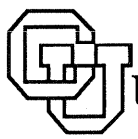
# Supporting Knowledge-Base Evolution with Incremental Formalization

Frank Major Shipman III

CU-CS-658-93 July 1993



University of Colorado at Boulder
DEPARTMENT OF COMPUTER SCIENCE

# Supporting Knowledge-Base Evolution with Incremental Formalization

by

FRANK MAJOR SHIPMAN III

B.S.E.E., Rice University, 1988

M.S., University of Colorado, 1990

A thesis submitted to the

Faculty of the Graduate School of the

University of Colorado in partial fulfillment

of the requirement for the degree of

Doctor of Philosophy

Department of Computer Science

1993

# Abstract

A number of systems have been built which integrate the knowledge representations of hypermedia and knowledge-based systems. Experiences with such systems have shown users are willing to use the semi-formal mechanisms of such systems leaving much structure implicit rather than use the formal mechanisms provided. The problem remains that it is hard (1) to encode knowledge in the formal languages required by knowledge-based systems and (2) to provide support with the semi-formal knowledge found in hypermedia systems. Incremental formalization enables users to enter information into the system in a informal or semi-formal representation and to have computer support for the formalization of this information. The domain independent Hyper-Object Substrate (HOS) differs from other systems that integrate hypermedia and knowledge-based system styles of representations in that it enables the incremental addition of formalism to any piece of information in the system. HOS actively supports incremental formalization with a set of tools which suggest new formalizations to be added to the information space. These suggestions are based on patterns in the informally and semi-formally represented information and the existing formalized knowledge in the information space. An important assumption is that suggestions need not be completely accurate to be of general benefit to users. These suggestions provide a starting point which can be edited, thus changing part of the process of formalization from creation to modification. XNetwork, an environment supporting the design of computer networks, is one of several applications that have been created with HOS. Experiences with HOS show that its flexibility for incrementally adding and formalizing information is useful for the rapid prototyping and modification of semi-formal information spaces.

# Acknowledgments

I want to first acknowledge the support of my advisor Gerhard Fischer, who provided the framework and environment in which this work was done. Ray McCall deserves special recognition for his encouragement of this work and our many discussions on hypermedia and its role in systems. I also thank the other members of my dissertation committee, Wayne Citrin, Mike Eisenberg, Skip Ellis, Clayton Lewis, and Jim Martin, for their valuable feedback and questioning of assumptions that have undoubtedly made this dissertation better.

I also thank Tony Gorry, Andy Burger, and Jesse Chaney for my experience while working on the Virtual Notebook System (VNS) as part of the Integrated Academic Information Management Systems project at Baylor College of Medicine. That experience greatly influenced both the topic of this dissertation and its subsequent development.

I owe a debt of gratitude to my coauthors whose ideas have made it in to this dissertation. Chapter 2 and Chapter 9 are largely based on papers written with Catherine C. Marshall. Chapter 4 contains many of the ideas of papers written with Gerhard Fischer, Ray McCall, Jonathan Ostwald, and Brent Reeves. Chapter 7 includes work presented in papers written with Brent Reeves. I also wish to thank my other coauthors whose ideas, while not being as easy to point to, have influenced this dissertation: Jesse Chaney, Andreas Girgensohn, Tony Gorry, Jonathan Grudin, Alex Jarczyk, and Peter Löffler.

The support of my friends Andreas Girgensohn and David Redmiles helped me through the frustrations that go with research and writing a dissertation. I especially thank David Redmiles for the numerous hours he spent discussing and editing this dissertation. Finally, I thank my parents, Pamela Baker and Frank Shipman Jr., for their support and understanding.

This research was supported in part through grants from the National Science Foundation (No. IRI-9015441) and Nynex.

# *Table of Contents*

# List of Figures



# List of Tables

# *Chapter 1: Introduction*

There has been a surge of interest in the integration of hypermedia and knowledge-based systems. Much of this interest has come from the realization that, though useful, knowledge-based systems are too expensive to create and maintain for many tasks. On the other hand, hypermedia systems are easier to build but are too passive for many uses. Thus, the interest in integration is in having both the active support of knowledge-based systems and the ease-of-use and ease-of-authoring of hypermedia systems.

A major difficulty encountered in integrating these two types of systems is that the representations for information in the two classes of systems have traditionally been quite different. Knowledge-based systems use *formal* languages, often based on production rules or frames, so that the system can use this information to provide services. Hypermedia systems use *semi-formal* representations to provide ways of linking together chunks of text, images and other media. These representations are called semi-formal because the hypermedia systems do not reason with the contents of a chunk of information, but only determine how that chunk is related to others and how to present (display) the chunk to the user.

The problem of integrating the two styles of knowledge representation can be solved with fairly straightforward methods. A number of approaches have been investigated, several of which will be discussed in the related work section of Chapter 5. Unfortunately, simply integrating the two representations does not address the difficulties of creating and maintaining knowledge-based systems and the lack of active support in hypermedia systems. The problem remains that it is hard (1) for people to encode knowledge in the formal languages required by knowledge-based systems and (2) for systems to provide support with the semi-formal knowledge found in hypermedia systems. A simple way of describing this problem is that people find informal representations easier to use, while the computer finds formal representations easier to use.

One method for trying to solve this problem is to search for a formal representation that is easy for the user and the system, the goal of end-user programming languages. This research has resulted in better programming languages, such as SQL and expert system shells. Although, these languages have been accepted as good languages for programming certain types of applications, they have not been widely accepted by users not trained in computer science.

At the other extreme, another way of reconciling the formal and informal is to make the system understand the informally represented information. This leads to the natural language problem, with respect to text, and the vision problem, with respect to images. There is much work towards solving these more general problems but general solutions along this line do not appear to be feasible without significant advances in the state of the art in these fields. The approach explored in this dissertation--"incremental formalization"--incorporates aspects of both of the above solutions: *The users enter information into the system in an informal or semi-formal representation and the computer supports the users' formalization of this information, incrementally over time.*

Chapter 2 describes, in more detail, the breadth of problems associated with formalisms, including experiences with formal representations in hypermedia, groupware, and knowledge-based systems. These experiences show how the actual use of systems often varies from the expectations of system designers; in the end, users reject formalizing information. The reasons why formal languages are more difficult to use than informal languages include problems of cognitive overhead, tacit knowledge, premature structure, and situational structure.

Chapter 3 describes the approach to incremental formalization taken in this dissertation: namely, the input of information in informal representations and the subsequent, gradual formalization of that information.

Because users may input information in an informal representation, the cognitive overhead required for the initial input is lowered. Once the information is in the system, the process of incremental formalization can be actively supported using mechanisms that suggest possible formalizations based on patterns in informally represented information. Also, incremental formalization enables the formalization of information whenever the user desires, such as when knowledge is no longer tacit or the user no longer is worried about premature structure.

Chapter 4 discusses the how incremental formalization can be integrated into the framework of domain-oriented design environments. These design environments use domain-oriented knowledge-based mechanisms to support designers. Incremental formalization fits into the seeding, evolutionary growth, and reseeding model of knowledge-base evolution with which design environments are designed and modified. Information that enters the knowledge base in an informal representation during use on real design tasks can later be formalized with the help of knowledge engineers during reseeding.

The approach of incremental formalization has been applied in two separate system projects. First, the Hyper-Object Substrate (HOS) was built from scratch to enable and support incremental formalization. The design and use of HOS will be described in Chapters 5 through 8. Second, the incremental formalization approach was added to Aquanet, an existing system which suffered some problems from its requirements of formalization from users. Chapter 9 describes the mechanisms developed to aid formalization based on previous usage of Aquanet.

Chapters 5 and 6 describe a system implementation that has the goal of providing support for the evolution of knowledge from less formal representations to more formal representations. Included in this work is the creation of an underlying object mechanism designed to help support this goal. This underlying layer, called the Hyper-Object Substrate (HOS), is described in Chapter 5. HOS information spaces are composed of a set of persistent first-class objects. These objects have a display component, such as a piece of text or image, and can have any number of attributes and relations and may play any part in inheritance relations. The representation and interface enable the gradual addition and modification of attributes and relations throughout the use of the information space.

Chapter 6 describes mechanisms that support the formalization process. Some of these mechanisms use the limited domain of the system to partially understand informal knowledge in or being added to the system. This limited understanding is used to aid the user in integrating the information in the informal knowledge with formal knowledge already in the knowledge base. HOS includes suggestions for new or modified attributes or relations based on textual analysis of the display text or textual values of attributes, and suggestions for views that may be appropriate for new hyperlinks based on the users' recent modifications to the information space.

This implementation work has been further refined to support the task of collaborative computer network design in the knowledge-based design environment, XNetwork. A discussion of network design and the extensions made to the basic mechanisms and tools to support this domain is found in Chapter 7. XNetwork provides an example of how domain-oriented design environments can be created using HOS and how HOS provides an integration of the different aspects of the design task not possible in previous design environments.

Chapter 8 describes observations from the use of HOS. This includes a discussion of how HOS was used in two class projects in the domains of archeology and neurosciences. The discussion focuses on how the observations of HOS's use reflect on the goals of incremental formalization.

As previously mentioned, Chapter 9 describes the application of the incremental formalization framework to the hypermedia system Aquanet. Unlike HOS, Aquanet was a system that had already been in use when

the work on adding incremental formalization took place. This chapter includes a study of the organizational conventions in diagrams, and a description of mechanisms built to support the formalization of information implicit in the layout of Aquanet discussions.

The use of such different mechanisms to support formalization in HOS and Aquanet leads to the question of how such mechanisms could be integrated within a single system. This and other open questions are discussed in Chapter 10. A summary of the dissertation and conclusions that can be drawn from this work are presented in Chapter 11.

The problems of requiring formalization from users are prevalent in many systems. Incremental formalization provides a framework which can be applied to a variety of systems to address these problems. The experience with the use of HOS and the development of formalization support in Aquanet provide evidence of the benefits of incremental formalization and examples to follow in enabling and supporting this process in other systems.

# *Chapter 2: The Problem with Formality: Experiences and Explanations*

Computer systems use abstract representations to support the user in a variety of ways: by structuring a task or users' work practices, by providing users with computational services such as information management and retrieval, or by simply making it possible for the system to process users' data. The abstractions used to provide such support are commonly expressed in a formal language [Chomsky 56].

When formalisms are embedded in computer systems, users must often engage in activities that might not ordinarily be part of their tasks: breaking information into chunks, characterizing information via keywords, categorizing information, or specifying relations between pieces of information. For example, with the Unix operating system, these activities might correspond to creating files, naming them, putting them in a directory structure, describing dependencies in "make" files or making symbolic links between directories or files.

The abstract representations that computer systems impose on users may involve varying degrees and types of formalization beyond what users are accustomed to. In some instances, little additional formalization is necessary to use a computer-based tool; text editors, such as vi or emacs, do not *require* additional formalization much beyond that demanded by other mechanisms for aiding in the production of linear text. Correspondingly, the computer can perform little additional processing. In other cases, more formalization brings more computational power to bear on the task; idea processors and hypermedia writing tools demand more specification of structure, but they also provide functionality that allows users to reorganize text or present it on-line as a non-linear work. These systems and their embedded representations are referred to as *semi-formal* since they require some - but not complete - encoding of information into a schematic form. At the other end of the spectrum, formal systems require people to encode materials in a representation that can be fully interpreted by a computer program. When the degree or type of formalization demanded by a computer system exceeds what the user expects, needs, or is willing to tolerate, the user will often reject the system.

Creators of systems that support intellectual work such as design, writing, or organizing and interpreting information are particularly at risk of expecting too great a level of formalization from their users. To understand the effects of imposing or requiring formality, experiences from the design and use of such systems are described.

This chapter first describes experiences related to the use of formalisms in systems ranging from general purpose hypermedia to knowledge-based systems. This is followed by a discussion of some problems which can lead users to reject formalisms: cognitive overhead, tacit knowledge, premature structure, and situational structure.

## 2.1 Experiences with Formalizations

The systems discussed in this section have been successful by many measures; yet they have all exhibited similar problems with user interaction that may be attributed to their underlying formalisms. To focus the discussion on these formalisms, any description of the interfaces by which users interact with the formalisms has been deliberately left out. The goal of doing so is to expose a seductive line of reasoning: providing the "right" interface to embedded representations will lead users to perform the desired formalizing of their task.

Many hypermedia systems try to coerce their users into making structure explicit. With few exceptions, they provide facilities for users to divide text or other media into chunks (usually referred to as nodes), and

define the ways in which these chunks are interconnected (as links). This formalism is intended as either an aid for navigation, or as a mechanism for expressing how information is organized without placing any formal requirements on content.

Systems that support argumentation and the capture of design rationale go a step further than general-purpose hypermedia systems in requiring users to formalize their information. They usually require the categorization of content within a prescriptive framework (for example, Rittel's Issue-Based Information Systems (IBIS) [Rittel 84]) and the corresponding formalization of how these pieces of content are organized.

Knowledge-based systems are built with the expectation of processing content [Waterman 86]. Thus, to add or change knowledge that the system processes, users are required to encode domain structure and content in a well-defined representational scheme. This level of formalization is built into a system with the argument that users will receive significant payback for this extra effort.

Groupware systems supporting coordination formalize something different from the structure of the information or its content. They expect a formalization of interactions between users of the system. This type of formalization allows the system to help coordinate activities between users, such as scheduling meetings or distributing information along a work-flow [Ellis et al. 91].

Each of these types of systems will be examined with a focus on how formalization influences system use and acceptance. Also, task-oriented systems that have specific formalisms embedded in them are examined specifically, systems for the capture of argumentation and design rationale [Jarczyk et al. 92] and design environments [Fischer et al. 92] that combine semi-formal design rationale with more formal representations of domain knowledge.

Formalisms used in each of these types of systems involve computer-mediated communication or coordination with other humans, or the capture and organization of domain knowledge. As a point of contrast, there is also a discussion of efforts to support the software engineering process, systems designed specifically to aid in the production of a formalized artifact, a computer program.

### 2.1.1 General purpose hypermedia

Hypermedia systems generally provide a semi-formal representation where chunks of text or other media, called nodes, can be connected via navigational links. The goal of these links is to accommodate individualized reading patterns through the non-linear traversal of the hyperdocument. Authors must formalize structure during the creation of such hyperdocuments.

Learning how to write, and to a lesser extent learning how to read, in a hypermedia system takes time. Observing users become accustomed with KMS [Akscyn et al. 88] during its use at Baylor College of Medicine, it became apparent that people do not easily accept new authoring styles. KMS is a page-based hypermedia system, meaning users author information in pages which can be linked together with navigational links. Some beginning users would write hierarchical outlines and full pages of text which they would connect by a single link to the next page of text, as if they were still using an outlining tool and word processor. By defaulting to the authoring practices of systems previously experienced, they avoided the decision of what information should be chunked together or what links should be created. Information that fit on a page became a chunk with a link to the next page.

Experiences with internal use of early prototypes of the Virtual Notebook System (VNS) [Shipman et al. 89], another page-based hypermedia system, also showed the added difficulties of chunking and linking information. Organizational conventions were decided upon within groups sharing "notebooks." These high-level conventions aided in understanding information from other users' notebooks, but there was still

a lot of variety between individuals in the amount of information on a page and the number of links created. More recent usage of the VNS outside of the development community shows a large variance in use of the system [Brunet et al. 91]. The heavy users build up sets of structured templates for reuse, thus reducing the overhead involved in adding structure to the information they are entering.

Marshall reports that training information analysts to use NoteCards revealed similar problems using formalizations [Shipman, Marshall 93]. Analysts had questions about chunking information into cards, naming cards, and filing cards. Monty documented similar problems in her observations of a single analyst structuring information in NoteCards [Monty 90].

Aquanet [Marshall et al. 91] is a hypermedia system with a substantially more complex model of hypermedia. Its model involves a user-defined frame-like knowledge representation scheme [Minsky 75] with a graphical presentation component. In a case study of a large-scale analysis task, it was observed that even sophisticated users with a background in knowledge representation had problems formalizing previously implicit structures [Marshall, Rogers 92].

### 2.1.2 Argumentation and design rationale

Recently there have been many different proposals for embedding specific representations in systems to capture argumentation and design rationale. Some of them use variations on Toulmin's micro-argument structure [Toulmin 58] or Rittel's issue-based information system (IBIS) [Rittel 84]; others invent new schemes like Lee's design representation language [Lee 90] or MacLean and colleagues' Question-Option-Criteria [MacLean et al. 89].

Some of the expected benefits of having a formal argumentation or design rationale are shorter production time, lower maintenance costs on products, and better designs [Jarczyk et al. 92]. There have been a number of applications of these mechanisms, from McCall's use of PHI [McCall et al. 83] to Yakemovic and Conklin's use of itIBIS [Yakemovic, Conklin 90]. The results can be interpreted both as successes and as failures. The methods did result in long-term costs reduction but success depended on severe social pressure, extensive training, or continuing human facilitation. In fact, Conklin and Yakemovic reported that they had little success in persuading other groups to use itIBIS outside of Yakemovic development team, and that meeting minutes had to be converted to a more conventional prose form to engage any of these outside groups [Conklin, Yakemovic 91].

Like general-purpose hypermedia systems, the argumentation and design rationale systems force their users to divide information into chunks which in this case are categorized as certain types, such as *issue*, *position*, or *argument*. Users of these methods must then specify connections between chunks, such as *answers*, *supports*, or *contradicts* links. Users have several problems in effectively formalizing their design rationale or argumentation in this type of system; these problems can be predicted from the previously described experiences with hypermedia.

First, people aren't always able to chunk intertwined ideas; users create, for example, positions with arguments embedded in them. Second, people often disagree on how information can be classified and related in this general scheme; what one person thinks is an argument may be an issue to someone else. It is easy to become engaged in extended arguments with collaborators on how pieces of design rationale or arguments were interrelated, and about the general heuristics for encoding statements in the world as pieces of one of these representation schemes. Marshall and colleagues [Marshall et al. 91] provide a short discussion of collaborative experiences using Toulmin structures. Finally, there is always information that falls between the cracks, no matter how well thought out the formal representation is. Conklin and Begeman document this latter problem as well in their experiences with gIBIS [Conklin, Begeman 88].

### 2.1.3 Knowledge-based systems: design environments

Design environments consist of a number of components integrated to support the process of design [Fischer et al. 92]. These design environments include different mechanisms for representing domain knowledge, including formally represented design units and rules, semi-formal argumentation and informal textual annotations.

This variety of knowledge representations has led to the development of different mechanisms for supporting the modification of knowledge in the systems. One such mechanism is the set of end-user modifiability (EUM) tools developed to support designers in modifying and creating formal domain knowledge with task agendas, explanations, and examples [Fischer, Girgensohn 90]. In a description of user studies on EUM tools, Girgensohn notes that most of the problems found in the last round of testing "were related to system concepts such as classes or rules [Girgensohn 92]." In short, these user studies revealed that, although the EUM tools made the input of knowledge significantly easier, users still had problems manipulating the formalisms imposed by the underlying system.

### 2.1.4 Knowledge-based systems: general

Knowledge-based systems have long exclaimed the goal of having users add or correct knowledge in the system. End-user knowledge acquisition imposes formalization requirements on users that are similar to those imposed by design environments except that they lack much of the support provided by the EUM tools. Users must learn the knowledge representation used by the system, even if it is hidden by a good interface, so they may understand the effects of their changes.

A different approach to the problem of creating user modifiable expert systems was taken by Peper and colleagues [Peper et al. 89]. They eliminated the inference engine, leaving a hypermedia interface in which users were asked questions and based on their answers, were directed to a new point in the document. For example, a user might see a page asking the question "Did the warning light come on?" with two answers "Yes" and "No". Each answer being a link to further questions or information based upon the answer of the previous question.

With this system users could add new questions or edit old questions in English since the computer was not doing any processing over the information. By reducing the need for formalized knowledge, they achieved an advantage in producing a modifiable system, though at the cost of sacrificing inferencing.

