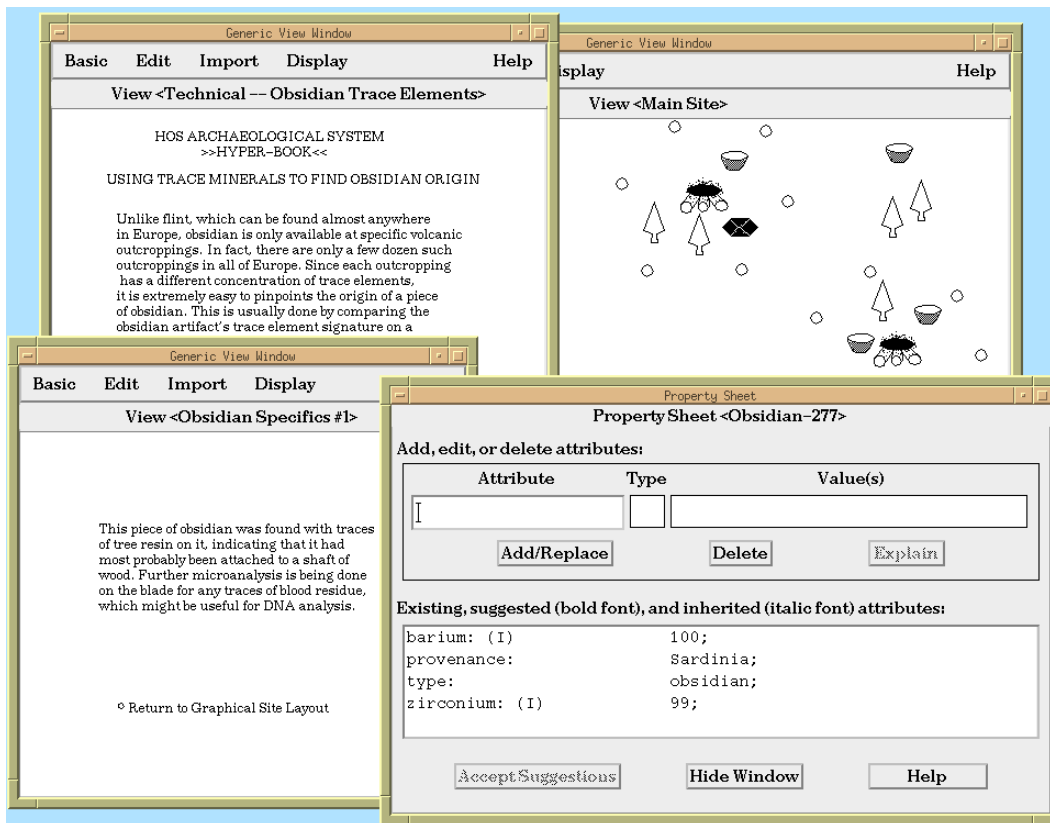


Figure 8: ASAE combines specifications and graphical layout of objects found at the site with communication about both specific objects and archeology in general.

To try to meet these goals, ASAE combines formal and informal information about archeology in general as well as information about the specific site. The top right of Figure 8 shows the HOS page object representing the dig site. Artifacts found at the site are represented with graphical icons indicating the type of object placed in the general location the object was found. Artifacts have their size and composition represented as attributes for use by knowledge-based decision support. HOS agents are used in ASAE to act as "advertisers" of some information and are intended to facilitate communication among group members. ASAE was developed with HOS's existing functionality except for the programming of numerical methods to classify artifacts and deduce information about the site based on the chemical composition and geometric arrangement of the artifacts. This use of domain-specific programming exemplifies the need for domain-oriented programming languages [Eisenberg, Fischer 1994], a feature not

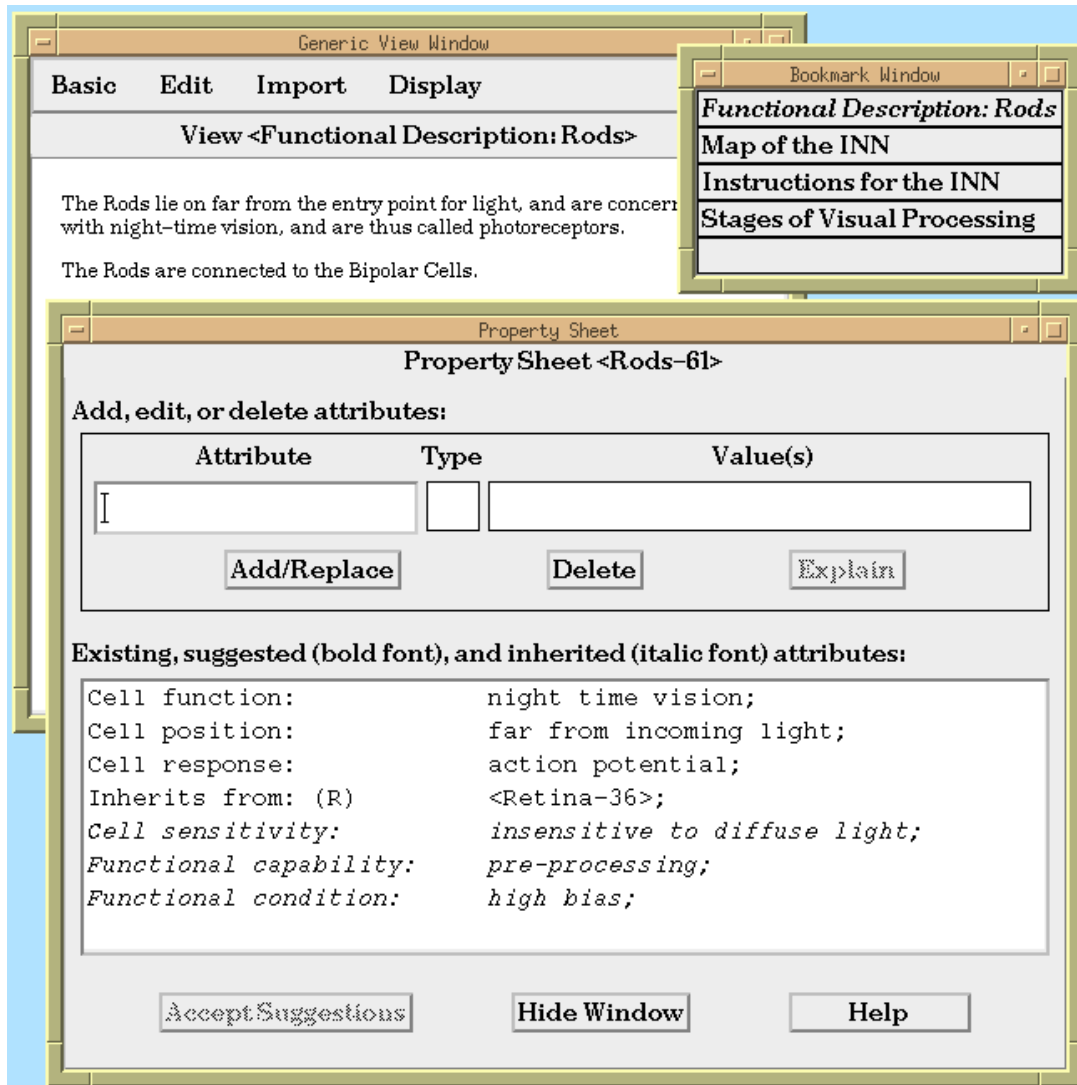


Figure 9: The Interactive Neurosciences Notebook (INN) showing the attributes and relations of an object in the completed version of INN

currently available within HOS.

The other knowledge systems class project using HOS developed the Interactive Neuroscience Notebook (INN). The purpose of INN is to provide an environment for student neuroscientists to collect and organize information within the context of a pre-authored framework and set of information. The resulting role of INN can be seen as a combination of textbook and personal notebook.

Like ASAE, INN is composed of both informally and formally represented information. Information from an introductory textbook on the visual system was placed in pages of textual information. Some of the objects on these pages represent concepts or objects in the domain, such as particular types of cells. These objects include formally represented information, mostly attributes describing features of the object. Figure 9 shows the property sheet for an object in INN. A number of HOS agents are used to suggest interesting or related information based on the modifications to the notebook performed by the student. The top item in the bookmark window in Figure 9 is in italics to show it has been added by the system rather than by the user. The development of INN was done solely within HOS, without the need for programming. The next sections will discuss the incremental process used to develop these applications.

## 5.2 Information Space Evolution

Close to fifty intermediate states of each project were recorded and a log of attribute modifications was collected in order to look for patterns of evolution. Both the snapshots and the logs provide some noticeable patterns of growth in the projects. In particular, both projects had two or three periods of rapid growth, or “jumps”, in the content of their information spaces, the rest of the time experiencing slow growth.