### 2.1.5 Groupware systems

Groupware systems that require the formalization of procedure and interaction have suffered many of the same problems as systems that enforce formalization of structure and content. For example, systems that extend electronic mail by attaching properties or types to messages require their users to classify exactly what type of message they are sending or what type of reply is acceptable. Experiences with systems like the Coordinator [Winograd, Flores 86] and Information Lens [Malone et al. 86] point out that many users ignore the formal aspects of such systems, and generally use them as basic electronic mail systems [Bullen, Bennett 90].

Coordination-oriented systems have the additional burden of formalizing social practices which are largely left implicit in normal human-human interactions. An example is the limited success of automatic scheduling systems [Grudin 88] due to the unwillingness of users to describe their normal decision methods for whether and when to schedule a meeting with other people. The same rules of scheduling that apply to your boss do not apply to an unknown person, but formalizing such differences is difficult.

### 2.1.6 Software engineering

The process of software engineering echoes the difficulties described above. As in the above situations, people (in this case including programmers) are required to explicitly communicate information to a computer. The interfaces through which this communication occurs, in the form of specification tools or programming languages, are often part of the problem, but they only contribute what Brooks calls "accidental complexity" [Brooks 87] to the overall task. Whether a person uses popup menus, dialog boxes, "English-like" formal languages, or low level programming languages to state the information explicitly, the person must still know what they want to state, be it a relationship between two pieces of text or a complex algorithm.

In software engineering, deciding what needs to be stated explicitly (the specification and actual program code) has been termed "up-stream activity" to distinguish it from the "down-stream activity" of instantiating the specification [Myers 85]. Software engineering tools that focus on the up-stream activity are meant to support the process of coming up with a specification, the storage and retrieval of information associated with this process, and visualization of the result. While the results are still out on the successfulness of these initial tools, the same goals could be used to focus work on supporting formalization in the above classes of systems.

## 2.2 Why Users Should Not be Required to Formalize

From the above discussion it seems apparent that the problems of expecting formalization are endemic. This section explains why the users are making the right decisions, in some sense, by resisting premature, unnecessary, meaningless, or cognitively expensive formalization.

From the user's perspective formalization poses many risks. "Why should I spend my time and effort formalizing this when I have other things to do?" "What do I do when the ideas or knowledge is tacit and I cannot formalize it?" "What if I commit to this formalization only to later find out it is wrong?" "Why should I formalize this when I cannot agree with anyone else on what the formalization should be?" These are all valid questions and the answers that systems provide are often insufficient to convince people to use a system's formal aspects.

### 2.2.1 Cognitive overhead

There are many cognitive costs associated with adding formalized information to a computer system. Foremost, users must learn a system's formal language. Some domains, such as circuit design, have specific formal languages (e.g., circuit diagrams) to describe a certain type of information. More generic formal languages, such as production rules or frames, are almost never used for tasks not dealing with a computer. While knowledge-based support mechanisms and interfaces can improve the ability of users to successfully use formal languages, Girgensohn's experience shows that system concepts related to underlying representations still pose major obstacles for their use [Girgensohn 92].

Even knowing a system's formal language, users face a mismatch between their conception of the information and the system's formal representation; they face a conceptual gap between the goals of the user and the interface provided by the system. Norman describes the requirements to bridge this gap or "gulf of execution" [Norman 86]:

> "The gap from goals to physical system is bridged in four segments: intention formation, specifying the action sequence, executing the action, and, finally, making contact with the input mechanisms of the interface." [Norman 86] (page 39)

As this implies, formalisms are difficult for people to use often because of the many extra steps required to specify anything. Many of these extra decisions concern chunking, linking, and labeling, where formal

languages require much more explicitly defined boundaries, connections between chunks, and labels for such connections than their informal counterparts.

The obstacle created by this conceptual gap between users' goals and systems' formal languages was observed in an early prototype of the Virtual Notebook System's "interest profile matcher." The goal of the profile matcher was to enable users of the system to locate other users with certain interests and expertise. The vocabulary used in profiles was the Medical Subject Headings (MeSH), a set of around 20,000 terms divided in about twelve interconnected trees (forming a directed acyclic graph) which is used by medical journals to index articles. Defining an interest profile required choosing terms out of the hierarchies of concepts which best described one's interests. Queries for locating people also required choosing terms from MeSH terms and attaching "matching ranges" so that all terms in a given range in the MeSH hierarchies would be considered a match. The matching ranges were necessary because MeSH was large enough to experience the vocabulary problem [Furnas et al. 87]--people using different terms to describe the same topic. With the increase in expressiveness in queries came an increase in difficulty to define queries. Work on the profile matcher was discontinued because the effort required to define interests and queries of sufficient clarity overcame the usefulness of the service the system was to provide.

### 2.2.2 Tacit knowledge

Tacit knowledge is knowledge users employ without being conscious of its use [Polanyi 66]. Tacit knowledge poses a particularly challenging problem for adding information to any system since it is not explicitly acknowledged by users. The problem of tacit knowledge has resulted in knowledge engineering methods aimed at exposing expertise not normally conscious in experts, such as one described by Mittal and Dym:

> "We believe that experts cannot reliably give an account of their expertise: We have to exercise their expertise on real problems to extract and model their knowledge." [Mittal, Dym 85] (page 34)

When such introspection becomes necessary to produce and apply a formal representation during a task it necessarily interrupts the task, structures and changes it. These changes may be detrimental to the user's ability to perform their task. Hutchins et al. are discussing such a modification of the task when they say:

> "While moving the interface closer to the user's intentions may make it difficult to realize some intentions, changing the user's conception of the domain may prevent some intentions from arising at all. So while a well designed special purpose language may give the user a powerful way of thinking about the domain, it may also restrict the user's flexibility to think about the domain in different ways." [Hutchins et al. 86] (page 108)

An example of this interference is McCall's observation that design students have difficulty producing IBIS-style argumentation even though videotapes of their design sessions show that their naturally occurring discussions follow this structure [Fischer et al. 91]. A physiological example of the interference that making tacit knowledge conscious can cause is breathing (also from McCall). When a person is asked to breath normally, their normal breathing will be interrupted. Furthermore, chances are that introspection about what normal breathing means will cause the person's breathing to become abnormal - exaggeratedly shallow, overly deep, irregular.

### 2.2.3 Premature structure

One well known reason why users will not formalize is the negative effects of prematurely or unnecessarily imposing a structure [Halasz 88]. One problem is that creating a new formalization from information in a different formalization may be more difficult than formalizing the information from an informal state.

In his studies of how people organized information in their offices Malone found that the negative effects of prematurely structuring information were directly mentioned in describing their offices [Malone 83]. In particular, one of the subjects in Malone's study said of a pile of papers waiting to be filed:

> "You don't want to put it away because that way you'll never come across it again. ... it's almost like leaving them out means I don't have to characterize them. ... Leaving them out means that I defer for now having to decide--either having to make use of, decide how to use them, or decide where to put them." [Malone 83] (page 107)

This quote points out the perception that information formalized incorrectly or inconsistently will be more difficult to use or simply be of less use than information not formalized. This problem can also be observed in the directory structures of UNIX, Mac OS, or DOS. Many users have large numbers of disassociated files at the top level directory (or file-box) of their machine or account. Many of these users know how to create subdirectories or folders to organize their files but postpone classification until they "have more time" or "the mess gets too bad." For these users the perceived benefit of organizing their files does not make up for the effort required to organize the files and the possible cost of mischaracterizing the files.

### 2.2.4 Situational structure

The difficulties of creating useful formalizations for individual use are compounded when different people must share the formalization. Different people necessarily have different world experiences and likely will have different views of the task that the formalization is to support. Formalization makes such agreements difficult because it requires the formalized information to be stated explicitly so that there is little room for different interpretations.

For different people to agree on a formalization they must agree on the chunking, the labelling, and the linking of the information. As has been discussed in the context of earlier examples in the use of tools to capture design rationale, the prospects of negotiating how information is encoded in a fixed representation is at best difficult. Differences occur not just within a group of users but between groups as well. A study of the communication patterns in biomedical research groups showed that the characteristics of the research being performed influenced the organization and communication of the research groups [Gorry et al. 78]. A system which attempts to impose a particular structure on communication will likely not match the appropriate communication structure for any given group.

The problem of situational structure does not occur only when multiple people use the same structure but can also occur when the user's task changes. The context of the new task may not match well with the structuring scheme. In listing what are commonly considered the "most important properties" of a "formal system", Winograd includes:

> "There is a mapping through which the relevant properties of the domain can be represented by symbol structures. This mapping is systematic in that a community of programmers can agree as to what a given structure represents." [Winograd, Flores 86] (page 85)

Experience and intuition seems to indicate that domains for which this is true may be quite small and task dependent. Anecdotal evidence shows that a representation that is suitable for one task may not be appropriate for a very similar related task. For example, Marshall and Rogers describe [Marshall, Rogers 92] how a representation developed for the process of assessing foreign machine translation efforts proved to be of limited value in the closely related task of evaluating Spanish-English machine translation software. The second task shared a subset of the content with the first task, but the representation did not formalize appropriate aspects of the material. Attributes like speed and accuracy as well as cost and computer platform turned out to be very important in evaluating software, but only of secondary importance in a general assessment of the field, while in the general assessment of the field, the technical

approach of the various systems was deemed important. In short, different situations require different user support and thus different formalized structures [Suchman 87].

## 2.3 Summary

This chapter describes difficulties experienced with the acceptance and usage of systems that require users to formalize information. Users will sometimes accept such systems but refrain from using aspects of the system which require formalization. These problems are pervasive in systems designed to support intellectual work such as hypermedia, argumentation, knowledge-based systems, and groupware.

The difficulties that users have in formalizing information is not just an interface problem. Users are hesitant about formalization because of a fear of prematurely committing to a specific perspective on their tasks. It may also be difficult for them to formalize knowledge that is usually tacit. The added cost of formalizing information over using informal information makes formalized information far less attractive for users to provide. In a collaborative setting, people must agree on a formalization and the heuristics for encoding information into it. Such considerations by users cannot be overcome simply by providing a better interface to the system's formalisms.

# *Chapter 3: Incremental Formalization in Evolving Information Spaces*

A variety of systems types include information spaces that evolve over long periods of use. Common examples are hypermedia, database and expert systems. The people who are responsible for maintaining such information spaces--e.g. database administrators, knowledge engineers, and sometimes system users, face problems such as described in the previous chapter. In fact, many of the experiences discussed in the previous chapter come from systems that include some form of evolving information space. For the purposes of this dissertation, the term "evolve" is used in the general sense of "changing over time."

This chapter presents an approach to reducing the problems of formalization in systems with evolving information spaces. This approach is to allow the *incremental formalization* of information: *users enter information in a less formal representation and gradually formalize that information over time.* Incremental formalization is particularly applicable to systems which include evolving information spaces since they already support some type of modification of information over time.

Some amount of information must be formalized for a computer system to perform almost any task. For example, a word processor must know the order of characters, a drawing program must know the color and shape of objects being drawn, and a circuit analyzer must know the logical circuit design. Interaction based on a formalism can become transparent when the user has become skilled in the formalism. Failure to motivate the user to formalize information that is essential for the central task means failure of the system.

The first part of the chapter describes incremental formalization in more detail, including how it can be supported. The second section of the chapter discusses how incremental formalization addresses the problems raised in Chapter 2.

## 3.1 Incremental Formalization of Information

The advantage for the users in converting knowledge from informal to formal representations is that the system is able to provide better support services to the user with formally represented information. A critical issue is whether the user perceives this benefit as outweighing the effort required to formalize information at the time the formalization is done. This cost/benefit trade-off must be considered with respect to the individual providing the extra effort required to formalize information rather than over the whole group of users of the formalized information [Grudin 88].

The benefits of formalizing information are the added support the computer provides with the formally represented information, the preciseness of formally represented information, and the usefulness of formalization as part of analysis. Incremental formalization, in comparison to traditional approaches of acquiring formal information, occurs through a number of lower-effort steps (see Figure 1). This is analogous to the "divide-and-conquer" algorithms in theoretical computer science [Aho et al. 74]. The costs associated with these individual steps can be lowered further by providing support tools to aid the user in this process. Further, the cost/benefit trade-off implies that formalized information will most often be provided when the formalization is perceived to serve the user's current task, leading to demand-driven formalization, or "formalization on demand".

The transfer of information from informal to formal representations is not purely syntactic. Informal representations allow more information to remain implicit, and converting this information to formal representations requires some of this implicit information to be reevaluated while being made explicit. This can be seen in the difference between explaining conceptually how an algorithm works and actually

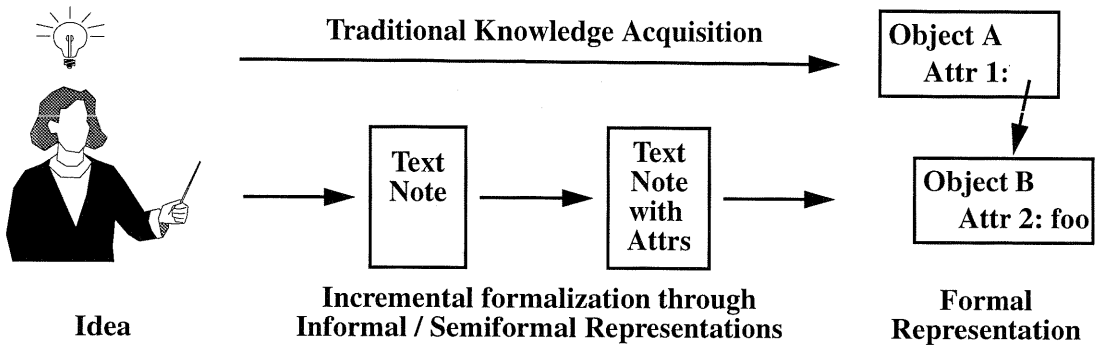



**Figure 1. Diagram of Incremental Formalization**

Conventional approaches to knowledge acquisition require the user to go immediately from an idea to a formal representation. With incremental formalization the information is formalized through a number of intermediate steps, such as a textual annotation that can collect attributes.

writing the code which implements the algorithm. The conceptual explanation allows many details to remain unstated, while the code requires the algorithm to be specified exactly.

## 3.2 Actively Supporting Incremental Formalization

Systems including mechanisms enabling incremental formalization can foster the use of these mechanisms by supporting the user in formalizing information. This support can be provided through tools which support the general process of formalization and through tools which suggest possible formalizations.

The process of formalization includes determining what formalizations to add and how to add them. Support for locating relationships between information aids the user in deciding what to formalize. This support can take the form of structure and content-based query languages [Halasz 88], full-text search mechanisms, and associative information retrieval techniques [Henninger 93]. Once the user has decided what to formalize, the system can aid the user in adding this information. Such support can take the form of context-sensitive help and other mechanisms for end-user modifiability [Girgensohn 92]. Other types of process support, such as providing interfaces appropriate to browsing through [Tesler 81] and maintaining consistency within [Ballance, Graham 91] knowledge bases can be found in knowledge engineering environments.

As an extension to supporting the process of incremental formalization, a system can provide mechanisms which suggest specific formalizations; thus supporting both the decision of what to formalize and how to formalize it. The suggestions need not be completely accurate to be of general benefit to the users. With suggestions the users' role in formalization can be switched from recall and creation to recognition and modification. For example, a suggestion based on natural language processing might provide a set of attributes and values for a piece of text. These suggested attributes could then be refined by the user.

Suggestions can be based upon the variety of information that the system has available: the already formally represented information as well as information placement, textual content, and other less formal aspects. Mechanisms already available in the areas of text grammars [Rama, Srinivasan 93], visual grammars [Lakin 87], knowledge discovery in databases [Frawley et al. 92], and system generated links in hyperdocuments [Bernstein 90] can be used to determine suggestions. Such mechanisms creating suggestions for formalizations are necessarily heuristic in nature.

## 3.3 Considerations in Enabling Incremental Formalization

There are a number of different representations and interfaces that would enable incremental formalization. Also, there are many mechanisms for determining suggestions for formalizations. Three goals should be considered when choosing an approach to incremental formalization: *"in-place" formalization*, *non-destructive formalization*, and *bootstrap formalization*.

### 3.3.1 "In-place" formalization

One of the costs associated with formalizing information is amount of interaction required to formalize information. One approach to lowering the amount of required interaction is to allow information to be formalized "in place", i.e. not requiring the removal or copying of information already in the system during the process of formalization.

### 3.3.2 Non-destructive formalization

Another goal to be considered when deciding on an approach for enabling incremental formalization is to enable non-destructive formalization. Formalization is non-destructive when it does not remove or overwrite the informal information being formalized. Different levels of formality are appropriate for different types of information. Informal representations may be preferred for communicating between users, but to gain computer support the same information may need to be formalized.

It is generally useful to preserve the informally represented information from which the formal representation is derived. Informal representations, such as natural language, have the ability to contain an indefinite amount of implicit information. An example is that depending on knowledge of the author of a piece of text, the wording can convey the author's opinion, mood, and covert goals. By not removing the source of formal information, the information not considered important enough to formalize remains available. Also, informal representations of information will be of use later should the need to better understand and evaluate a piece of formally represented information arise [Hofmann et al. 90].

### 3.3.3 Bootstrap formalization

Bootstrap formalization relies on suggestion mechanisms that can take advantage of formalized information already in the system. By doing so, the suggestion mechanisms may initially be able to provide relatively few suggestions but when more information is available the suggestions can improve in quantity and quality. Some suggestion mechanisms that use formal representations, such as those based on lexicons, can provide a greater variety of suggestions as more formal information is made available.

The quality of suggestions from certain classes of algorithms will generally improve when given more input. For example, suggestions based on statistical methods, such as Hidden-Markov Models [Schäuble, Glavitsch 90], would generally become more accurate as they have more information to process.

## 3.4 Solving the Problems with Formality

Chapter 2 described experiences and categorized some of the problems associated with formalisms in a variety of computer systems. The problems described are related to cognitive overhead, tacit knowledge, premature structure, and situational differences. Incremental formalization does not solve all of these problems but it can provide some benefit in each case.

### 3.4.1 Cognitive overhead

Incremental formalization reduces the overhead of entering information, and the later formalization of that information. Initial addition of information is changed from requiring the user to provide a formal language description to allowing the users to add information in an informal representation. Such a

difference provided increased modifiability in Peper's use of a hyperdocument instead of an expert system [Peper et al. 89].

Incremental formalization divides up the overhead associated with formalizing information in the system by dividing up the process. By formalizing less information at a time the number of decisions concerning chunking, linking, and labelling that the users must make is reduced. The overhead associated with having to be explicit still exists for incremental formalization, but the individual steps require less explicit information.

In determining the chunking, linking, and labelling of information, the user crosses the first of the four segments bridging Norman's gulf of execution, intention formation [Norman 86]. Besides supporting intention formation, suggestion mechanisms actively support the second and third segments of Norman's bridge, specifying the action sequence and executing the action, by providing specifications and easy executions of actions. The fourth stage of Norman's bridge, making contact with input mechanisms, is not addressed by incremental formalization.

### 3.4.2 Tacit knowledge

The problem of representing tacit knowledge is difficult for all systems requesting information from their users. Informally represented information can capture some of the users' tacit assumptions through their use of language and other informal media. While it is not likely that the computer will be able to recognize much of such implicit information, other users can interpret it.

Suggested formalizations also have the possibility of bringing previously tacit knowledge to consciousness. Recognized patterns in informal information may be the result of tacit knowledge, triggering the recognition that this information is important. Such an occurrence was reported in the use of Infoscope [Stevens 93]. In the use of Infoscope, where information filters are suggested based on the users' reading patterns of USENET News, a particular suggestion triggered a better understanding in the user of his/her goals.

### 3.4.3 Premature structure

Systems which enable incremental formalization do not require the imposition of premature structure on information being added. Like the desks of the office workers in Malone's study [Malone 83], information in such systems can be kept without structure until the user wants to add structure. Also, since structure does not need to be added all at once, the user can add just the structure they feel comfortable adding, leaving other possible structure for later.

Resistance to premature structure can still be a problem if users never feel ready to formalize. This problem is related to the problem of intention formation, which is discussed above as part of the problem of cognitive overhead.

### 3.4.4 Situational structure

The problem of using formally represented information for different situations is only partially addressed by incremental formalization. Incremental formalization requires that formal representations are able to evolve. By adding situation specific formalisms the user may modify existing or add new situation specific structures as needed.

Structure imposed on information for one task may not be of use for a new task without major changes to the chunking, linking, and labelling of information. In the cases when there is no way to resolve the different structures necessary for the different situations, informally represented information may still be useful in deriving the new required structures.

## 3.5 Applicability of Incremental Formalization

Incremental formalization is not a possibility for all systems or all tasks. The requirements for the application of incremental formalization reflect the long-term and gradual nature of the process. First, applying incremental formalization requires that the information space will be in use for a extended period of time. Tasks for which information is only of use for short periods of time will be less likely to benefit from incremental formalization as information will need to be formalized quickly to be of use. Second, the information space must be such that small additions of formal knowledge will be of use. Tasks where all information must be formalized for any part of this information to be of use will not benefit from incremental formalization. This is analogous to the "critical mass" problem that has also been discussed in the discussion of acceptance of computer-supported cooperative work systems [Markus, Connolly 90].

## 3.6 Other Methods of Addressing Formalization Problems

The problems of formalization are too great to be addressed in whole by any single solution. As long as a particular formalism is not crucial to the main goal of a system, this formalization task may be avoided by users without jeopardizing the success of the system. Many systems provide features which are not necessary for the most common uses of the system but are available for users who want the added benefits of providing more information. Spreadsheet programs include many features which are used only by a small percentage of the user community [Nardi, Miller 90]. The rest of the users either get by without using the features, or ask for help when they cannot avoid doing otherwise.

Problems of scale, i.e. too many inferences to make users acknowledge each piece of inferred structure, lead to more automatic reasoning by the system instead of suggestions. This approach provides services to the user based on informally represented information; structures can be inferred by textual, spatial, temporal, or other patterns. The system's inferences will be incorrect at times but, as long as the inferences are right part of the time and it is apparent to the user when the system has made the wrong inference, these features will cost the user little for the benefit they provide.

## 3.7 Summary

While not addressing all the problems that users have in dealing with system formalisms, incremental formalization does partially address the problems of cognitive overhead, tacit knowledge, premature structure, and situational structure outlined in Chapter 2. The general approach is to "divide and conquer" the problem of formalization by dividing up the initial problem of getting formally represented information from the users and attempts to support the smaller steps that result.

There are decisions that system designers can make to reduce the need of formal information by systems and also methods to reduce the difficulty for users in providing this information. Building systems that enable the process of incremental formalization and structure evolution is one such decision. The reasons why users resist formalization, namely the threat of premature formalization, time constraints that limit effort, and inability to formalize with their current understanding, may change over time. Furthermore, tasks are frequently reconceptualized during performance. This evolutionary nature of problem solving is why incremental formalization can be a benefit.

Supporting gradual formalization can extend beyond enabling users to choose the level of formalization when adding information. Systems can be designed to support the process of formalization and can also make suggestions about what formalizations might be appropriate by noticing patterns in informally represented information. Suggestions based on such inferences do not have to be correct all the time; the user just needs to be able to know when to accept them and when not to. Suggestions can also lower the cost of defining structure, by providing an initial formalization which can then be modified rather than having to be created from scratch.

# *Chapter 4: Application to Domain-Oriented Design Environments*

When considering applying incremental formalization to a particular application or class of applications the existing representation and acquisition characteristics of the system(s) must be considered. This chapter discusses the application of incremental formalization to domain-oriented design environments. Domain-oriented design environments consist of a number of components, such as construction kits and argumentation sub-systems, integrated through knowledge-based mechanisms to support the process of design [Fischer et al. 92].

The first part of this chapter discusses the representations and existing support for modification of the design environment information space. Next is a discussion of the evolutionary nature of design and how knowledge is acquired by design environments through phases of seeding, evolutionary growth, and reseeding. The last part of this chapter discusses the different rates at which formal and informal information can be expected to enter the design environment during the seeding, evolutionary growth, and reseeding phases and how incremental formalization can support this acquisition in all the phases.

## 4.1 Representations and Modification

Domain-oriented design environments include a number of different mechanisms for representing knowledge, including semiformal and formal knowledge representations for domain knowledge. The different components of the design environment contain information in different representations ranging from natural language text to inheritance hierarchies. (See Figure 2.)

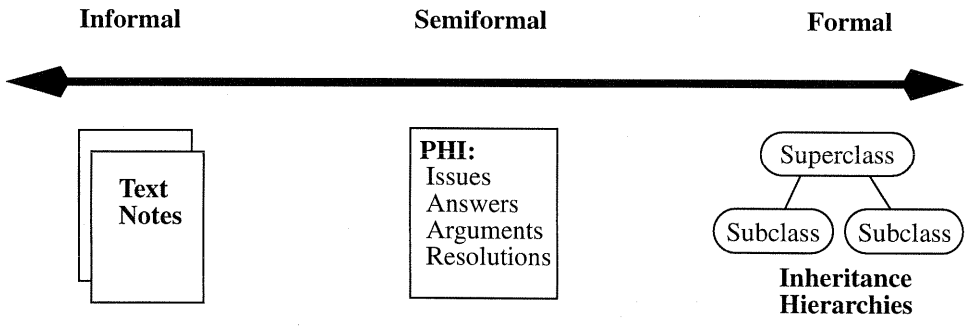



**Figure 2. Range of Formalisms in Domain-Oriented Design Environments**

Information in a knowledge-based design environment is represented in a variety of methods ranging from informal, like text notes, to formal, like inheritance hierarchies and production rules.

Construction kits represent domain objects as graphical objects that are directly manipulated. Issue bases contain argumentative discussions represented by textual records linked together as hypermedia. Knowledge-based critics represent design rules and principles as predicates which the system uses to evaluate partial designs as they are created by designers. Construction kits and knowledge-based critics are formal representations of design knowledge and are interpreted by the computer to provide better support for the designer. Textual annotations and notes, on the other hand, are informal design knowledge because they are interpreted by the designer only. The argumentation mechanism in design environments includes a

semiformal knowledge representation in the form of the Procedural Hierarchy of Issues (PHI) method [McCall 87] of issue-based deliberation [Kunz, Rittel 70].

With these different knowledge representations there have been different mechanisms developed for supporting the modification of knowledge represented in different forms. Systems in which PHI-style argumentation is editable and extensible have existed since the early 1980s [McCall et al. 81]. Knowledge-based tools supporting the designer in modifying and creating palette items and critic rules were developed and general principles for achieving end-user modifiability have been outlined [Girgensohn, Shipman 92]. Still, all of this work has focussed on supporting the use of individual representations. In order for incremental formalization to occur the existing mechanisms must be augmented by methods to support the transfer information represented in one representation to other representations in the design environment.

## 4.2 Design as an Evolutionary Process

The difficulty that designers have in formalizing information is partly due to the evolutionary nature of design. Designers using a design environment do not have a static understanding of the domain or of the issues involved in their task. They gain understanding of their specific task as they follow the interaction between their design and the various constraints placed upon the design process [Simon 81][Rittel 84][Schön 83]. Their understanding of the design issues in a task gradually evolves along with their design [Suchman 87]. An example in the computer network domain is when a new type of machine is added to the network. Over time the network administrator will learn more about how this new machine interacts with the other machines on the network. Figure 3 shows an example of an initial problem being recorded and later, when the designer has more information, the information initially entered being elaborated.

Initially, a designer may attempt one partial design scheme only to later find that this does not work as expected. This is called a breakdown situation [Winograd, Flores 86]. Schön has said that it is at this time the designer often gains some (possibly limited) understanding of the unexpected problem in the design and acts on this insight [Schön 83].

Designers will not be able to immediately enter new knowledge in its final form into the design environment unless the information involved is very simple. But capturing the new information in some form is important because as time passes, the "story" the designer tells about why the information changes to reflect the designer's more recent experiences [Schank 90]. Because of (1) the limited understanding the designer may have of the new information, and (2) the need to not require too much effort from the designer at this time, the design environment should try to remain transparent by allowing the designer to enter this information using the method most comfortable to the designer. Text and other representations used in human-human communication are the most appropriate to provide for use since designers will have much experience with them and since they allow much information to remain implicit.

As designers encounter unforeseen problems in the design task, they will come to understand more about the domain and the relationships between information within the design environment, sometimes making it possible to formalize previously informally represented information. Support for the incremental modification of knowledge, not just within a single representation, but also across representations, is needed to support this information formalization process.

## 4.3 Knowledge Acquisition in Design Environments

The approach to knowledge acquisition in design environments falls between two extreme approaches used by most other systems. One is to input information in advance of use; this has been typified by expert and information retrieval systems. The other is to start with an empty system and let its information base to grow and become structured as a consequence of use. This latter approach is characterized by most hypermedia systems that support authoring.

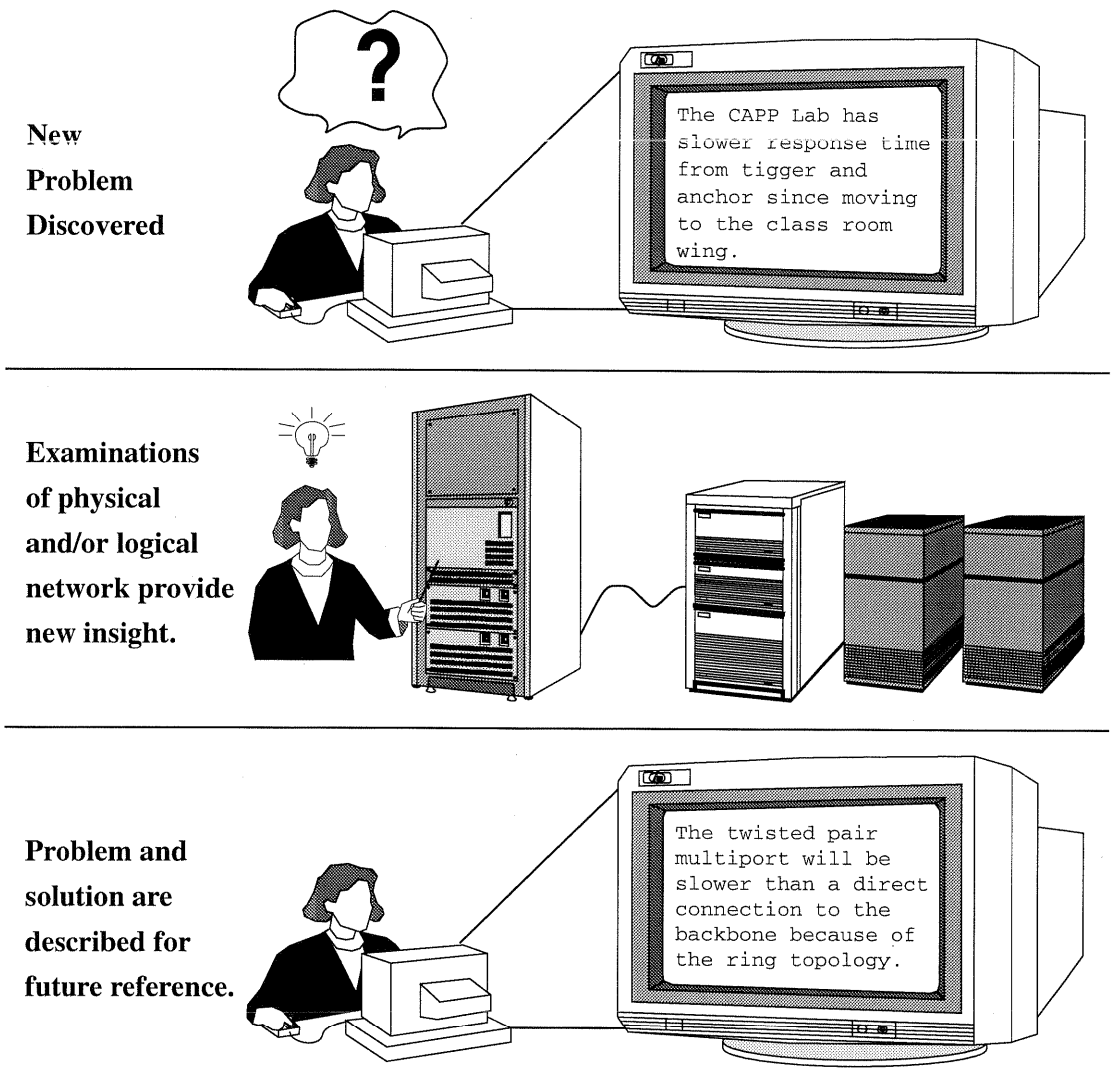



**Figure 3. Evolution of Problem Understanding**

The designer's knowledge of the domain and their particular problem changes over time. The system's information needs to be able to evolve along with the designer's understanding.

The first approach, putting all the information in the system prior to use, unacceptably limits the uses of a system. It prohibits the addition or modification of information in the system. This is not possible for most information systems designed to support the sharing of information. This approach is also impractical in rapidly changing domains. The second approach, starting with an empty information space and letting the users put in their own information, has resulted in the problems outlined in Chapter 2 when the added information needs to be formalized.

The approach to knowledge acquisition in design environments avoids the two extremes and has been described by Fischer et al. in [Fischer et al. 92]. This approach begins by starting the system off with a "seed": an initial set of information and methods. A seed is incomplete but sufficiently well-developed to be capable of growth. After this initial seeding, the information base will grow for a period of time in an

"evolutionary" manner through normal use of the system. Occasionally, this growth-through-use must be interrupted by a deliberate effort at revision and infusion of information and methods. This is called "reseeding". The system can then be used again and its information base will grow, but must be periodically reseeded.

### 4.4 Seeding, Evolutionary Growth, and Reseeding

The defining characteristic of the evolutionary approach to information space growth is the use of a seed. A seed is the initial collection of information, created by knowledge engineers and domain experts, that is delivered with the system to the users. Underlying the use of a seed is the notion that users are more likely to use and add to the information space if they do not have to create it from scratch. "Seeding" the information space reduces the user's required input to just that knowledge left out of the initial information space or that knowledge specific to their task. In addition to requiring less work of the user to create their specialized information space, the seed can provide information on aspects in which they do not have expertise.

When provided to the users, a seeded information space will not be complete for the same reasons that other systems cannot have all the information put into the system before it is given to the users. The users need to be able to add to and edit the information space. Such addition of information can occur naturally if the users of the system already use electronic media to communicate with one another or otherwise as a source for information. By providing for communication and external information resource interactions as part of the system the information space will grow without requiring effort beyond that already performed by users of electronic mail.

While the system is in use the information space will likely become less organized and some information will become out-of-date. When these problems become too great the information space needs to be reseeded. Knowledge engineers reseed the information space by reorganizing, generalizing, and formalizing the information so that the information provides a greater benefit to the user.

### 4.5 Comparing the Growth in Formal vs. Informal Information

For understanding the role that incremental formalization can play in this process, the seeding, evolution through use, and reseeding will be discussed within the context of the growth of formal information compared to the growth of informal information, as shown in Figure 4. Not shown on this graph are the further evolution through use and reseeding phases which would continue alternating throughout the use of the information space.

The initial seeding, done by knowledge engineers in conjunction with domain experts, will produce both formally and informally represented information. Formalized information enters the system slowly during use phases as compared to both the seeding and reseeding phases due to the extra effort required for users to add formalized information. One goal of reseeding is to organize, formalize, and generalize the information entered during the use phase. While reseeding also updates the information space to include new design components and features, similar to the role of new releases of commercial software, the emphasis on working with information acquired during use distinguishes reseeding from the traditional software revision process. The emphasis on organizing and formalizing information entered during use is why the line representing total information in Figure 4 flattens out during reseeding while the line for formalized information becomes steeper than during the use phase.

### 4.6 Role of Incremental Formalization

As shown in Figure 4 formalized information is entering, albeit at different rates, throughout the life cycle of a design environment's knowledge base. Because of this incremental formalization can be of use in all

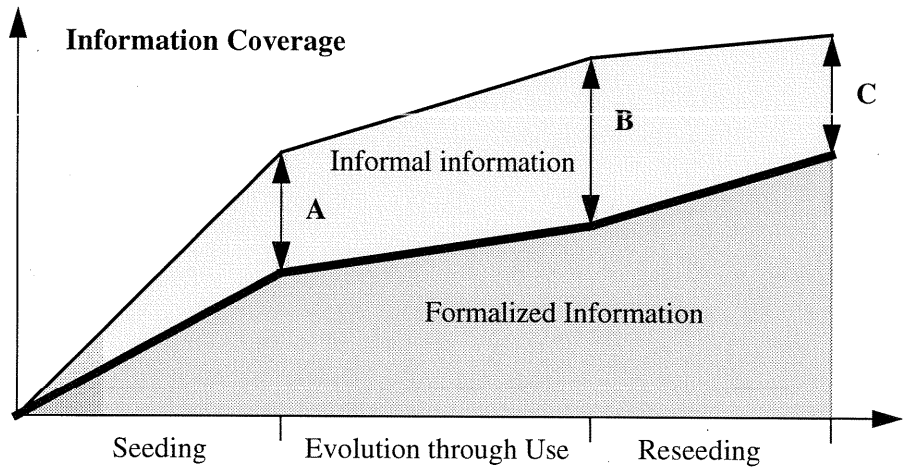



**Figure 4. Graph of Formalized and Total Information Increase**

The rates at which formal and informal information enter an information space vary during the different phases. The vertical axis represents a conceptual coverage rather than some physical measure, such as storage consumption. Formalized information enters the system more rapidly during the seeding and reseeding phases than during the evolution through use phase while the influx of informal information is lower during reseeding than during the other phases.

phases of the design environment evolution. Most importantly, for this is where the problems discussed in Chapter 2 are most pervasive, incremental formalization can support designers in adding or formalizing information during the evolutionary growth phase. But incremental formalization can also support the initial creation of the seed, and can help with the reorganization and formalization that occurs during reseeding.

### 4.6.1 Seed building

Building the seed for a design environment primarily occurs in the traditional manner of knowledge engineers observing and interviewing domain experts to gain domain information. The knowledge engineers build knowledge bases based on the observations and discussions which are then evaluated and refined with the help of the domain experts [Waterman 86][Hart 88].

Incremental formalization's support for this knowledge acquisition combines the benefits of tracking informal information during knowledge engineering with the benefits of knowledge-based support found in automated knowledge acquisition tools. Hofmann et al. realized a number of benefits from the tracking of informal information during knowledge engineering: integration of source information, improved communication with domain experts, and availability of knowledge sources for formalized information [Hofmann et al. 90]. These benefits point to the need to not just keep the informal information for the initial knowledge acquisition process during seeding, but during the whole life for use in further knowledge base revision during evolutionary growth and reseeding.

Work in automated knowledge acquisition implies that the active support for incremental formalization can aid in the initial seeding. Automated knowledge acquisition tools have found that suggestions for knowledge-base refinements can support the "tuning" of subtle interactions within knowledge bases using models of the domain [Davis 84] and statistical mechanisms [Politakis, Weiss 84]. As with the active

support for incremental formalization this work leaves the user in charge but makes suggestions to support modification of the knowledge base.

### 4.6.2 Evolutionary growth

This phase is when the system is in actual use and when the domain workers using the system are modifying the knowledge base without the help of knowledge engineers. This is also the phase when the problems discussed in Chapter 2 are most likely to influence the system usage. The impact of incremental formalization on these problems was discussed earlier in this chapter.