Was the occurrence of these periods of rapid growth something other than just the students working harder on their projects? To determine if these jumps were significant to the support of incremental formalization, we looked at what information changed during these jumps and used our weekly discussions with the students to consider motivation. These large changes often occurred when the students redefined an aspect of their task or started a new phase in their projects. Like many real-world tasks, the class projects were created without any detailed specification of how the final system would work. The goals were set in the initial report, but continued to be refined throughout the projects. This co-evolution of problem definition with problem solution is one characteristic of what Rittel [1984] describes as “wicked problems”, which include much of the work done by professional designers and analysts.

The addition or modification of attributes for objects already in the system accompanied several of these information “jumps”. One of the larger jumps was the addition of agents to volunteer information in INN. This jump included the addition of agent objects, but also included a large increase in the number of attributes within the information space as formal information that could be used by the agents was added to the previously informal (textual) objects.

This general observation supports the view that flexible representations, where formalization can be demand-driven, enable the evolution of an information space to match evolving user goals. The following section describes a particular example of this process and discusses some of the issues resulting from this practice.

## 5.3 Example of Incremental Formalization

The development of INN provided examples of incremental formalization in the form of paragraphs of text that came to be used as concept objects. In particular, text objects concerning a domain concept were sometimes used to represent that concept, meaning an object would be named and have attributes attached appropriate to the concept rather than attributes that described the discussion that made up the object’s display. For example, the object representing the concept of “Rods” in the INN is the top two lines of text in the “Functional Description: Rods” page object, shown in the upper left of Figure 9. This object has the textual display “The Rods lie on far from the entry point for light, and are concerned with night-time vision, and are thus called photoreceptors.” Early in the project, this object was purely textual, that is there were no attributes or relations attached to the object.

Figure 10 shows the attributes for this object but in the middle of the semester. At this point the object is being used to represent the domain concept of “Rods” but the concept has not been developed within the context of other concepts. Figure 9 shows that by the completion of the project the object was placed in an inheritance relation to a generalization of cell types in the retina and thereby had a number of other attributes. This object changed from a piece of text on a page to an object with attributes to an object taking part in an inheritance relation. The student followed the process of incremental formalization as described in Section 2 of the paper but left the textual object to be both the original material from the textbook and a representation of a specific domain concept.

This example also shows a difficulty with HOS’s support for “in place” formalization. The use of paragraphs of text to represent domain concepts results in these objects being difficult to distinguish from other textual objects. In order to locate objects representing domain concepts users must check each object or use queries to locate objects with certain attributes.