Beyond these general advantages, during the evolutionary growth phase incremental formalization enables the addition and partial structuring of information which can be further formalized during reseeding. If the system required formalized information to begin with then there would be less information available for the reseeding process. By enabling the formalization of the information, users can augment the seed with formal knowledge when their task demands it. Also, partially formalized information will provide the knowledge engineers cues as to what was important about the information.

### 4.6.3 Reseeding

Reseeding has the goal of having professional knowledge engineers help to organize, generalize, and formalize the information added to the information space during the use phase. This process can be seen largely as a knowledge engineering task in many ways similar to the initial seeding. As such the same arguments that hold for the usefulness of incremental formalization during seeding are applicable during reseeding.

Differences between seeding and reseeding concern the characteristics of the information and interactions with the users. During reseeding the task of the knowledge engineers is to incorporate information added during some particular situation. The possibility of limited knowledge of the situation and little or no access to the author can make this task more difficult than the original task of seeding. In these cases the active support mechanisms provided for incremental formalization can help with locating related information scattered across the information space.

## 4.7 Summary

Domain-oriented design environments support users engaged in design activities. These systems include a number of knowledge representations which vary in their degree of formality. The evolutionary nature of design has led to different mechanisms for enabling modification of the design environments' information spaces. Design environments acquire their domain knowledge through a process of seeding, evolutionary growth, and reseeding.

Information in both informal and formal representations is added during seeding, evolutionary growth, and reseeding, although this will likely be at different rates in the different phases. To better support this acquisition of information, incremental formalization should take place throughout the process of seeding, evolutionary growth, and reseeding. Mechanisms which enable and support incremental formalization need to be available throughout the different phases for use by both knowledge engineers, in the seeding and reseeding phases, as well as design environment users.

# *Chapter 5: The Hyper-Object Substrate*

In designing a system that enables incremental formalization, two considerations of primary importance are (1) that the system be able to support representations of varying degrees of formality and (2) that interactions with less formal information not be made more difficult because of the system's support for formal information. This chapter describes the Hyper-Object Substrate (HOS), a combination of hypermedia object mechanisms (such as those built into systems like the Virtual Notebook System [Shipman et al. 89], and prototype-based knowledge representation languages (such as SELF [Ungar, Smith 87]). In particular, this chapter discusses the properties of HOS related to the incremental formalization of information. More general information on HOS may be found in the HOS Users' Manual [Shipman 93].

The first section discusses why hypermedia systems provide an appropriate starting point to building a system enabling incremental formalization. The next topic is the need to use so called "first-class" objects in a system allowing the incremental formalization of information. Next is a discussion of removing or hiding many traditional system-oriented concepts in representation languages. This is followed by a discussion of the static and dynamic navigational mechanisms in HOS, including bookmarks and agent objects. Finally is a description of the storage mechanism that enables semi-synchronous interactions between concurrent users of HOS. The chapter concludes with a discussion of related work on integrating formal and informal representations.

## 5.1 Hypermedia as a Starting Point

Hypermedia [Conklin 87] is characterized in the introduction and in Chapter 2 as chunks of informally represented information connected by formally represented links. This description of hypermedia is useful for discussing formality in representation. Other characterizations emphasize additional aspects of the hypermedia research field. Such as:

> "a combination of natural language text with the computer's capacity for interactive branching, or dynamic display ... of a nonlinear text ... which cannot conveniently be printed on a conventional page." [Nelson 67]

Nelson, the originator of the term "hypertext," emphasizes how a hypertext is different from a standard text or book; a hypertext uses the computer to provide a dynamic, non-linear structure to the text.

In his keynote address at Hypertext '91, Halasz provided a definition of hypermedia to reflect the diverse nature of the research being done within the hypermedia community:

> "Hypermedia is a style of building systems for the creation, manipulation, presentation, and representation of information in which:
>
> - the information is stored in a collection of multi-media nodes
> - the nodes are explicitly or implicitly organized into one or more structures
>   commonly, a network of nodes connected by links
> - users can access information by navigating over or through the available information structures." [Halasz 91]

The combined emphasis on interface ("creation, manipulation, and presentation") and representation of information ("network of nodes") mirrors the primary considerations in designing HOS for incremental formalization. This emphasis, along with the ability of hypermedia to "explicitly or implicitly" structure information, makes hypermedia an appropriate starting place for building a system to support incremental formalization.

A number of researchers in the hypermedia and AI communities have come to the conclusion that hypermedia can be of use in knowledge acquisition and representation. Russell summarizes his experiences working with hypermedia and AI knowledge representations as follows:

> "Knowledge representation has something to learn from the hypermedia experience. While formal representations are laudable, they seem inadequate to the real-world tasks of daily knowledge engineering. Not only must their semantics be completely specified, but their relative paucity of expression leaves them unfriendly for human use. Hypermedia representations offer [sic] contain a richness of content that is desirable to support human users, while simultaneously allowing reinterpretation when a new interpreter becomes available, or when a different perspective on the knowledge is required." [Russell 90] (page 8)

Similarly, Marshall describes hypermedia's usefulness for addressing representation problems:

> "Hypertext systems provide tools to view and manipulate structure as well as content, thus supporting the move from textual descriptions to emergent forms and abstract structures. The ability to work with unstructured information in conjunction with formalized, systematically organized information is the chief advantage of using a hypertext system rather than a knowledge representation language or a database description language. Systematic structures and expressions of content can be introduced and manipulated without the constraints of a formalism. Examples can be collected and analyzed, and structure can be created and imposed as general patterns are understood." [Marshall 87] (page 254)

Thus HOS is not unique in its use of hypermedia to support degrees of formality and incremental formalization. Although hypermedia can support the creation of structure, it does not intrinsically support the utilization of that structure beyond its use for browsing information. For this reason, HOS combines the features of hypermedia systems with features of object-oriented databases and prototype-based representation languages. Figure 5 diagrams the role of HOS as a layer in the creation of domain-oriented applications, such as design environments, and built on top of C, the X Window System, UNIX, and the Motif toolkit.

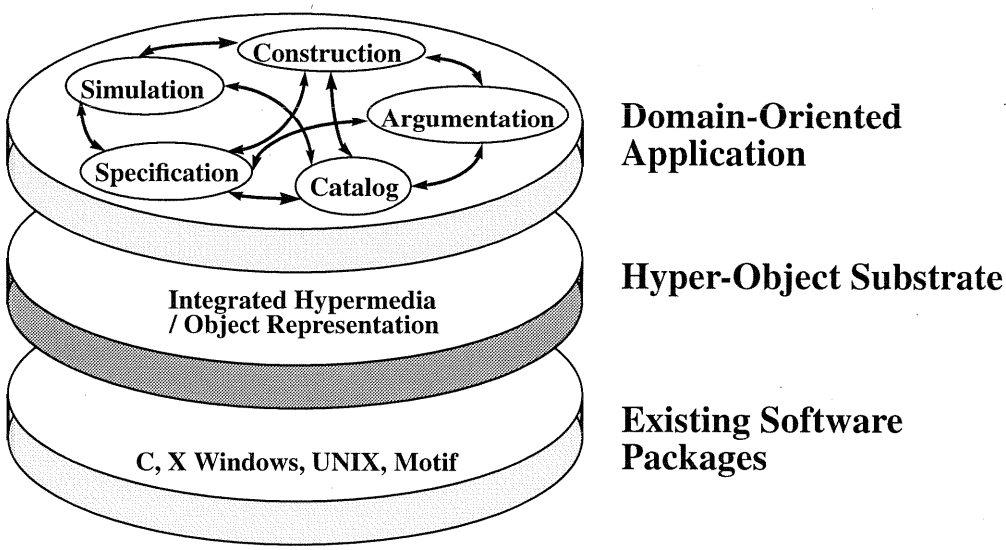



**Figure 5. Role of Hyper-Object Substrate**

The Hyper-Object Substrate (HOS) is a toolkit for building interactive information-oriented applications. HOS provides a persistent object system which is a hybrid of hypermedia and knowledge-based object systems.

## 5.2 First-Class Objects Allow Incremental Formalization

HOS integrates representations of varying degrees of formality by insuring that all information in the substrate (informal text, semiformal design rationale, and formal knowledge including objects with attached properties) is represented as *first-class objects*. A first-class object is an object that has no restrictions placed upon it, i.e. there is no precedence with respect to the type of objects in HOS's reasoning mechanisms. Objects in HOS are first class in that every object can have an unlimited number of attributes, can be referred to by other objects, and take part in inheritance relations.

HOS does include a number of object types. The object types within HOS are text-graphic, composite, view, agent, and shell. *Text-graphic objects* are objects which contain a drawing method which describes the display of that object. A *compound object* is a set of other objects which can have interactive and conceptual properties as a group. A *view object* is a resizable finite two dimensional plane which may have any number of the other types of objects displayed in the plane. *Agent objects* may have dynamically computed displays and actions based on information in the current object space. *Shell objects*, discussed more in Chapter 7, provide an interface to information available to the Unix shell. Text-graphic, composite, agent, and shell objects can be moved or copied between views and can be displayed in multiple views at once.

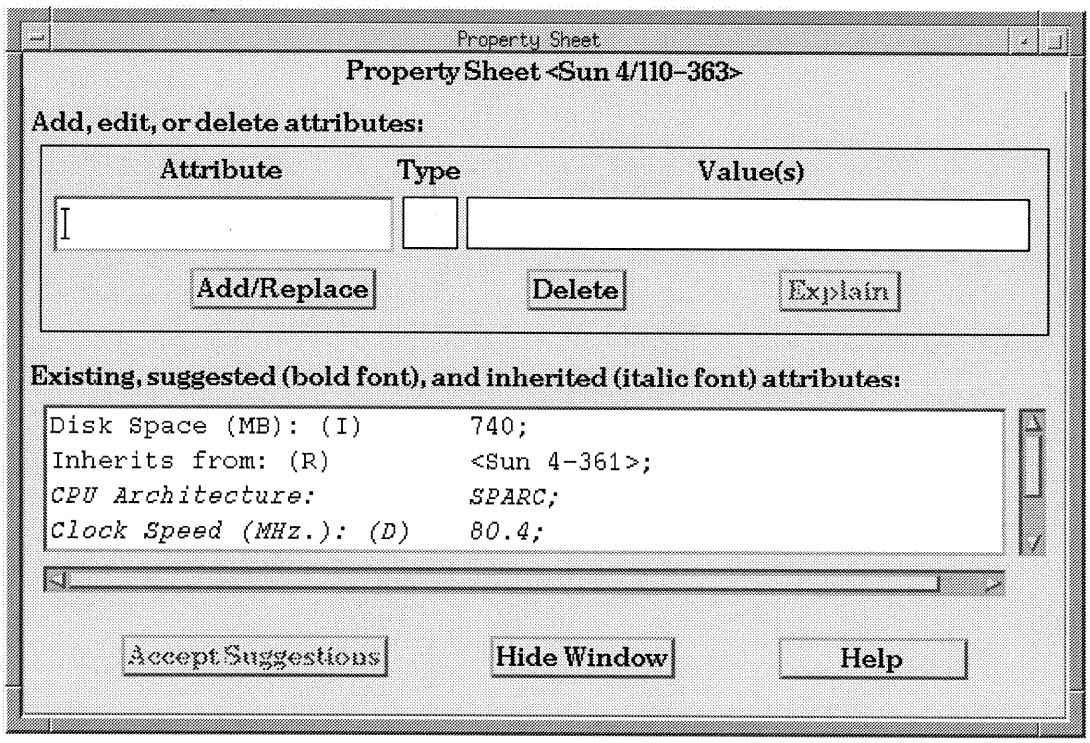



**Figure 6. Property Sheet in the Hyper-Object Substrate**

Each object in the Hyper-Object Substrate, whether a text-graphics object, compound object, view object, agent object, or shell object has attributes and values which can be edited in a property sheet. Attributes printed in italics indicates that the attribute is inherited.

All of these object types have an extensible set of attributes and values. Attributes and values of objects can be created and edited in a property sheet, as seen in Figure 6. Attributes may be of type string, decimal, integer, or relation. Attributes of type *relation* have references to other objects as values and are used to represent relationships between objects.

HOS supports inheritance of attributes between objects. The second attribute in Figure 6 shows that the object being inspected in the property sheet, Sun 4/110, inherits attributes from the object Sun 4. Inherited attributes are shown in italics and follow any locally defined attributes in the list provided by the property sheet. Any object can inherit attributes from any other object; this will be discussed in greater detail in the next section.

The importance of first-class objects comes from the need for the Hyper-Object Substrate to provide the low-level functionality required to support the incremental formalization of information. In HOS, an object whose only content is a piece of text has the representational potential of every other object in the substrate. The text object, like all other objects, can have any number of new attributes added to it and can be referenced by every other object's attributes.

## 5.3 Removing System-Oriented Distinctions

One of the major differences between HOS and many other representation systems is HOS's emphasis on supporting users who may not be experienced knowledge engineers. The approach is to eliminate the artificial system distinction of class and instance, and to delay attribute type conflicts until resolution is unavoidable.

To avoid the class/instance distinction that most object systems include in their inheritance mechanism, HOS uses a variation of prototype inheritance [Lieberman 86]. Prototype inheritance does not distinguish between any objects, meaning that any object may take any role in inheritance relations with other objects. By removing the distinction between classes and instances, the use of the inheritance mechanism no longer requires learning about these knowledge engineering concepts. The creation of objects that act as classes is still possible, although the system will place no restrictions on how such objects can be used.

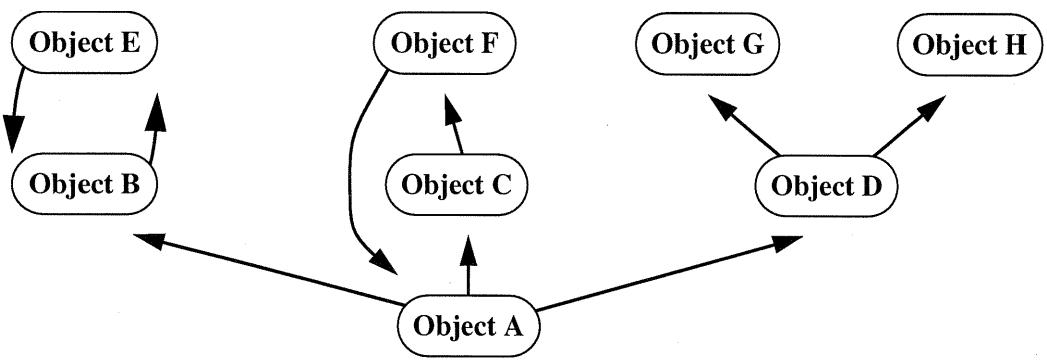



**Figure 7. Example of Cycles in Inheritance Graph in HOS**

Inheritance graphs in HOS can contain cycles, as in the graph shown above. Resolution of inherited values is carried out in a breadth first search through the inheritance links.

Another difference between the inheritance mechanism in HOS and that in many other object systems is that HOS's inheritance relations are allowed to form generic graphs; there can be cycles in the inheritance

graph. Cycles can be used to create equivalence classes of objects--each object in the cycle will have the same set of attributes. If an attribute is defined in multiple places in the cycle then not all objects in the cycle will necessarily have the same value for that attribute.

Attributes and values are concepts which are in some sense hidden from the user until needed. Objects are normally created in HOS without any attributes or values, unless the object is a copy of an existing object which has attributes and values. With a single mouse click the user can create a new object and begin typing right on the view. In this way the system can be used as a hypermedia system without learning about the more formal aspects of HOS's representation language.

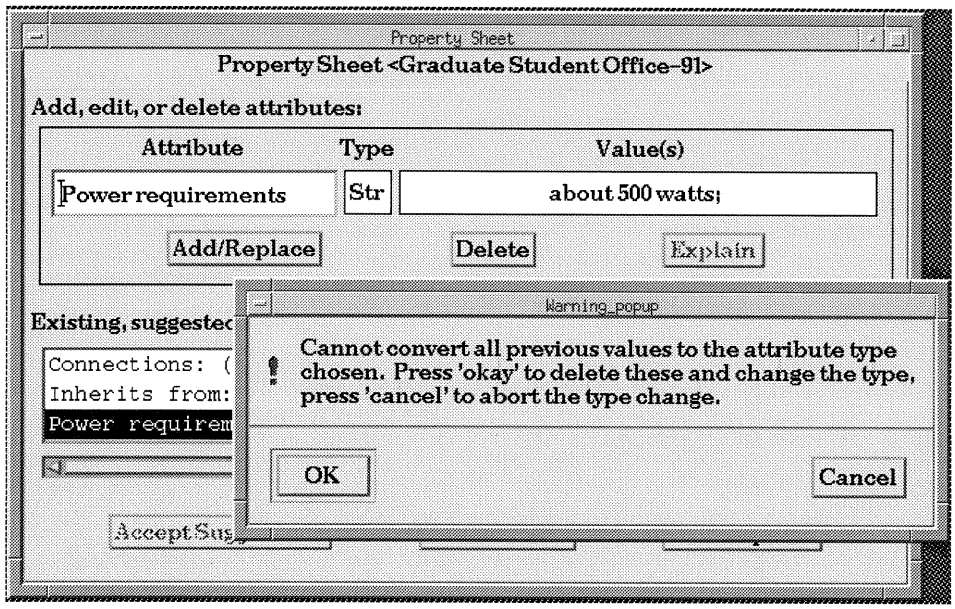



**Figure 8. Message that Type Conversion of Values is not Possible.**

When a user changes the attribute "Power requirements" from type string to type integer a warning appears telling the user that the existing value cannot be automatically converted and will be deleted if the type change is completed.

Similarly, attribute types are another example of a knowledge-engineering concept that HOS hides until necessary. When attributes are initially created they default to type string, thus allowing any value to be entered. Only if a type is selected does the attribute check to see whether its values match its type. When a new type is selected for an attribute with previous values the system attempts to automatically convert the values, such as from the string "12" to the decimal value twelve. When such automatic transformations cannot be made the system warns the user that some values could not be converted to the new type and will be removed if the user decides to continue. Figure 8 shows a property sheet and warning popup after the user has changed the type of an attribute whose values could not automatically be converted to the new type.

## 5.4 Information Navigation and Location

A type of formalism that is found in almost all hypermedia systems is the navigational link. Links, sometimes called hyperlinks, provide pathways which readers can follow to browse the information; the

readers traverse links that of interest to them. Each object may have a navigational link to a HOS view object--only view objects may be the destination of links. Clicking on an object with a link (signified in HOS by a hollow circle next to the object) causes the linked view to be accessed and displayed.

Many information systems, especially hypermedia systems, require ways of quickly getting back to information previously located. In systems that include hypermedia browsing, users often decide they want to traverse a different link pathway. Two approaches to facilitate this have been investigated in hypermedia systems: automatically keeping a history of nodes visited that the user can back-up through [Akscyn et al. 88], and enabling the user to create "bookmarks" to point to information that they feel they will want to retrieve again [Walker 87].

---

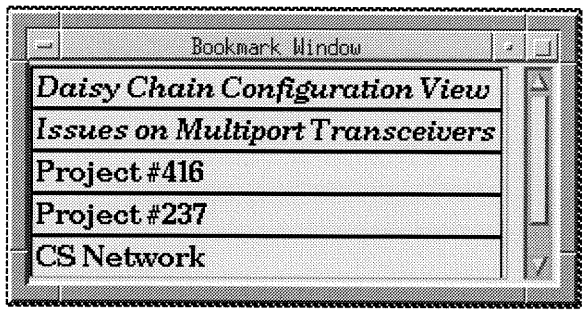



**Figure 9. Bookmark Window in HOS**

The bookmark window shows the names of views that have been marked by the user or suggested by the system.

---

HOS includes both of the above methods for enabling users to return to previous views. The bookmarks in HOS are somewhat unique. They have the basic functionality; HOS bookmarks refer to view objects by name and are displayed in a special bookmark window, as shown in Figure 9. Selecting the name of a view in the bookmark window displays that view. This interaction method means bookmarks can provide an easy way to switch among views by having both views listed in the bookmark window. Different views may represent, for example, different perspectives of a task. HOS's bookmark mechanism differs from other systems using bookmarks in that HOS includes the ability for the system to automatically suggest bookmarks. These suggestions appear in the bookmark window, along with the user-defined bookmarks. Figure 9 shows two system suggested bookmarks with three user bookmarks. The system-suggested bookmarks are distinguished by being displayed in italics.

The combination of browsing and bookmarks can support location only up to a point. When an information space gets large (hundreds of views) or disorganized, locating information by browsing alone is no longer acceptable [Fischer, Reeves 92]. A query mechanism has been included in HOS to allow more direct information location strategies. Queries can be used to locate views or select objects within a view by locating objects within the information space that match chosen attributes and values. The combination of the many information location mechanisms provides the user with flexibility in information location strategies. Being able to use a query or a bookmark to get close to the information, and then browsing from there to locate the information is an option the HOS user has that is not available in strictly browser-based or strictly query-based location mechanisms.

## 5.5 Creating Dynamic Information Spaces

The static nature of most hypermedia information spaces makes the application of these systems to tasks requiring adaptiveness of the information space difficult. The query mechanism described above can partially solve this problem. Queries replacing static information support "virtual structures" [Halasz 88]; the information displayed by virtual structures is determined by evaluating the stored query when the virtual structure is accessed by the reader, rather than when it is authored. Virtual structures are supported in HOS through a type of dynamic object, called an agent object. Agent objects, or simply *agents*, allow for a variety of dynamic behavior.

Agents search for objects with certain attributes within the system and perform operations based what, if any, objects they locate. Agents consist of a trigger, a query, and an action. This representation, similar to the representation of agents in OVAL [Malone et al. 92], provides flexibility in specifying the delegation, interruption, and control characteristics for individual agents. The trigger specifies when the agent evaluates its query. Any objects that are returned by the query are passed to the action, otherwise the action is not performed.

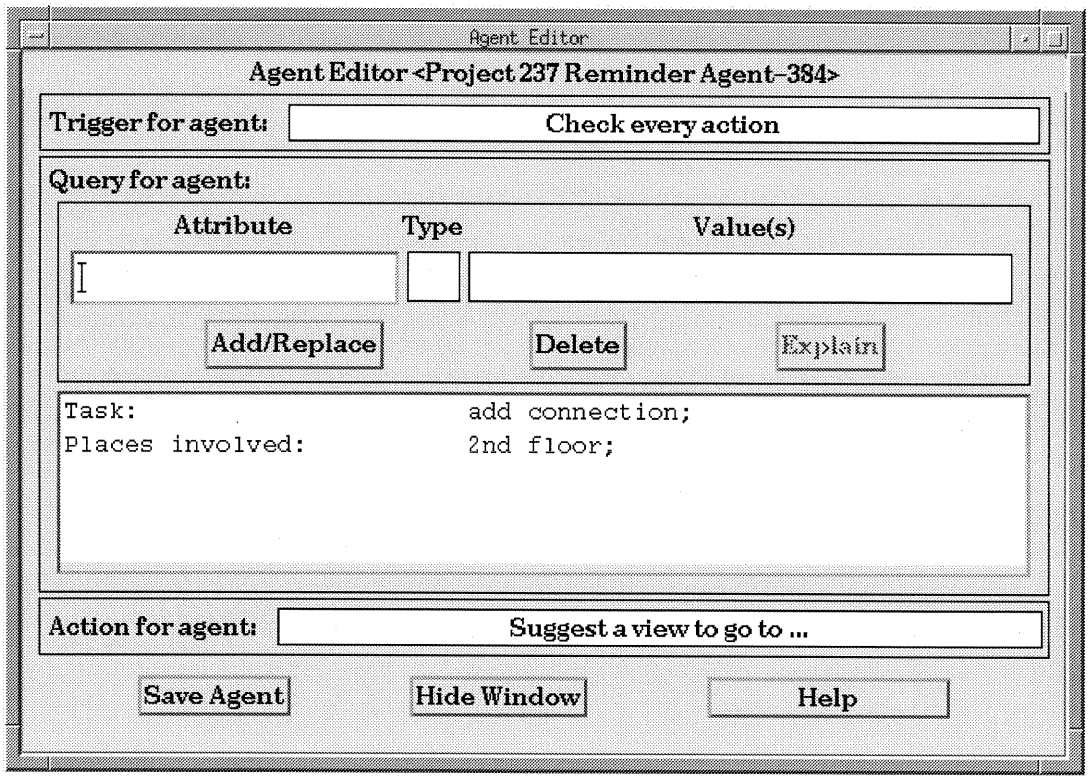



**Figure 10. The Agent Editor in HOS**

Users define agents through the agent editor. The top part of the interface is where the user chooses a trigger from a popup menu of choices. In the middle the user defines the query and the lower part is where the designer specifies the action to perform if objects are returned by the query. The "Project 237 Reminder Agent" is currently being edited. This agent will check every user action watching for an object to be changed that has the both of the attributes and values defined. When found it will put the project 237 view in the bookmark window.

Users can create or modify agents in the "agent editor" shown in Figure 10. The agent editor provides a set of triggers and a set of actions from which the user may choose. After selecting a particular trigger or action, the user is asked to specify extra information needed for that trigger or action. For example, when a user selects the action to "present a message" or "add a suggestion to the bookmark list" the system asks what message should be displayed or what view should be added to the bookmark window.

### 5.5.1 Triggers: controlling the activity

The first component of an agent object is the trigger. The trigger defines when an agent is active, i.e. when it will evaluate its query. As such the trigger defines the agent's control characteristics--whether the agent will interrupt the user or not. HOS includes the following options for triggering: (1) check every action of the user, (2) check only when requested by the user, (3) check when the agent is displayed, and (4) check immediately.

Agents which check every user action can be used as "demons" to watch for certain information to become available or for certain situations to occur in which they will act. Agents which only check when requested by user are guaranteed not to interrupt the user but will not be able to help the user if the user does not know to ask for help. This trigger is also likely to be appropriate for actions which produce side-effects such as modifying the database of objects.

Agents, like all objects in HOS, may be displayed in view objects. Agents which check when displayed evaluate their query and action when a view they are part of is to be presented to the user. This trigger can be used to create virtual structures based on the current state of the information space. The check immediately trigger means that the agent will be activated immediately upon creation of the agent and also implies that the agent will not be saved in the database. This is appropriate when the user only wants to perform some query or action once or is in the process of incrementally formulating a query [Fischer, Nieper-Lemke 89].

### 5.5.2 Queries: looking at the current situation

The second component of a HOS agent is a query. The query defines the information that must be located before the agent will execute its action. The query definition area within the agent editor is similar to the property sheet used to attach attributes to objects in the information space. An implicit conjunction is used when multiple attributes or values are defined in the query section. This interface limits the expressiveness of the query to the location of objects matching attribute patterns but allows the transfer of skills acquired in using the property sheets.

Use of a more powerful query language based around a hypermedia model, such as that found in HERMES [Stahl 93], would enable greater expressiveness but with the added cost of the users being required to learn the syntax and semantics of the formal language. Beyond such traditional query mechanisms, queries definitions should also be allowed to use built-in primitives which do complex analysis, such as latent-semantic indexing (LSI) [Dumais et al. 88].

### 5.5.3 Actions: advertisement and collection

The final component of a HOS agent is an action. The trigger and query together determine whether an action will be taken. The action defines the support service that the agent will provide to the user. In HOS the options for actions are (1) select/highlight found objects, (2) present a message to the user, (3) create a system-suggested bookmark, and (4) collect found objects in current view.

The action to select and highlight objects found by the query is likely to be used when defining a query to be immediately executed. Agents which present a message to the user create a dialog box displaying the

message to the user that must be acknowledged before HOS will continue. This level of interference will not be appropriate for all messages. The agents which create a system suggested bookmark do not interrupt the user, but place a new item in the bookmark window. The collect objects in the current view action is used to define agents which act as virtual structures or to collect information scattered across the information space and provide a new view of that information.

## 5.6 A Storage Mechanism for Semi-Synchronous Work

Another difference between HOS and other object systems such as SELF [Ungar, Smith 87] is that HOS's objects are persistent. When an object is created, it is stored and updated in an object database on disk. Database activities, such as space allocation, are transparent to users.

One purpose of the database in HOS is to ensure the integration and communication between different parts of HOS's interface. The importance of an integrated interface was also discussed as one of the goals of the Human Interface Tool Suite:

> "To act collaboratively, an interface must be integrated. Events and objects in one part of the interface must be accessible to the other parts so that tasks can be split between interface components as appropriate and still function with users in a collaborative and integrated fashion." [Hollan et al. 91] (page 294)

Because all parts of the HOS interface must store and retrieve information through the database, all information is available to all parts of the applications of HOS. Flexibility of the underlying representation language is crucial for creating a centralized storage mechanism which can support the different representations required for different domains.

Persistence of objects in HOS also enables *semi-synchronous* communication, or "not quite real-time" communication. HOS uses a combination of a write-through cache and a polling mechanism to communicate modifications between concurrent applications. The users of HOS can define database polling characteristics so that, at certain intervals, HOS rereads objects already in the cache to check for updates. When multiple users are working with the same HOS object database simultaneously, the objects in HOS's cache may become out-of-date for a period of time up to the polling interval. Figure 11 shows the polling interval to be set by the user.

HOS's polling characteristic enables semi-synchronous work by multiple users of a HOS. After each polling cycle the users' views of information are updated to show other users' changes to the information space which have visible effects. While such semi-synchronous communication is not appropriate for all tasks [Rein, Ellis 91], it does enable multiple users to simultaneously access and modify an information space with little overhead.

The granularity of database access is at the object level; when a user action causes a change in an object, the whole object is written to disk. Changes to compound objects or view objects do not modify their component objects. This limited effect avoids the necessity of including a locking mechanism. The result of not having locking is that users may write over other users' concurrent changes, but the small granularity of objects in HOS makes the frequency of such collisions low.

## 5.7 Related Work

Two main areas of research relate to the Hyper-Object Substrate: hypermedia systems which include formal representations and knowledge representation languages from artificial intelligence researchers which include informal representations. Whether coming from a AI background or from a hypermedia background, the integration of formal knowledge representations with hypermedia representations has been investigated by a number of projects.

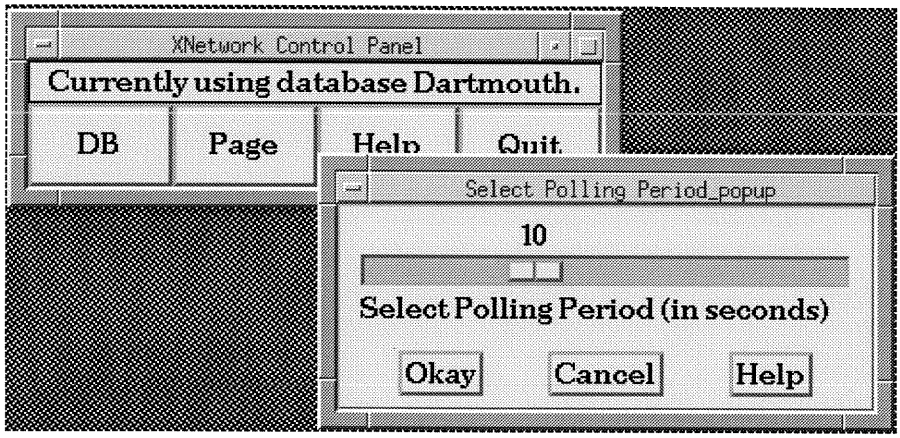



**Figure 11. HOS Control Panel and Polling Frequency Dialog.**

The control panel displays the name of the database in use. HOS allows the user to select the database polling frequency, which determines the frequency of passing changes between concurrent users of a database.

### 5.7.1 Hypermedia systems including formal representations

A number of researchers have looked at the relationship between formal representations and hypermedia systems. Both RelType [Barman 91] and MacWeb [Nanard, Nanard 91] use 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 [Kaindl, Snaprud 91]. An example of the need for less formal representations is Schwabe and colleague's decision to add a hypermedia component to a tool for creating Prolog encodings of engineering knowledge [Schwabe et al. 90]. In this project they found it desirable to provide a less formal representation than Prolog for the knowledge engineers engaged in the process of structuring and translating written engineering knowledge. Two systems discussed below, NoteCards and KMS, do not include much of a formal representation but are described to provide comparisons for some of the other systems described.

**NoteCards.** NoteCards [Halasz et al. 87] is a prototypical hypermedia system using a node and link model for representation. Nodes in NoteCards are typed as to what type of information they include, such as text, image or a list of other cards. Links to other NoteCards can be embedded in a piece of text or in an image. By selecting a link with the mouse the card at the other end of the link appears.

**IDE.** The Instructional Design Environment (IDE) is a system built on top of NoteCards and includes a number of "structure accelerators" to make structuring information easier [Jordan et al. 89]. IDE extends NoteCards by including the ability to define template cards, a type of structured form in which the user can fill in blanks instead of having to recreate the form. These forms act as a set of typed slots, each slot being able to be filled by a particular card or form type. Filling in these forms is accelerated by the use of autolinks, which create the appropriate type of card to fill a particular slot and add the link between the cards.

**KMS.** The Knowledge Management System (KMS) [Akscyn et al. 88] chose a slightly different representation by making nodes be approximately the size of a 8 1/2 x 11 inch piece of paper. Each KMS

node can have a number of text, image, or active objects on it. Each object has a number of attributes defining its presentation and can also be linked to a node, which will appear when that object is selected.

**Virtual Notebook System.** The Virtual Notebook System (VNS) [Gorry et al. 91] is based on a slightly more abstract representation. Like KMS, VNS nodes (called pages) default to the approximately the size of a piece of paper (although they can be resized to be any size) and can contain any number of objects. From the representational view the main difference between VNS and NoteCards and KMS is that objects and links in VNS exist separate of pages and can exist on any number of pages at a time. Another abstraction in VNS is that all representational types (i.e. objects, links, etc.) can have any number of user defined attribute/value tuples associated with them [Shipman et al. 89].

Experience in aiding in the design and implementation of VNS led to the interest in providing an even more expressive representation, instantiated in HOS and in supporting the use of this representation. HOS differs from the representation in VNS in that all information types are objects, objects can have attributes whose values point to other objects, objects can inherit attributes from other objects, and will be able to contain functional methods.

**SPRINT.** One of the first hypermedia systems to include inheritance and methods was SPRINT [Carlson, Ram 90]. SPRINT assists corporate managers in explicitly representing mental models as a network of associations among the elements of a strategic plan. In order to provide this type of support Carlson and Ram propose to use a frame-based representation to integrate hypermedia, semantic network, and expert system representations. This representation includes the attachment of SmallTalk methods to frames to provide the system with computational ability within and computation over the represented information.

The addition of methods and inheritance in SPRINT extended the representational expressiveness beyond that in VNS. A question mentioned by Carlson and Ram as being unanswered in their work is whether corporate managers would really use all the formal representational power provided by the system. The main contribution of SPRINT was in the integration of representations, not in usability of such representations. The topic of usability and how it can influence the design of integrated knowledge representations will be discussed with respect to HOS in the next chapter.

**CONCORDE.** The system CONCORDE [Hofmann et al. 90] is a hypermedia system designed to support the process of knowledge acquisition for expert systems. CONCORDE contains an object-based representation language for collecting domain information into hypermedia nodes and creating relation types that are specially adapted for the domain being modelled. This includes mechanisms for constraining relations between nodes and mechanisms for knowledge engineers to specify interactions for the system and the domain experts to collect their knowledge.

While the goal of supporting knowledge acquisition is closely related to the goal of supporting incremental formalization, CONCORDE lacks any active support for formalization. CONCORDE assumes that a professional knowledge engineer will be structuring the 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.

**Hoopertext.** By being a platform for building other applications Hoopertext [Berlin, O'Day 90] has some different goals than the hypermedia systems previously described. This goal makes Hoopertext comparable to HOS. Both Hoopertext and HOS are meant as starting points for applications addressing more specific problems and both use a variation on object-oriented approaches for knowledge representation, although neither are strictly object-oriented due to implementation considerations.

Hoopertext has focussed on providing a sharable information space, placing more emphasis in the support of synchronous and semi-synchronous work than HOS. Hoopertext leaves the issue of how users will deal with the representations provided to be handled by the applications.

**PHIDIAS.** McCall's PHIDIAS system is designed to support designers using the Procedural Hierarchy of Issues (PHI) approach to issue-based argumentation. PHI includes three types of nodes: issues, answers, and arguments. It uses single a inter-issue relationship, the *serves* relation, and uses other links to connect answers to issues and arguments to answers. PHIDIAS is characterized by its support for CAD graphics integrated with the argumentation system [McCall et al. 90]. In PHIDIAS all objects in both the text and graphics portions of the system are part of a single node and link structure. PHIDIAS extends the PHI model by allowing graphics and PHIDIAS queries to be the contents of nodes. This use of the query language provides for the creation of argumentation dynamic in relation to the state of other sections of the argumentation structure and the state of the design in the CAD graphics subsystem [McCall et al. 91].

### 5.7.2 Semi-formal substrates and knowledge representation languages

There has also been interest outside of the hypermedia community in combining informal and formal representations. The topic of semi-formal representations is being looked at within the computer-supported cooperative work and knowledge representation communities.

**OVAL.** The usefulness of semi-formal information for supporting collaborative work was shown by Malone's Information Lens [Malone et al. 86]. Malone's most recent system, OVAL [Malone et al. 92], is characterized as being "radically tailorable", which means that OVAL can be adapted by the user to provide a wide variety of functionality. Like HOS, OVAL uses a type of object-centered representation language for encoding information and uses a template style of interface to define the slots for object types and to set the slot values for objects. Also, OVAL includes "agents" similar to HOS agent objects for adding dynamic behavior to the information spaces.

Unlike HOS objects but like Aquanet objects, OVAL objects are created as an instance of a type and cannot have new attributes added without changing the type definition. Another difference from HOS is that OVAL objects are limited to a set of alphanumeric slot values and there is no direct manipulation interface to OVAL objects. Instead, the user of OVAL interacts with objects in form-based interfaces that are based on the object type definitions.

**CODE4.** Informal representations have also become a topic within the knowledge representation and knowledge acquisition communities [Mundie 91]. One system built to allow informal information in knowledge bases is CODE4 [Lethbridge, Skuce 92a]. CODE4's representation allows the inclusion of everything from "sloppy English to formal logic" [Lethbridge, Skuce 92b]. 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.

## 5.8 Summary

The Hyper-Object Substrate (HOS) combines functionality normally found in hypermedia systems, persistent object bases, and prototype-based representation languages in order to create a flexible, domain-independent environment in which information can undergo incremental formalization. Table 1 summarizes the relationship between the conceptual framework and the functionality in HOS.

Hypermedia was chosen as a starting point in building a system enabling incremental formalization because of its combined emphasis on interface and representation. The hypermedia interface allows the authoring and use of informal information without concern for the formal capabilities of the system. The informal representation allows the user to enter information with the minimum overhead.

**Table 1: Relation of System Functionality to Conceptual Framework**

| System Functionality | Aspect of Conceptual Framework |
|---|---|
| hypermedia-style interface | ease of use for informal information, no overhead for informal because of formal capabilities |
| informal representation | problem of premature structure, minimize overhead for getting information into system |
| first-class objects | need for "in-place" formalization, non-destructive formalization, formalization on demand |
| flexible representation enabling structure evolution | problem of situational structure |
| removal/hiding of system concepts | lower "accidental complexity" and cognitive overhead |
| queries, inheritance, agent objects | formal information must provide benefits to user |
| suggestion mechanisms (described in chapter 6) | tacit knowledge, not knowing what to formalize, bootstrap formalization |

All information in HOS is contained in first-class objects, which each can collect attributes and take place in inheritance hierarchies. Objects that start out composed of mostly informally represented information may later be formalized without losing or undoing informal information. Formalisms added may later be modified to better match the user's current understanding and needs. System-oriented concepts, such as the class-instance distinction, were removed or hidden to reduce the level of knowledge-engineering understanding required for adding formalized information.