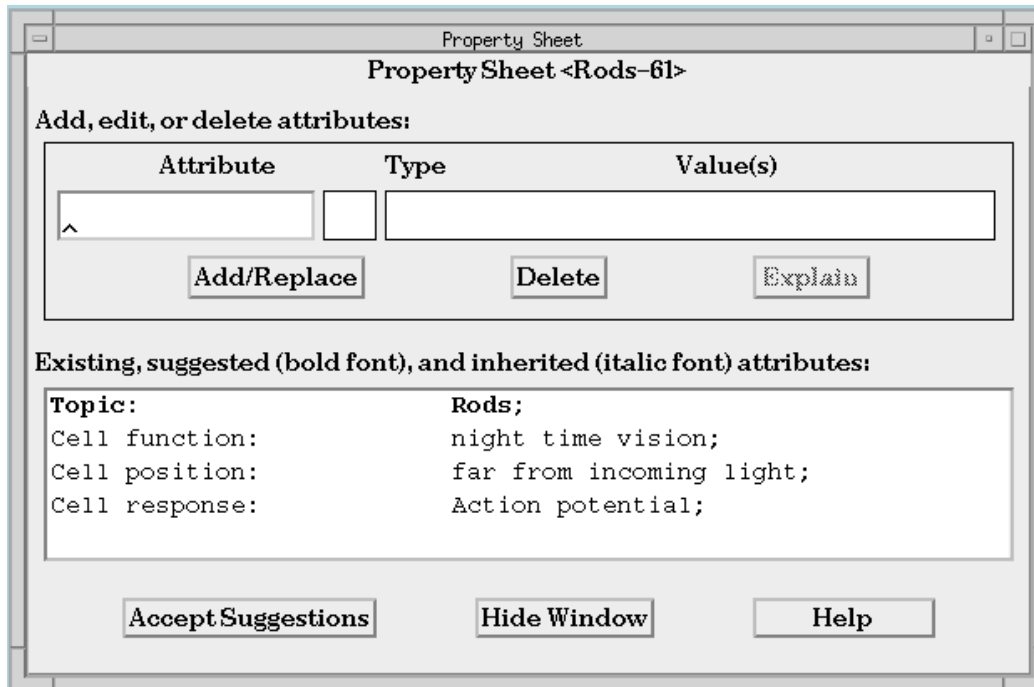


Figure 10: Properties of object early in the development of INN

#### 5.4 Experience with Suggestions in Class Projects

Because the suggestion mechanism requires named objects (the lexicon used is created from these names), it was assumed that it would not be of use during the initial development of new applications. This being the case, HOS's suggestion of formalizations (described in section 4.2) was not mentioned to the students. About midway through the projects, the suggestion mechanism did become noticed by one of the students. In one of the weekly meetings, a student asked how the system "knew" what he was doing. This question occurred because the student had named objects which were then used to create the lexicon by which the suggestion mechanism works. The student started to see suggestions based on the object names, but did not know why or how they were created. One example of a suggestion in the case of the INN is the attribute "Topic" shown in Figure 10. Another example which appeared was the suggestion for the topic of the rest of the paragraphs on the same page (which form one HOS object) to be "Bipolar Cells", "Horizontal Cells", "Cones", and "Ganglion Cells".

The experience during the class projects showed that the suggestion mechanism works, that it will frequently make sensible suggestions even given only a small set of named objects. This occurrence of suggestions in small, largely informal information spaces is evidence that the suggestion mechanism can help bootstrap the information space. The quality of the suggestions was not evaluated during these projects. Most of the suggestions seemed reasonable to the students, although sometimes, as was the case with the suggestion in Figure 10, of questionable utility. In the end, only a couple of suggestions were accepted during the projects. The following section about the creation of XNetwork will discuss experiences with the suggestion mechanism in more detail.

#### 5.5 Creation of XNetwork

HOS has been applied in the creation of XNetwork, a prototype domain-oriented design environment to support the collaborative long-term design and administration of computer networks initially described in [Fischer et al. 1992]. By acting not only as a tool for specifying the network design, but also as a medium for network designers to communicate, XNetwork supports the process of designing a computer network:

recording design discussions; storing past, present, and future designs; and advising designers when design rules are perceived to be broken.

The development of XNetwork in HOS, driven by a consultation with professional network designers over a period of close to two years, provides an example of the evolutionary development of a knowledge-based system. We chose computer network design as a domain to study incremental formalization because this domain's rapid change guarantees a continual need for both the designers and the system to learn about new technologies and devices.

The initial creation of XNetwork showed that the suggestion mechanism is useful beyond the formalization of specific information; they also act as shortcuts in adding information. For example, while building the inheritance hierarchy of device types, HOS frequently suggested "device involved" relations to the object planned to be the recipient of the inheritance relation. By changing this suggested relation to an inheritance relation, the number of user actions required to create inheritance links was substantially less than required when no such suggestion was available for modification. While such suggestions did not influence what was formalized, they did speed up the process of adding these relations.

The development of XNetwork also showed that suggestions could aid the user in finding mistakes or in learning about relevant information within the information space. We found that users begin to develop expectations of when and what types of suggestions will appear. When the system does not meet the user's expectations, the suggestion (or lack thereof) may trigger a realization that the information space is either inaccurate or incomplete. Unexpected suggestions can also lead to the discovery of information previously unknown by the user. For example, during the creation of XNetwork, occasionally suggestions were made that were based on information that was imported automatically from other on-line information sources. These suggestions functioned as notifications of this other information's existence.

A final observation about the suggestion mechanism in HOS concerns the inclusion of suggestion mechanisms within the HOS interface. By only suggesting new formalizations when the user opens a property sheet, indicating their interest in formal information, the system avoided problems associated with system-initiated dialogs such as interrupting the user's work. As users could (and did) readily ignore suggestions, there was no cost to the user from the suggestion mechanism other than potentially having to scroll through more attributes in the property sheet.