Systems must provide benefits based on the formalized information. Utilization mechanisms of formal information in HOS include inheritance, information navigation and location, and agents. HOS includes a variety of information navigation and location mechanisms. The combination of navigational links for browsing with bookmarks and a query mechanism for more directed searches provides flexibility. To provide more dynamic features to the HOS information spaces, agent objects can be used. Agents consist of a trigger, query, and action providing a variety of possible use and interaction characteristics. HOS includes a storage mechanism which enables concurrent access and modification to an information space. The communication of changes within the information space occurs semi-synchronously, at intervals determined by the users.

HOS differs from most other combinations of hypermedia and knowledge representation in its emphasis on the ability to add formalism gradually. Many systems use object types or classes to define what attributes an object will have. HOS's use of prototype inheritance enables all objects to have attributes added, modified, or deleted (including inheritance relations) at anytime.

# *Chapter 6: Tools Supporting Incremental Formalization in HOS*

As described in the previous chapter, the Hyper-Object Substrate (HOS) includes a representation and interface that enable incremental formalization. HOS also includes tools to support the user in formalizing information. This chapter divides these tools into two types: tools which support the process of formalization and tools which actively suggest formalizations.

## 6.1 Process-Oriented Tools

There are many types of tools that can help support the evolution of knowledge from less formal to more formal representations. Some are common to knowledge engineering environments-- providing interfaces appropriate for browsing through information, such as the Smalltalk object browser [Tesler 81], or maintaining consistency within knowledge bases [Ballance, Graham 91]. These tools help users manage formal information but do not support the use of informally represented information. Augmenting such knowledge engineering tools by using hypermedia has been explored by Hofmann and colleagues [Hofmann et al. 90]. They found this use of hypermedia aided communication between domain experts and knowledge engineers and provided context for the formal representations that was useful for later modification of the knowledge base.

Expanding on this role of hypermedia in knowledge engineering, HOS includes mechanisms which support the process of adding and formalizing information. Mechanisms for importing information from other on-line information resources support the user in initially adding information. Once the information is in the system, part of the problem of formalizing information is knowing what related information is already in the system. The process of locating related information is supported by the information location mechanisms provided in HOS.

### 6.1.1 Importing information from external information resources

For a system to become integrated into the users' computational environment it must assist in bringing information into the system that is already on-line. It is likely that for whatever task is being supported that there are a variety of computerized information resources containing domain-specific information already available, as was found in the case of biomedical research [Gorry et al. 88]. HOS includes mechanisms for importing standard text (ASCII) files since many systems, including domain-oriented ones, produce some type of textual representation of their information.

A problem with just importing text files is that the information loses any formality that it might have had before being turned into a text file. In order to bring in some formal information HOS can also import electronic mail messages and USENET News articles. These files include headers of formalized information, used to encode information such as sender and topic. When importing an electronic mail message or USENET News article the headers are parsed and added as attributes to the newly created object.

The attributes that result from the parsing of mail and news article headers are limited to being untyped, that is they cannot initially be typed attributes or relations. To show how such an imported message might be formalized over time this chapter will follow the evolution of an electronic mail message from the network design domain. By selecting "Import E-Mail Message ..." from the "Import" pull down menu an electronic mail message between network designers is imported into the a view of the network. Figure 12 shows the imported electronic mail message near the part of the network it is describing. The message

concerns using a Decstation with a thicknet communication card to provide a gateway to the undergraduate laboratory.

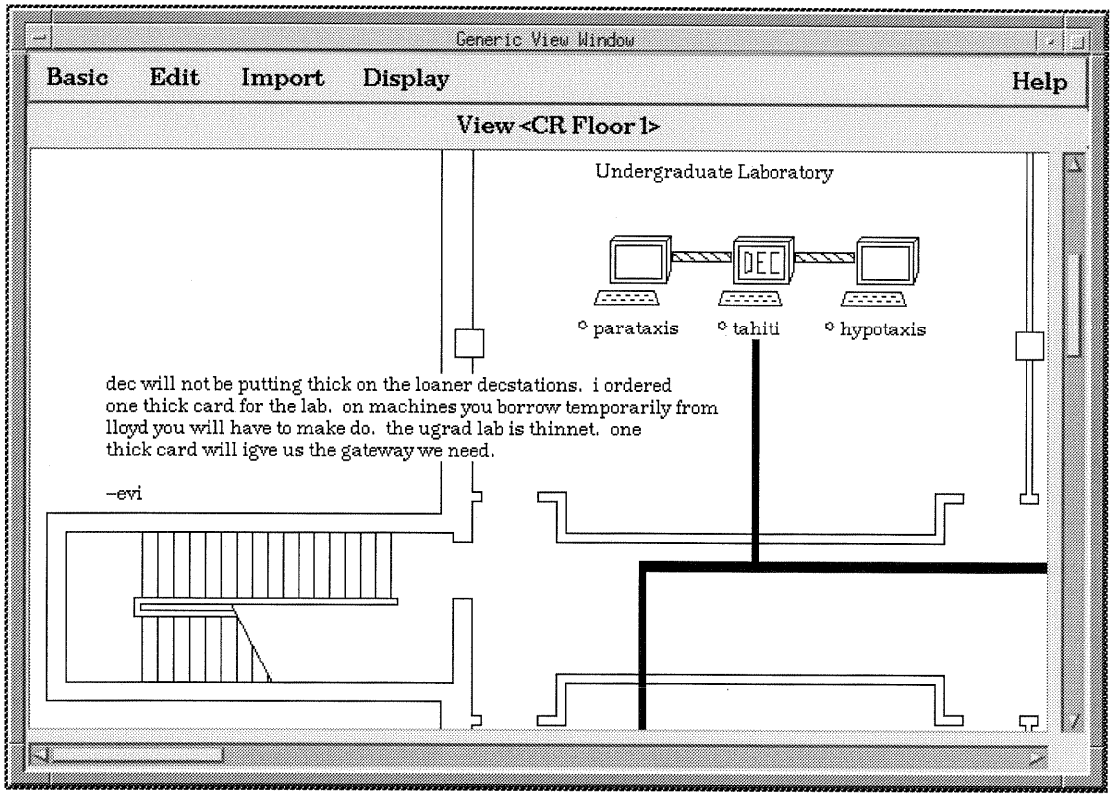



**Figure 12. Imported Electronic Mail Message**

Here an electronic mail message concerning the network design in this page has been imported into the system near the formally represented design that it discusses. The mail header of the message as been parsed and attached as attributes of the object.

### 6.1.2 Locating related information

The various information location techniques described in the previous chapter--hyperlinks, bookmarks, and the query mechanism--can be of use during the process of formalization. Locating information related to what is being formalized is part of the process of deciding exactly what and how to formalize.

To continue the example of the imported electronic mail message shown in Figure 12, assume the user is now looking for other Decstations with thicknet communication cards. To locate such objects the user can either navigate through the hyperlinks to views of the different labs looking for such machines or can use the query mechanism. Figure 13 shows a query defined to locate Decstations with thicknet cards and a matching object, highlighted in the lower-right corner of the page in the background.

## 6.2 Tools that Suggest Formalizations

The individual aspects of the process of formalization supported by the above tools are common enough tasks (importing and locating information) to be of general use in HOS. A different class of tool being

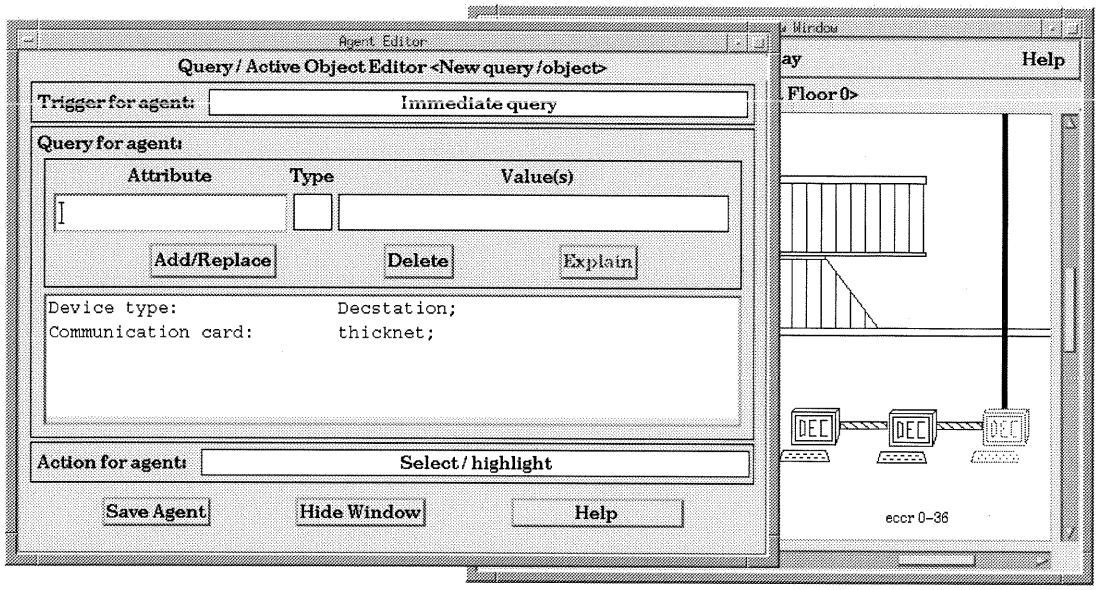



**Figure 13. Query for Decstations with Thicknet**

The query mechanism is accessed through the agent editor. Here the user has defined and executed a query looking for Decstations with thicknet communication cards. An object matching this description has been highlighted in the lower-right corner of the page in the background.

explored in the context of HOS make suggestions for formalizations based on the patterns within the information already in the system. These tools take a more active role than the tools supporting the process of formalization. Suggestions about possible formalizations, such as the addition or modification of attributes and relations or new hyperlinks, are based on formal information already in the system and patterns in informally represented information. These tools use the current state of the information space and so produce suggestions based on more information as the information space becomes larger. In this way these suggestions can help bootstrap the information space.

Suggestions need not be completely accurate to be of general benefit to users. By providing the users with suggestions the tools provide a starting point which can be edited, thus changing part of the process of formalization from creation to modification. The tools also provide an explanation of why each suggestion was made, providing users with rationale about the formalization process and about the specific formalization being suggested.

### 6.2.1 Suggesting attributes for text objects

The heuristic mechanisms in HOS used for making suggestions based on text have been kept simple, using free-text search rather than any real natural language processing. One suggestion mechanism, diagramed in Figure 14, uses free-text search in combination with a lexicon in order to suggest new or modified attributes and relations. The lexicon is created from the object names and their synonyms. When triggered, the mechanism looks for occurrences of the items in the lexicon within the text display or attribute values of an object. When a reference is found, a rule base is used to determine what attribute or relation is suggested based on characteristics of the object referenced.

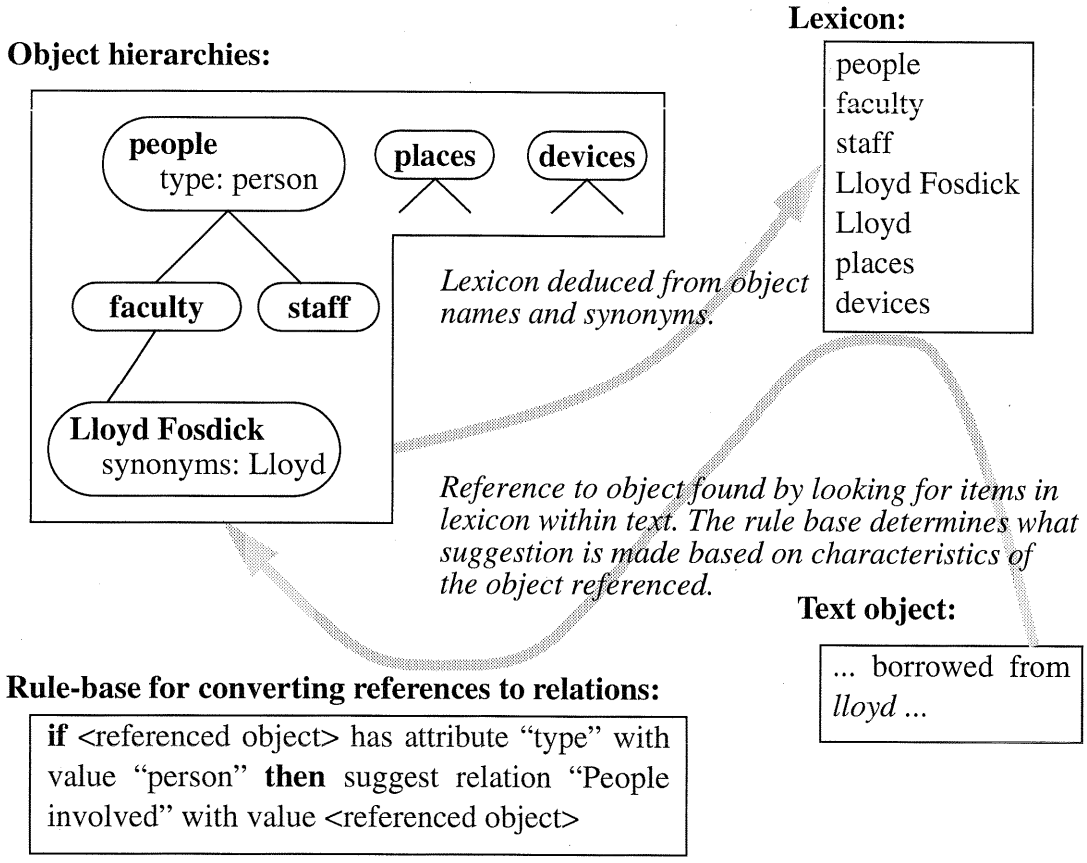



**Figure 14. Diagram of Relation Suggestion Mechanism**

The relation suggestion mechanism uses a lexicon derived from the objects in the HOS database to attempt to locate references within text to those objects. These references are turned into suggested relations using a rule-based mechanism.

The suggestion mechanism executes and displays its suggestions when the user views an object's attributes in the property sheet. Figure 15 shows the property sheet for the electronic mail message from Figure 12 just after it was imported. In this case four suggestions (the top four attributes listed) have been created. Three are based on possible references within the body of the text to people, places, and devices in the knowledge base. 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>" for the attribute with a relation to the object named "Evi Nemeth".

The property sheet uses of regular, italic, and bold fonts for local, inherited, and suggested attributes, respectively, in order for users to easily discern the types of attributes. As with other attributes, when a suggestion is selected it appears in the edit region at the top of the sheet. Suggestions can be accepted as is, modified and accepted, deleted, or just ignored. If the attribute being edited is one suggested by the system the "Explain" button becomes active, as it is in Figure 15. Explanations are generated by filling in information from the lexicon item, the rule base, and the object believed to be referenced into a script.

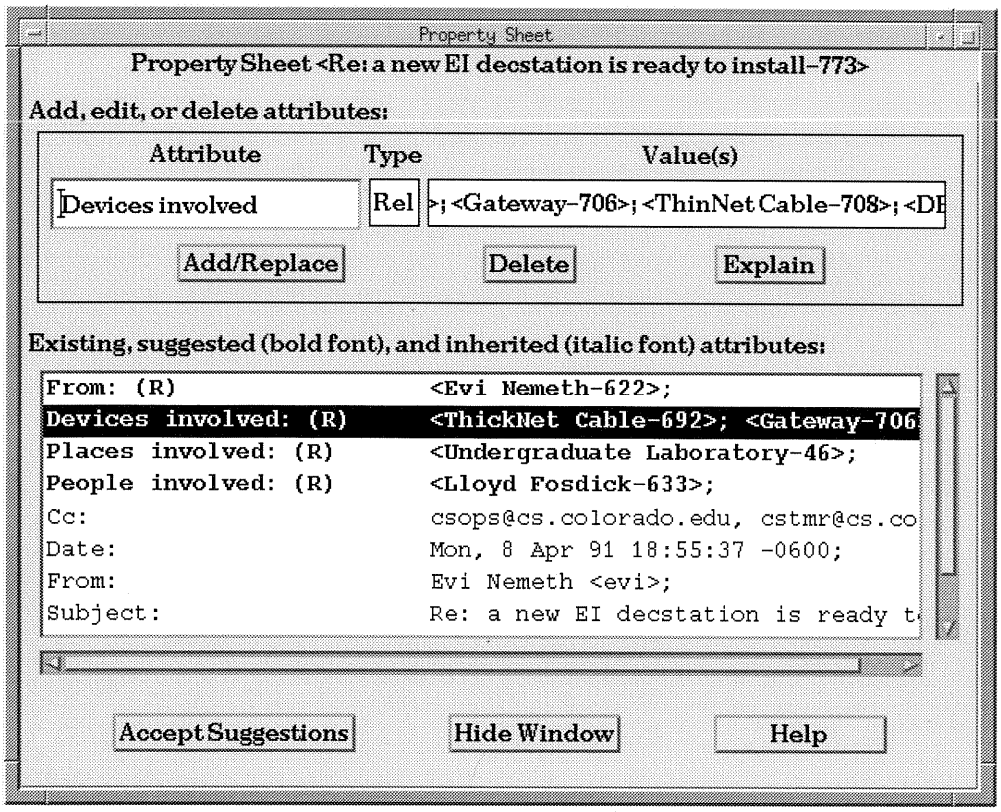



**Figure 15. Suggested Attributes in a Property Sheet**

This property sheet is for the electronic mail message concerning a computer network design shown in Figure 12. Four suggested attributes are shown in bold at the top of the attribute list. The other attributes are the result of parsing the electronic mail header. Here the system believes it has found references within the electronic mail message and the mail header to some people, places, and devices already in the information space.

### 6.2.2 Suggesting hyperlinks to related views

Another suggestion tool uses the agent object mechanism described in Chapter 5 to place views that seem to be related to the current view in the bookmark window. The bookmark window distinguishes between user defined and system suggested bookmarks by using different fonts, similar to the property sheet. Such bookmark suggestions can be used as suggestions for new hyperlinks. As with suggested attributes and relations, the user of HOS can get an explanation of why the system suggested any particular bookmark by selecting "Explain" from a popup menu attached to the bookmark.

Continuing the example of the electronic mail message, Figure 16 shows that the user has accepted the "Places involved" suggested relation. This new piece of information triggers a bookmark suggesting the view "Detailed view of Undergraduate Lab" as a hyperlink for the electronic mail message object.

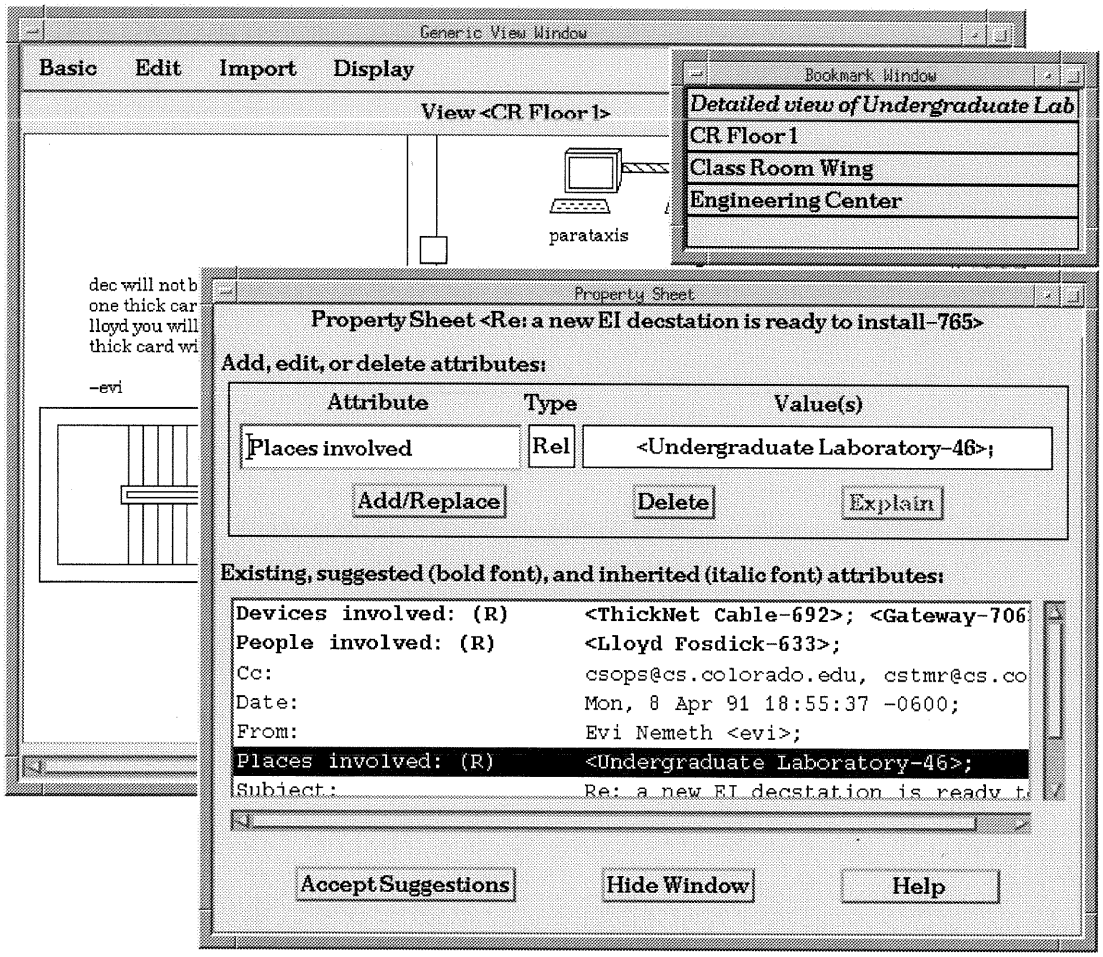



**Figure 16. Accepted Suggestion Leads to Another Suggestion**

After the user accepts the "places involved" suggestion shown in Figure 15 the system creates a system-generated bookmark for a detailed view of the undergraduate lab. The explanation for the bookmark suggests this view as a potential hyperlink. The accepted attribute is now listed among the local attributes.

### 6.2.3 General and specific suggestion tools

There are both domain-independent and domain-dependent suggestion mechanisms. The three new relations suggested in Figure 15 are produced by mechanisms which require a situation-specific lexicon containing the important people, places and things in a domain. In this case the "things" suggestion has been specialized to devices for the domain of network design. People, places, and things help to answer the who, where, and what questions. These questions are common across domains and so this aspect of the suggestion mechanisms could be considered domain-independent. The fourth suggestion in Figure 15, to change the textual value of an attribute to a relation to another object, looks for references to *any* other objects in the current information space and so is also domain-independent.

While not included in HOS because of the emphasis on it being a domain-independent substrate, mechanisms could be created which used domain-specific procedures for recognizing implicit structure.

The suggestion mechanisms in HOS are meant to support the process of incremental formalization. As was mentioned earlier, the suggestions may be accepted, modified, rejected, or just ignored. Suggestions are presented in a non-intrusive manner that avoids being disruptive to the users activities but makes them easily distinguished from non-suggested formalisms. Suggestions accepted or modified and accepted cause the information space to grow providing more information for later suggestions on which to be based.

## 6.3 Related Work

Work related to tools supporting formalization is being done in a number of different communities. Within the hypermedia community mechanisms are being investigated for automatically generating hypermedia links. Also, the areas of automated knowledge acquisition, and knowledge discovery are related to supporting the creation and location of formally represented information.

### 6.3.1 Hypermedia systems which automatically generate links

Not all hypermedia systems require links to be created by users. Some systems provide automatically generated links between pieces of information that seem related. Some of the algorithms used for the automatic generation of links in hypermedia are similar to the algorithms used by the suggestion tools, although how their results are used is slightly different.

Systems which advertise automatically generated links occasionally just provide access into a dictionary or other previously compiled reference source. Such a system might present a definition as well as encyclopedic information about a term selected by the user. These systems provide dynamic links into static documents. Some of these systems use a thesaurus and so will present information on synonyms as well as the word selected. This type of dictionary/thesaurus-based approach can be useful but does not solve the problem of how to connect pieces of information which were entered by the user.

A simple extension to the dictionary/thesaurus based approach does begin to provide connections within user created text. In the PC-based system SmarText the system provides connections between the use of the same word or phrase. In this type of system the user can select a word or phrase to locate other occurrences of that word or phrase. This system provides a type of automated "search for next/previous occurrence" without requiring the standard dialog box for search mechanisms.

SuperBook [Remde et al. 87] is a tool for information exploration that provides automatic indexing to provide an interaction style similar to that found in hypermedia systems. SuperBook creates a full-text index of a textual document which is used for locating occurrences of words, word stems, or boolean combinations thereof. The creation of the index is performed during a computationally expensive preprocessing step turning the plain text document into the representation needed by SuperBook. This is appropriate when a completed text exists and is being put "on-line" but is not as applicable to the interactive authoring of such documents.

A more holistic algorithm for suggesting related pieces of information has been created by Bernstein [Bernstein 90]. His system includes an apprentice which 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 the suggestion tools, makes suggestions to the user, leaving it up to the user as to whether accept the suggestion or not. This apprentice uses a shallow method of text comparison for providing suggestions due to a concern for computational efficiency.

### 6.3.2 Automated knowledge acquisition

Support for the creation of formally represented information has also been investigated in the area of automated knowledge acquisition. This area focuses on tools for knowledge engineers and possibly end-users, but is limited to supporting only formal knowledge representations.

Automated knowledge acquisition tools SEEK [Politakis, Weiss 84] and SEEK2 [Ginsberg et al. 85] support the refinement of production-rule based expert systems using statistical data to evaluate rule sets and to suggest possible modifications. While SEEK2 included an "automatic pilot" mode which would make successive modifications on its own, the systems were generally designed to support the user (presumed to be a knowledge engineer) in the refinement process. TEIRESIAS [Davis 84] uses a model-based approach to make suggestions to support knowledge acquisition. More recent automated knowledge acquisition systems, such as MOLE [Eshelman et al. 87], OPAL [Musen 89], and the HITS Knowledge Editor [Terveen et al. 91], improve on these earlier approaches through the use of a presupposed problem solving method, an explicit domain model, and cooperative problem solving respectively.

To reduce the time-consuming and difficult tasks of creating domain-oriented knowledge acquisition tools, such as OPAL, domain-independent *meta-level tools* have been created. Meta-level tools, such as PROTEGE [Musen 89] and DOTS [Eriksson 91], support the creation of domain-oriented knowledge acquisition tools by knowledge engineers. The domain-oriented knowledge acquisition tools, such as P10 and ALF-A [Eriksson 91], created are then meant to be usable by the domain experts.

End-user modifiability (EUM) [Girgensohn 92] also falls into the category of automated knowledge acquisition tools. This work focuses on helping the end-user of a knowledge-based application modify the formally represented knowledge in the application. The support provided by EUM tools is complimentary to the support provided by the tools to support formalization. Formalization tools support information being converted from informal to formal representations and EUM tools support the modification of formalized information.

All of these automated knowledge acquisition approaches are limited to supporting the use of formal knowledge representations and do not provide support for the transfer of knowledge between representations. Another difference from the formalization tools is that automated knowledge acquisition tools are most often part of the knowledge engineer's environment but not part of the application environment. The notable exceptions being the domain-oriented knowledge acquisition and end-user modifiability tools.

### 6.3.3 Knowledge discovery in databases

The area of knowledge discovery in databases has goals related to the tools suggesting formalizations included in HOS. One definition of knowledge discovery is "the nontrivial extraction of implicit, unknown, and potentially useful information from data." [Frawley et al. 92] This definition could equally well be used to define the approach used to come up with suggestions for formalizations. The algorithms used in knowledge discovery vary widely and are similar to those used to create formalization suggestions.

Despite the above definition, knowledge discovery in databases differs significantly from suggestion mechanisms in 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.

## 6.4 Summary

There are many types of tools to support the evolution of knowledge from informal representations to more formal representations. Some of the tools included in HOS are designed to support the process of formalization by providing mechanisms for importing information from other computational information resources and for locating related information in the information space. The aspects of the formalization task these tools support, such as information location, makes them applicable to other uses besides formalization.

Another class of tools included in HOS use the information already in the system to make suggestions about possible formalizations to add to the information space. The heuristic nature of the algorithms used to create suggestions means these tools will not be completely accurate, but this is not necessary for them to be useful to the user. The suggestions provide an "object to think with" for the user, which may be accepted as is, modified and accepted, declined, or just ignored. Suggestions in HOS are presented to the user in a non-intrusive but distinguishable form to avoid disrupting or confusing the user. Simple explanations of suggestions are also available to help the user in understanding the suggestions.

# *Chapter 7: Evolutionary Development of a Network Design Environment*

The Hyper-Object System (HOS) and the tools to support formalization have been applied in the creation of XNetwork, a domain-oriented design environment to support the collaborative long-term design and administration of computer networks [Fischer et al. 92]. The evolution of XNetwork in HOS, driven by a changing understanding of the problem of network design, provides one example of the evolution of knowledge-based systems in general. Building an environment for collaborative network design in HOS also exemplifies the usefulness of incremental formalization in combining communication with knowledge-based support in rapidly changing domains. A computer network design environment needs to enable network designers to add and modify formal information or else the system will quickly become out-of-date.

An analysis of the task of network design and administration is provided to motivate the design decisions made in the creation of XNetwork. This analysis combines a description of tools already used by network designers, characteristics of network design information, and observations of interactions between network designers. Following the task analysis is a discussion of what modifications were made in order for HOS to better support design in general and in particular the design of computer networks. Finally, there is a discussion of the creation of one generic and two case-specific information spaces on network design. XNetwork is the combination of HOS and the extensions specific to supporting design, with one of the network information spaces loaded.

## 7.1 Analysis of Network Design and Administration

Computer network design involves planning connections between devices such as workstations, file servers, and printers. The connections use cables of different lengths and types, supporting different networking software and data transfer capabilities. Networks can be viewed at different levels of abstraction. There is the level of individual devices and the level of subnetworks that are connected using bridges and gateways. On a larger scale, the local area network must be properly integrated into a nation-wide network such as the Internet.

Designers of networks must consider criteria such as cost, reliability, and extensibility. There is a large body of design rules of different flexibility. *Hard rules* are rules that must be adhered to. For example, CSMA/CD networks must have a hierarchical topology and do not function in a ring topology. Some rules are malleable. For example, although technical specifications require that RS 232 asynchronous communication lines are no longer than 75 feet, experience has shown that in normal circumstances they can safely be extended to several hundred feet. *Soft rules* are guidelines that describe good practice, and their violation may result in decreased performance and increased error rates in specific situations.

Networks are rarely designed as a whole, but almost always evolve from a minimum configuration by the addition of needed connections to other networks and new hardware. This implies that the design phase is never over and decisions that were made in the past continue to interact with current decisions. The administration and design of local area networks often passes from one network administrator to another. The new network administrator must often modify the design (e.g., add new machines or connect new rooms) of the previous administrators. In most cases, this occurs without the new network manager being aware of the rationale of a previous design decision or what problems had been addressed in the past.

The design of networks often relies on a number of people who form the network support team and who each have to make decisions about the design. Localized decisions (e.g., where and how to connect a new

workstation to an existing network) should be made in the context of the overall design goals. Most networks are large enough that no single person can know all the details about the network design.

### 7.1.1 Existing tools used by network designers

The technical challenges of designing and debugging networks has produced a large market for software that supports network designers. Tools and artifacts already in use by network designers provide ideas and measures for comparison in supporting network design.

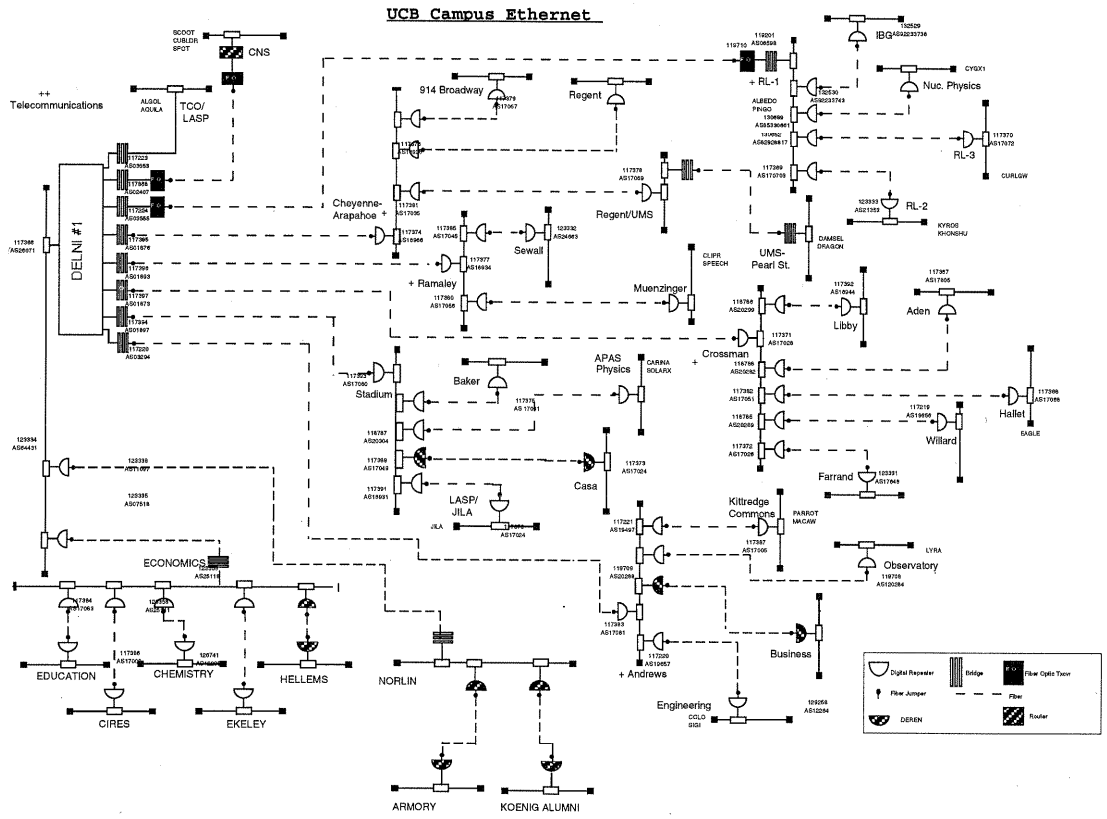



**Figure 17. Part of a Network Diagram Produced in MacDraw.**

A network diagram in MacDraw shows the logical layout of the campus network. This diagram can be easily modified, but because the are not formally represented in the system, it can only be used to record the state of the network or as a topic of discussion.

Not all systems in use by network designers were specifically designed for the domain of network design. Figure 17 shows an example of a logical map that is a portion of a MacDraw document created by a network designer. This artifact was used to focus issues that arose during design meetings. Use of MacDraw makes it easy to add new device types, one just adds a symbol in the lower right-hand side box and then does a cut-and-paste onto the document. This representation has the limitation that changes to this document have no semantic meaning with respect to the software, i.e. nothing beyond the interpretation given to them by the users.

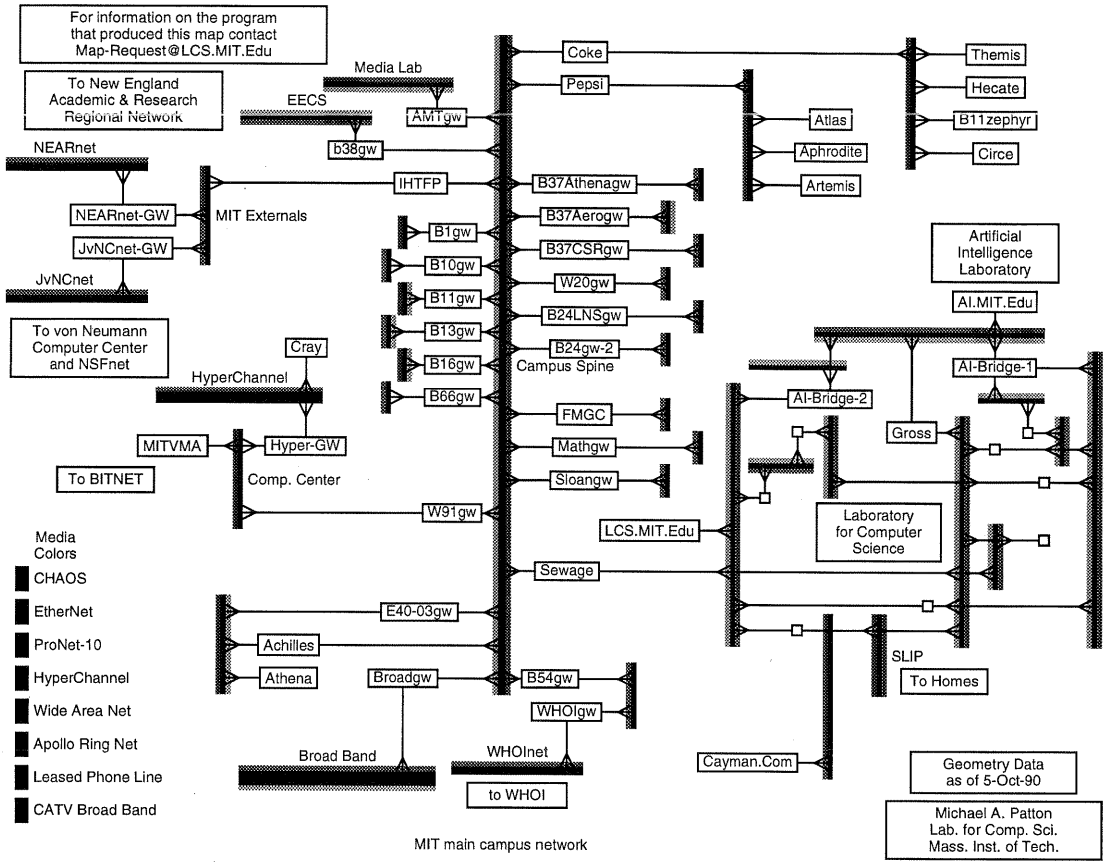



**Figure 18. A Network Diagram from the MIT LCS Interactive Map System.**

This diagram shows a logical network diagram from the MIT LCS Interactive Map system. Because of the system's domain semantics this diagram can also be used to access the state of network devices and parts of the network but cannot be easily modified.

Figure 18 shows another logical map produced by the MIT LCS Interactive Map tool. This tool goes beyond the static document depicted in Figure 17 by adding domain-specific support and active components, e.g. an SNMP (simple network management protocol, [Case et al. 90]) component that allows the user to select a node and query the network to see its current state. The consequence is that the system is not easily changed to accommodate new devices: one cannot just "add a new icon" and start placing it in the network, as in the case with the MacDraw document.

There are also many commercial systems for use in monitoring traffic patterns and diagnosing problems in a network design. These systems often provide a computer-aided design (CAD) style of interface and are specific to certain classes of hardware or network protocols and topologies. These systems either require the network designers to maintain a technically accurate model of the network design or they build such a model automatically.

Design artifacts, such as those shown in Figure 17 and Figure 18, influenced the work on XNetwork. HOS's integration of formal and informal representations enables XNetwork to combine easy use and extension with the formal model of the network that is needed for the system to and support the design

process. These diagrams are similar to those used extensively by network designers in videotaped sessions to explain the history and rationale behind previous decisions and to discuss possible revisions when given "what if" scenarios [Reeves, Shipman 92].