These experiences with suggesting formalizations indicate potential and warrant continued investigation into such mechanisms. Improving the suggestions by using more sophisticated natural language and message understanding algorithms is one direction for future work. Also, currently only textual information is used to make suggestions. Using additional information, such as the layout of objects in the views, or similarity in graphical display, might both improve the current mechanism and provide the user access to transient information structures as is the case in the work on VIKI [Shipman et al. 1995].

## **6. RELATED WORK**

Related attempts to overcome the difficulties of formalization fall into two main categories of research: systems combining informal and formal representations and systems that automatically or semi-automatically generate formal representations based on informal information.

### **6.1 Systems Integrating Informal and Formal Representations**

A number of researchers have built systems combining formal knowledge representations and hypermedia systems. MacWeb [Nanard, Nanard 1991] uses a model of semi-structured typing to integrate such representations. Kaindl and Snaprud also describe an integration of hypermedia and structured object representations and its use in knowledge acquisition [1991]. PHIDIAS [McCall et al. 1994] uses virtual (computed) links and nodes defined in an applicative language. An example of the need for less formal representations is Schwabe and colleagues' [1990] decision to add a hypermedia component to a tool for

creating Prolog encodings of engineering knowledge. One of the first hypermedia systems to include inheritance and methods was SPRINT [Carlson, Ram 1990]. These systems combine hypertext and more formal representations, but do not support the process of formalization.

The system CONCORDE [Hofmann et al. 1990] is a hypermedia system designed to support the process of knowledge acquisition for expert systems. CONCORDE is designed for a professional knowledge engineer structuring information and limits its support to providing an information space which includes both the formalized knowledge for the expert system and the sources of the formalized knowledge, such as comments from domain experts and text from books. CONCORDE lacks any active support for formalization and is not meant to support knowledge-base evolution during use of the resulting system.

Hoopertext [Berlin, O'Day 1990] is a platform for building collaborative applications and, like HOS, uses an object-oriented knowledge representation to provide both the development environment for the application and the underlying interface for the eventual users of the application. Hoopertext leaves the issue of how users will work with the formal and informal information to be handled by the application.

There has also been interest in combining formal and informal representations within the knowledge representation and knowledge acquisition communities [Mundie, Shultis 1991]. One system that allows informal information in knowledge bases is CODE4 [Lethbridge, Skuce 1992]. CODE4's representation allows the inclusion of everything from "sloppy English to formal logic". Users interact through a set of browsers with popup menus and dialogs. The browsers display relationships between concept objects and can be used in outline or graphical modes.

Shipman and Marshall [1993], reporting experiences with the use of the Virtual Notebook System, NoteCards, Aquanet, and other systems, describe the difficulties users have working with formal representations and the need for users to determine the level of formality for their particular task. Haake and colleagues' [1994] requirements for more flexible hypermedia systems for use in meeting rooms further support these findings.

## **6.2 Systems Supporting or Automating Formalization**

Work related to actively supporting formalization includes mechanisms for automatically generating hypermedia links, knowledge discovery in databases, and the discovery and use of implicit structure.

Not all hypermedia systems require links to be created by users. Some systems provide automatically-generated links between pieces of information that seem related, such as a link between a word and its entry in a dictionary or other previously compiled reference source. In the PC-based system SmarText the system provides connections between the use of the same word or phrase, providing automated links to the "next/previous occurrence" of a term without requiring the standard dialog box for search mechanisms. Bernstein's link apprentice [1990] evaluates the similarity of pieces of text based on the occurrence of words and word-parts. This apprentice does not automatically generate links but, like HOS's support for formalization, makes suggestions to the user, leaving it up to the user to accept the suggestion or not. While these systems focus on the creation of navigational links, HOS emphasizes the formalization of domain knowledge.

The area of knowledge discovery in databases has goals related to the discovery of domain knowledge. Frawley and colleagues [1992] describe knowledge discovery as "the nontrivial extraction of implicit, unknown, and potentially useful information from data." Despite the similarity of this description to the HOS suggestion mechanism, knowledge discovery in databases differs significantly with regard to the problem being addressed. In the case of knowledge discovery the goal is to discover knowledge that the user does not have, such as the purchasing habits of the readership of a particular magazine, from a mass of data. While similar algorithms may be applicable, the goal of formalization suggestions is to aid the user in expressing their own knowledge.

A number systems have been built to reduce the need level of formality required from users in the performance of particular tasks, such as the spatial analysis of information [Marshall, Shipman 1993; Shipman et al. 1995], the sketching phase of early design [Saund, Moran 1994; Gross 1996] and the interactions that occur in a meeting [Haake et al. 1994; Moran et al. 1995]. Many of these systems use the recognition of patterns in pen strokes or spatial layout to aid the user's manipulation of perceptually apparent but not explicitly stated structures. These systems support the user at a different point on the formality scale shown in Figure 1 — their goal is to transparently support the user moving from pen strokes to objects and from object layout to lists and piles, not to support the modeling of a domain for a knowledge-based system.

## 7. CONCLUSIONS

Computers require formally represented information to perform computations that support users; yet users who have needed such support have often proved to be unable or unwilling to formalize it due to issues including premature structuring, cognitive overhead, and tacit knowledge. To address this problem, we have introduced an approach called *incremental formalization*, in which users provide information informally and the system aids them in formalizing it over time.

Our prototype, HOS, supports incremental formalization through the integration of the capabilities of hypermedia and knowledge engineering systems. By allowing users to choose the degree of formality for entering information, HOS reduces the up-front costs for users adding information. In particular, knowledge added in a less formal representation has the potential to evolve into a more formal representation “in place” — that is, without needing to be removed and re-added to the system.

HOS supports the importing of information from text, electronic mail, and USENET News files and actively supports the in-place evolution through the use of recognized references or patterns within less formally represented information to suggest possible new or modified formalizations.

HOS has been used to create a number of domain-oriented systems, the largest which is an environment to support the collaborative design of computer networks called XNetwork. The suggestion mechanism in HOS was used during the creation of XNetwork to provide shortcuts for the creation of formal representations and to help inform the user of the existence of information.

Two more domain-oriented systems were created with HOS as projects in a graduate class on knowledge systems. One of these systems supports the recording, sharing, and analysis of archeological site information. The other class project supports students learning about neuroscience by providing a seeded interactive notebook that would volunteer information as it was being personalized by the student.

During the development of these systems, the need to support the evolution of formal representations was reiterated. As the students' goals and understanding of the domains changed over the course of the semester, so did the structure required to be formally represented. Sometimes this meant adding new formalisms, but also this occasionally resulted in the removal or modification of previous formalisms.

The experiences with HOS provide evidence that some of the problems that users have with formalization were solved or mitigated with our approach to incremental formalization:

- premature structuring — solved by not forcing users to initially enter formal information;
- cognitive overhead — mitigated due to lower overhead for initial entry and demand-driven effort required for later formalization; and
- tacit knowledge — mitigated by suggestion mechanisms which promote conscious understanding of previously tacit knowledge.

This project has looked at the incremental formalization of declarative information in the context of domain-oriented systems. In addition to the facts and opinions recorded in objects, attributes, and relations,

there is additional domain knowledge encoded in agents, rules, and program code. Enabling and supporting the incremental development of this procedural information will require different tools, and is an area for future work. Additional future work could augment the mechanisms reported here to use the non-textual information in the system as additional cues for providing suggestions and to work with alternative knowledge representation schemes.

## **ACKNOWLEDGEMENTS**

This research was supported in part by grant No. IIS-9734167 from the National Science Foundation.

## **BIOGRAPHIES**

Frank M. Shipman III is an Assistant Professor in the Department of Computer Science and Center for the Study of Digital Libraries at Texas A&M University. He has been pursuing research in the areas of hypermedia, computer-supported cooperative work, and intelligent user interfaces since 1987. Dr. Shipman's doctoral work at the University of Colorado and subsequent work at Xerox PARC and Texas A&M has investigated combining informal and formal representations in interfaces and methods for supporting incremental formalization. He manages a number of on-going research projects in the areas of spatial hypertext, intelligent interfaces, and computers and education.

Raymond McCall is an Associate Professor in the Department of Planning and Design at the Boulder campus of the University of Colorado. Since the mid-1970s his research has dealt exclusively with the creation of hypertext systems to support design rationale capture and delivery for designers. His work currently focuses on the integration of hypermedia, computer-aided-design graphics, and knowledge-based computation within a hypermedia framework. The main application domain for this work has been the design of space-based habitats by NASA and its contractors. Professor McCall has a Ph.D. in architectural design theory and methods from the University of California, Berkeley.

## **REFERENCES**

- Akscyn, R.M., McCracken, D.L., & Yoder, E.A. (1988). KMS: A Distributed Hypermedia System for Managing Knowledge in Organizations. *Communications of the ACM*, 31, 7, pp. 820-835.
- Berlin, L., & O'Day, V. (1990). Platform and Application Issues in Multi-User Hypertext. *Multi-User Interfaces and Applications*, S. Gibbs, A. Verriijn, Eds., Amsterdam: North-Holland, 1990, pp. 293-309.
- Bernstein, M. (1990). An Apprentice That Discovers Hypertext Links. *Proceedings of the European Conference on Hypertext (ECHT'90)*, pp. 212-223.
- Carlson, D., & Ram, S. (1990). HyperIntelligence: The Next Frontier. *Communications of the ACM* 33, 3, pp. 311-321.
- Chomsky, N. (1956). Three Models for the Description of Language. *IRE Transactions on Information Theory*, 2, 3, pp.113-124.
- Conklin, E.J., & Burgess Yakemovic, K.C. (1991). A Process-Oriented Approach to Design Rationale. *Human Computer Interaction*, 6, 3-4, pp. 357-391.
- Eisenberg, M. & Fischer, G. (1994). "Programmable Design Environments: Integrating End-User Programming with Domain-Oriented Assistance," *Human Factors in Computing Systems, CHI '94 Conference Proceedings*, pp. 431-437.
- Fischer, G., Grudin, J., Lemke, A., McCall, R., Ostwald, J., Reeves, B., & Shipman, F. (1992). Supporting Indirect, Collaborative Design with Integrated Knowledge-Based Design Environments. *Human Computer Interaction*, 7, 3, pp. 281-314.

- Fischer, G., Lemke, A.C., McCall, R., & Morch, A. (1991). Making Argumentation Serve Design. *Human Computer Interaction*, 6, 3-4, pp. 393-419.
- Fischer, G., McCall, R., & Morch, A. (1989). "JANUS: Integrating Hypertext with a Knowledge-Based Design Environment," *Proceedings of Hypertext '89*, pp. 105-117.
- Frawley, W.J., Piatetsky-Shapiro, G., & Matheus, C.J. (1992). Knowledge Discovery in Databases: An Overview. *AI Magazine*, 13, 3, pp. 57-70.
- Girgensohn, A. (1992). *End-User Modifiability in Knowledge-Based Design Environments*. Ph.D. Dissertation., Department of Computer Science, University of Colorado, Boulder, CO.
- Gross, M.D. (1996). The Electronic Cocktail Napkin - computer support for working with diagrams. *Design Studies*, 17, 1, pp. 53-69.
- Grudin, J. (1994). "Groupware and Social Dynamics: Eight Challenges for Developers," *Communications of the ACM*, 37, 1, pp. 92-105.
- Haake, J., Streitz, N., & Neuwirth, C. (1994). "Coexistence and Transformation of Informal and Formal Structures: Requirements for More Flexible Hypermedia Systems," *Proceedings of European Conference on Hypertext (ECHT '94)*, pp. 1-12.
- Halasz, F. (1988). "Reflections on NoteCards: Seven Issues for the Next Generation of Hypermedia Systems," *Communications of the ACM*, 31, 7, pp. 836-852.
- Hofmann, M., Schreiweis, U., & Langendoerfer, H. (1990). "An Integrated Approach of Knowledge Acquisition by the Hypertext System CONCORDE," *Hypertext: Concepts Systems and Applications*, A. Rizk, N. Streitz, and J. Andre, Eds., Cambridge University Press, UK, pp. 166-179.
- Kaindl, H., & Snaprud, M. (1991). Hypertext and Structured Object Representation: A Unifying View. *Proceedings of Hypertext '91*, pp. 345-358.
- Lethbridge, T.C., & Skuce, D. (1992). Informality in Knowledge Exchange. *Proceedings of the AAAI Workshop on Knowledge Representation Aspects of Knowledge Acquisition*.
- Lieberman, H. (1986). "Using Prototypical Objects to Implement Shared Behavior in Object-Oriented Systems," *OOPSLA 1986 Conference Proceedings*, pp. 214-223.
- Malone, T.W., Lai, K.Y., & Fry, C. (1992). "Experiments with Oval: A Radically Tailorable Tool for Cooperative Work," *Proceedings of the Conference on Computer Supported Cooperative Work (CSCW '92)*, pp. 289-297
- Marshall, C.C., & Rogers, R.A. (1992). "Two Years before the Mist: Experiences with Aquanet," *Proceedings of European Conference on Hypertext (ECHT '92)*, pp. 53-62.
- Marshall, C.C., Shipman, F.M. (1993). "Searching for the Missing Link: Discovering Implicit Structure in Spatial Hypertext." *Proceedings of Hypertext '93*, pp. 217-230.
- Marshall, C.C., Shipman, F.M., & Coombs, J.H. (1994). "VIKI: Spatial Hypertext Supporting Emergent Structure," *Proceedings of European Conference on Hypertext (ECHT '94)*, pp. 13-23.
- McCall, R., Mistrik, I., & Schuler, W. (1981). "An Integrated Information and Communication System for Problem Solving: Basic Concepts," *Data for Science and Technology: Proceedings of the Seventh International CODATA Conference*, pp. 107-115.

- McCall, R., Bennett, P., & Johnson, E. (1994). "An Overview of the PHIDIAS II HyperCAD System," *Reconnecting: Proceeding of ACADIA '94*, Association for Computer Aided Design in Architecture, St. Louis, Missouri, pp. 63-74.
- Moran, T.P., Chiu, P., van Melle, W., & Kurtenbach, G. (1995). "Implicit Structures for Pen-Based Systems Within a Freeform Interaction Paradigm," *Human Factors in Computing Systems, CHI '95 Conference Proceedings*, pp. 487-494.
- Mundie, D.A., & Shultis, J.C. (Eds.). (1991). *Proceedings of the Workshop on Informal Computing*.
- Nanard, J., & Nanard, M. (1991). "Using Structured Types to incorporate Knowledge in Hypertext," *Proceedings of Hypertext '91*, pp. 329-343.
- Polanyi, M. (1966). *The Tacit Dimension*, Doubleday, Garden City, NY.
- Reeves, B.N. (1993). *Supporting Collaborative Design by Embedding Communication and History in Design Artifacts*. Ph.D. Dissertation, Department of Computer Science, University of Colorado, Boulder, CO.
- Reeves, B.N. & Shipman, F.M. (1992). "Supporting Communication between Designers with Artifact-Centered Evolving Information Spaces," *Proceedings of the Conference on Computer Supported Cooperative Work*, 1992, pp. 394-401.
- Rittel, H. (1984). "Second Generation Design Methods," *Developments in Design Methodology*, N. Cross, Ed., John Wiley & Sons, New York, pp. 317-327.
- Saund, E. & Moran, T. (1994). "A Perceptually-Supported Sketch Editor," *Proceedings of ACM Symposium on User Interface Software and Technology*, pp. 175-184.
- Schoen, D. (1983). *The Reflective Practitioner: How Professionals Think in Action*. Basic Books, New York.
- Schwabe, D., Feijo, B., & Krause, W. (1990). "Intelligent Hypertext for Normative Knowledge in Engineering," *Hypertext: Concepts Systems and Applications*, A. Rizk, N. Streitz, and J. Andre', Eds. Cambridge University Press, Cambridge, UK, pp. 123-136.
- Shipman, F.M., Chaney, R.J., & Gorry, G.A. (1989). "Distributed Hypertext for Collaborative Research: The Virtual Notebook System," *Proceedings of Hypertext '89*, pp. 129-135.
- Shipman, F.M. (1993). *Supporting Knowledge-Base Evolution with Incremental Formalization*. Ph.D. Dissertation, Department of Computer Science, University of Colorado, Boulder, CO.
- Shipman, F.M., & Marshall, C.C. (1993). *Formality Considered Harmful: Experiences, Emerging Themes, and Directions*. Technical Report CU-CS-648-93, Department of Computer Science, University of Colorado, Boulder, 1993.
- Shipman, F.M., Marshall, C.C., & Moran, T.P. (1995). "Finding and Using Implicit Structure in Human-Organized Layouts of Information," *Human Factors in Computing Systems, CHI '95 Conference Proceedings*, pp. 346-353.
- Shipman, F.M., & McCall, R.J. (1994). "Supporting Knowledge-Base Evolution with Incremental Formalization," *Human Factors in Computing Systems, CHI '94 Conference Proceedings*, pp. 285-291.
- Suchman, L. (1987). *Plans and Situated Actions: The Problem of Human-Machine Communication*. Cambridge University Press, UK.
- Ungar, D., & Smith, R.B. (1987). "Self: The Power of Simplicity," *OOPSLA 1987 Conference Proceedings*, pp. 227-242.