### 7.1.2 Characteristics of network design information

Network designers have developed certain representations to meet their needs. Over time, the logical map has evolved to serve as a key design document. It eliminates many physical characteristics of the network but preserves connectedness. Types of devices are represented by icons that are generally understood by the community of network designers. The use of logical diagrams in network design as the central artifacts during design sessions and in recording designs implied that XNetwork should support the use of logical diagrams.

In videotaped design sessions, network designers consistently used a white-board to draw a logical view of the network which served their task. Diagrams of the same network vary for different tasks. The designers only provide detail to the diagram when that detail is useful for the current task--abstracting away unnecessary detail. While the diagrams do not consistently portray physical space, certain aspects of the physical space are sometimes maintained in the diagrams. For example, when describing network cables going through three distinct sections of a building, the network designers labelled the sections of the building on the diagram and were careful to place machines in the section of the logical diagram that other machines in that section of the building were placed in. Such a diagram is shown in Figure 19.

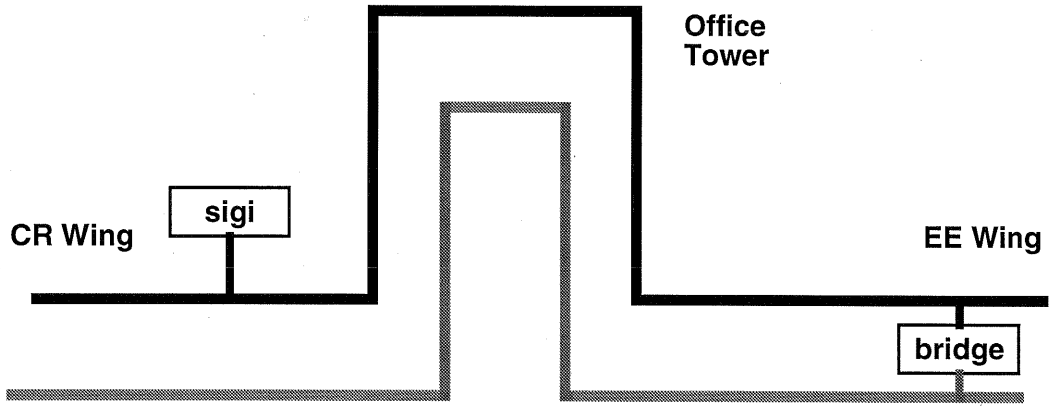



**Figure 19. Diagram Combining Logical Layout with some Physical Information.**

This logical diagram also includes some information about the physical layout of the network. This diagram shows that the bridge connecting the two cables is in the EE wing while the device sigi is located in the CR wing.

One difficulty in providing tools to network designers is that group members need to view the current design from different perspectives. Supporting multiple perspectives on a complex design artifact is not something that can be easily added later as a separate component to an existing system. The difficulty of multiple perspectives in network design can be seen from the two most apparent views: (1) Physical: this includes the physical attributes of all the devices, such as disk space and physical location. (2) Logical: this is a higher level view that abstracts away many of the details in order to provide a global picture.

These perspectives are supported by similar data, namely formal representations of physical placement, devices, and connectivity. Updates to this data that occur in one view must also update other views using this data. But these are not the only perspectives, the physical and logical concerns for a task are not separable from the social context. Information about which group uses a set of machines and the type of work they do affects the requirements for the network design. This information involves formal information about which machines are used in which rooms by the group as well the implications of the network requirements for the type of work this group does. The variety of views, and the open-ended nature of the influences on a design decision, again points to the need for XNetwork representations to be able to evolve.

Much of the technical information concerning networks is likely to already be on-line in some format. Data files required for the network to operate, and files or databases used by network designers to keep track of information useful for administration are often kept up-to-date by automated broadcasts of updated information. For XNetwork to be of use it must not try to drastically alter the current distribution of information about the network. At the same time such a system must integrate information not currently recorded in data files, such as the rationale about the designs and the social constraints on the network.

Besides various technical details necessary to provide a good design tool, the analysis of network design has shown the need to integrate less formal information with the formally represented artifact. Designers need to be able to deliberate decisions and to document those deliberations in the context of the design. In design meetings (videotaped for analysis), those discussions were not divorced from the design artifact, but grounded in it [Reeves, Shipman 92].

### 7.1.3 Interactions between network designers

When possible, collaborating designers ground discussions in representations of the artifact being designed [Reeves 93]. In video-taped sessions, network designers were observed as they explained previous design decisions and solved theoretical and upcoming expansion problems. They used logical maps to: (1) point out inconsistencies between an appealing idea and its difficulty of implementation, (2) remind participants of important constraints, and (3) describe network states before and after changes.

Most of the time the network designers work independently or in small groups on separate projects. As a reflection of this, methods of asynchronous communication, such as electronic mail and USENET News, account for much of the network designers communication. This communication, because of the nature of the medium, occurs without the use of a diagram like the ones used during face-to-face discussions.

The amount of communication and the need to record this communication are exemplified in the approximately 500 electronic mail messages a month that are archived by the University of Colorado's Computer Science Department's network designers. These messages concern the design and administration of the departmental network of approximately 400 devices. The archive of messages provides a record of the design discussions, problems, and solutions concerning the network. To locate information within these e-mail messages the designers use standard Unix tools such as grep, awk, and sed.

Communication about the design also occurs on the actual physical network itself. Labelling cables and keeping records of the end-points of each cable requires constant updates as the network constantly changes. Without such records problems of redesign and implementation of changes become more difficult. An example of the difficulty of not having information was seen after one fairly complete change in networking personnel within the department. The new network design team, being left a network without recent diagrams and labels on cables, spent the better part of a year mapping out and labelling cables. Without the current records that work would have to be repeated if there was another drastic turnover in the network design team.

## 7.2 XNetwork: Building a Design Environment for Collaborative Network Design

The analysis of network design has been used to point out the benefits and limitations of other systems supporting network design and in general to characterize the nature of information and tasks that network designers and administrators face. XNetwork was created using HOS to support the construction of a design, the capture of design rationale, and the communication among network designers. Previous work on domain-oriented design environments has integrated the construction of form with the argumentation about decisions through the combination of a construction component with a hypermedia argumentation component [Fischer et al. 89]. The creation of other components, such as a catalog, specification component, and simulation component, have been discussed in [Fischer et al. 92].

By using HOS as an underlying substrate, XNetwork goes beyond this framework of connecting components that deal with a particular type of knowledge in isolation and allows the integration of partial or whole design artifacts, PHI-style argumentation, and plain text notes within a single view. While separate components may and still do exist to support different activities, the need for designers to use a particular component to access a certain type of knowledge is greatly reduced. All the knowledge in the system is available to all the components.

### 7.2.1 Rethinking the domain-oriented design environment architecture

As in other design environments XNetwork includes a workspace where the design of form will often occur. This construction kit window includes a palette, a work area, a construction overview, and a message area. The palette can be loaded with views containing domain specific building blocks which the designer uses to build graphical representations of the design. Figure 20 shows a construction kit with a palette of network devices and cables on the right that can be selected to be used in the design shown in the workspace on the left.

The construction overview, not part of the previous construction kits used in design environments, provides the designer with a context for the part of the design shown in the workspace. This is needed in network design because of the size and complexity of designs. Previous design environments have dealt with domains where designs contain at most tens of objects, while a departmental network will often include many hundreds of devices.

The construction kit window is one of the additions to HOS required to better match the task of supporting design. The work area of the construction kit window functions like any other page in HOS. Any view may be loaded and used as a palette. When an object is selected from the palette a copy of that object is instantiated in the work area. As in all views text objects can be added to the work area to annotate the design.

When a discussion is no longer needed in the workspace it can be moved to another view, perhaps into a view that is operating as an argumentation page. A second extension to HOS for supporting design is the addition of a PHI argumentation mode. The argumentation mode provides a set of options on the popup menus in the view to allow the easy creation of PHI (Procedural Hierarchy of Issues) structured issue bases [McCall 87]. Pages in XNetwork used for argumentation can include PHI structured argumentation, pieces of designs, and textual discussion of the design. There is no loss of functionality in moving a piece of a design or discussion from the work area to another page except the absence of the palette and overview. Figure 21 shows an argumentation page containing PHI structured argumentation which, in turn, contains a piece of the design seen in Figure 20, along with its discussion.

Pieces of a design may be copied for purposes other than argumentation. In design sessions, network designers often referred to the state of a design at a particular time, such as "Before the CAPP lab move ..."

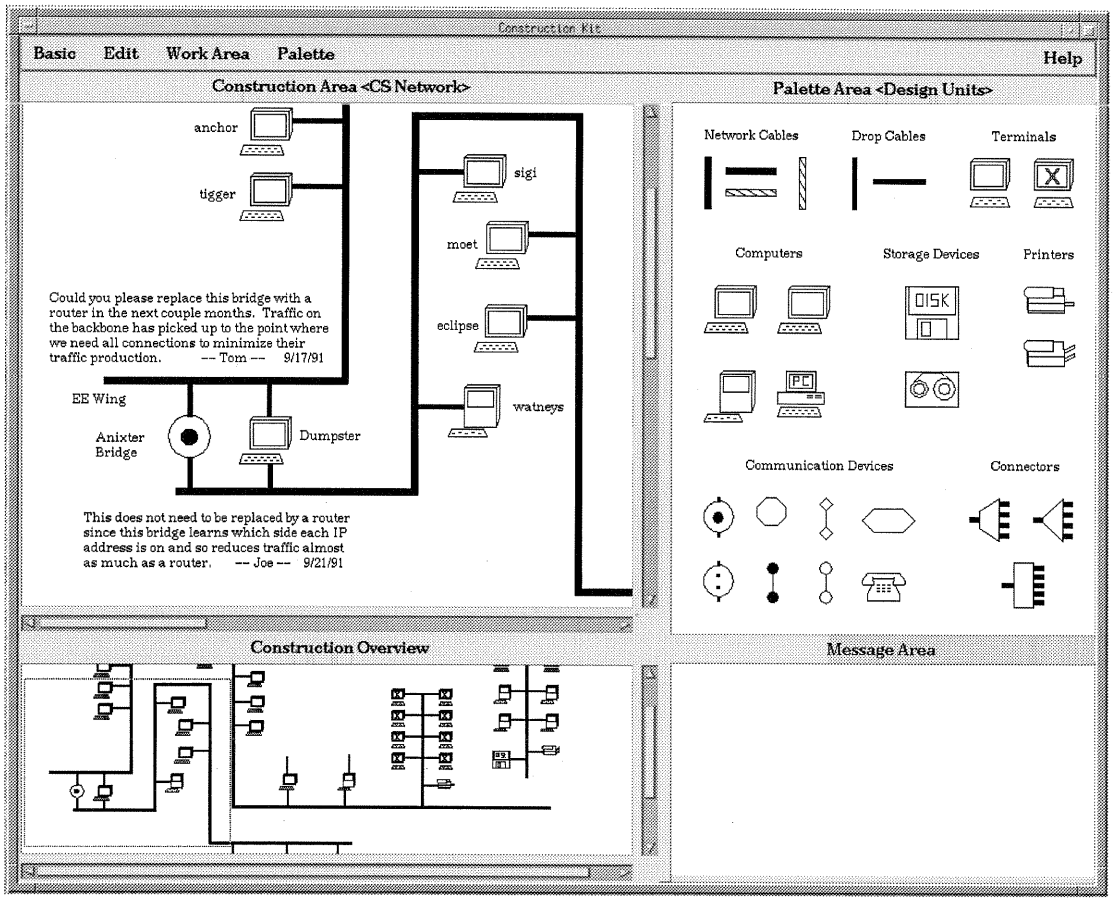



**Figure 20. The Construction Kit Window in XNetwork.**

The construction kit page includes a palette area (on the right), a work area (on the left), a construction overview (bottom left), and a message area (bottom right). Design units can be selected from the palette for use in the work area. The construction overview shows the current design at a small scale to provide the designer with a larger context of where they are working.

Copies can be used to record previous states of a design that can later provide a context for an old discussion or as prototype examples.

In observed discussions between network designers, tasks are performed without recognition of discrete sub-tasks based on the types of information they require. The various types of information are not separated in the designers' minds, they switch from one type of information, such as design rules, to another, say previous experiences with specific devices, without hesitation. To enable the network designers to operate in this mode within the system, XNetwork integrates the interface to all types of information. This means that hard constraints, soft constraints, experience with certain hardware or designs, specifications, expectations for future networking developments, and examples of previous designs can be used in the process of design without considering where in the interface one can find them.

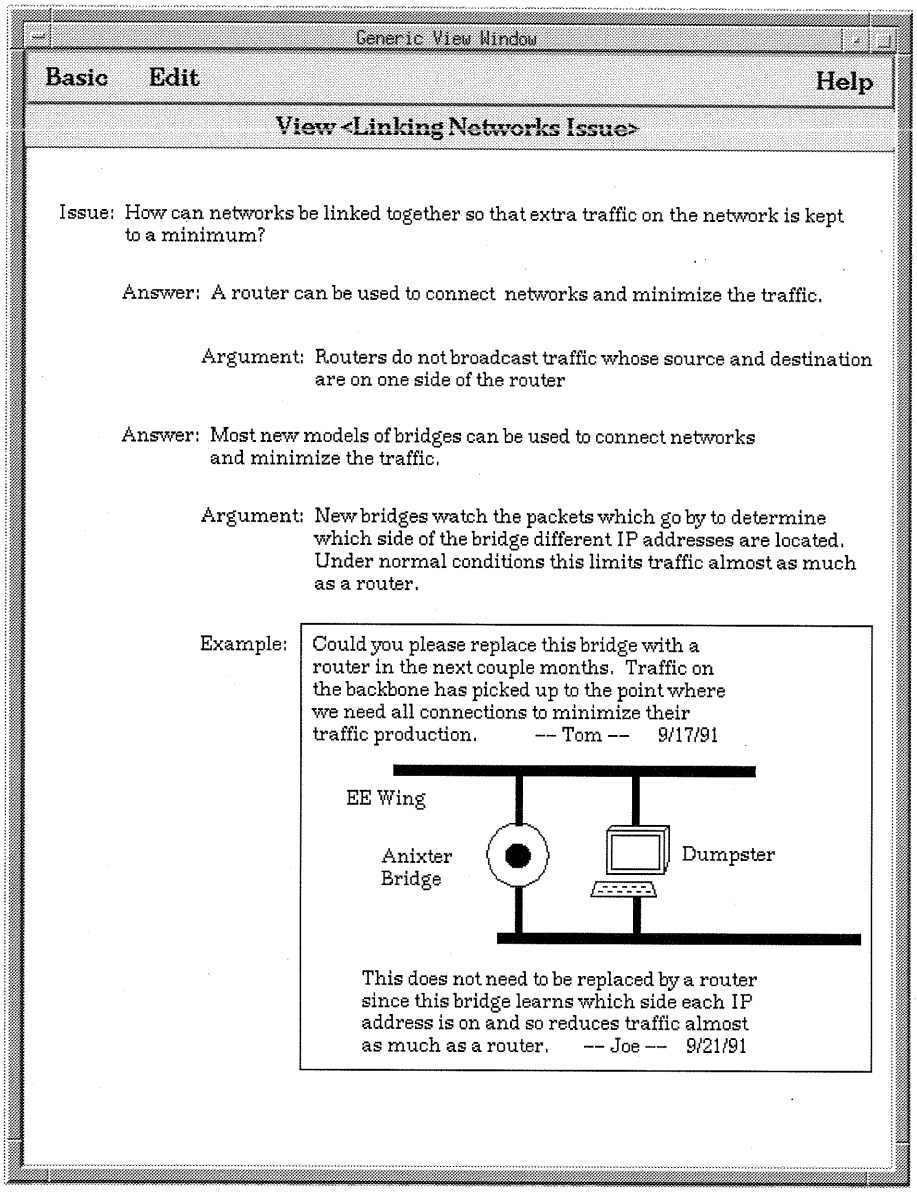



**Figure 21. A Page of Argumentation with an Example.**

A view that is used to record design rationale in an issue-based representation. Here a piece of the design from Figure 20 is shown as an example in the issue network.

HOS makes the integration of these different components possible because of its homogeneous underlying knowledge representation. One cannot simply piece together a construction kit with an annotation and argumentation system and achieve the integration necessary to support the evolution of the design artifact and the discussion about that artifact. The system must be designed from the ground up to support the diverse types of knowledge that must be represented in such a system. From formal knowledge about domain-specific data types, through semi-formal argumentation and to free-text notes: all are part of a single evolving artifact.

A conclusion from the observations of network designers that was described earlier is the need for the system to allow different perspectives of a design. XNetwork supports the use of different views of the same network through the creation of *virtual copies* and equivalence classes of objects in the design. Virtual copies are created to display the same object within different view objects showing different perspectives of the network. An example of use is that an important disk server will likely be displayed both in a map of its local subnet and in a larger network map. Changes made to a design unit's underlying knowledge, such as changing the disk size of the disk server object, will be available through all the virtual copies of that object.

Equivalence classes of objects, built using cycles in inheritance graphs (see Figure 7), may also be used to create different perspectives of a design. An example of the use of equivalence classes is the display of a subnet as a single line in a view of the whole network while in a detailed view it could be a compound object consisting of dozens of devices with interconnections. By setting up an inheritance cycle containing the line in the whole network view and the compound object in the detailed view, attributes added to one of the objects will be inherited in the other. Use of virtual copies for providing perspectives is more computationally efficient than equivalence classes but each virtual copy is limited to having the same display.

### 7.2.2 Supporting different levels of abstraction

Network designers operate at many different levels with regard to the design of the network. Some tasks, such as the initial design of a new network, may be loosely specified and the result may be an abstract design plan. Other tasks, such as adding a new set of workstations to an existing network will often result in a much more concrete design. But abstraction does not only vary with regard to the details of the result. For common tasks a network design team will have standard combinations of devices, such as pairs of transceiver and drop cable choices. These subassemblies enable the designers to work at different conceptual levels depending on the task. Also, members of a network design team tend to specialize into certain aspects of a design so that the subassemblies are likely to differ between designers.

There are two ways in which XNetwork supports different levels of abstraction. First, HOS's compound objects can be used as subassemblies, such as a combination of a workstation with its peripherals and drop cable. The use of such subassemblies enables more efficient construction of large design artifacts. Second, abstract objects can be created which match the level of abstraction required. An example is that a network designer concerned with a campus-wide network might create objects which represent departmental networks and the devices connecting them without ever specifying the actual devices within the departments' networks.

HOS's compound objects also blur the distinction between the catalog of previous designs and the palette. Catalog examples in JANUS consist of a set of design units grouped together in a particular formation which might be connected to argumentation and specification information for the example [Nakakoji, Fischer 90]. In XNetwork such a catalog example can be represented by a compound object or a view which contains a configuration of design objects, textual annotations, formal knowledge about the configuration, and links to related argumentation.

### 7.2.3 Domain specific implementation

The analysis of the network designers showed that they already use software to aid in their design process. Rather than trying to eliminate their use of other software, XNetwork tries to become an integrating addition to their current software. There are two extensions to HOS which were primarily incorporated to better integrate XNetwork into the network designers' existing activities: the shell object and the import

communication. XNetwork incorporates this communication into an information space through import functionality which can read standard (ASCII) text files, electronic mail messages, USENET News articles, and some variation of CAD files. In the case of electronic mail and USENET News articles, the header information is parsed and added as attributes to a text object that contains the body of the message. By providing mechanisms for importing information from the their existing environment, XNetwork respects the existing tradition of network designers.

## 7.3 Generic and Specific Instantiations of XNetwork

There are three distinct information spaces that have been created in the process of looking at network design. One information space is being used to collect information on network design in general. The other two information spaces are being used to look at specific instances of network design. One of these concerns the University of Colorado (CU) Computer Science Department network. The other project specific information space is following the plans for a new computer network to be installed for the computer science department at Dartmouth College.

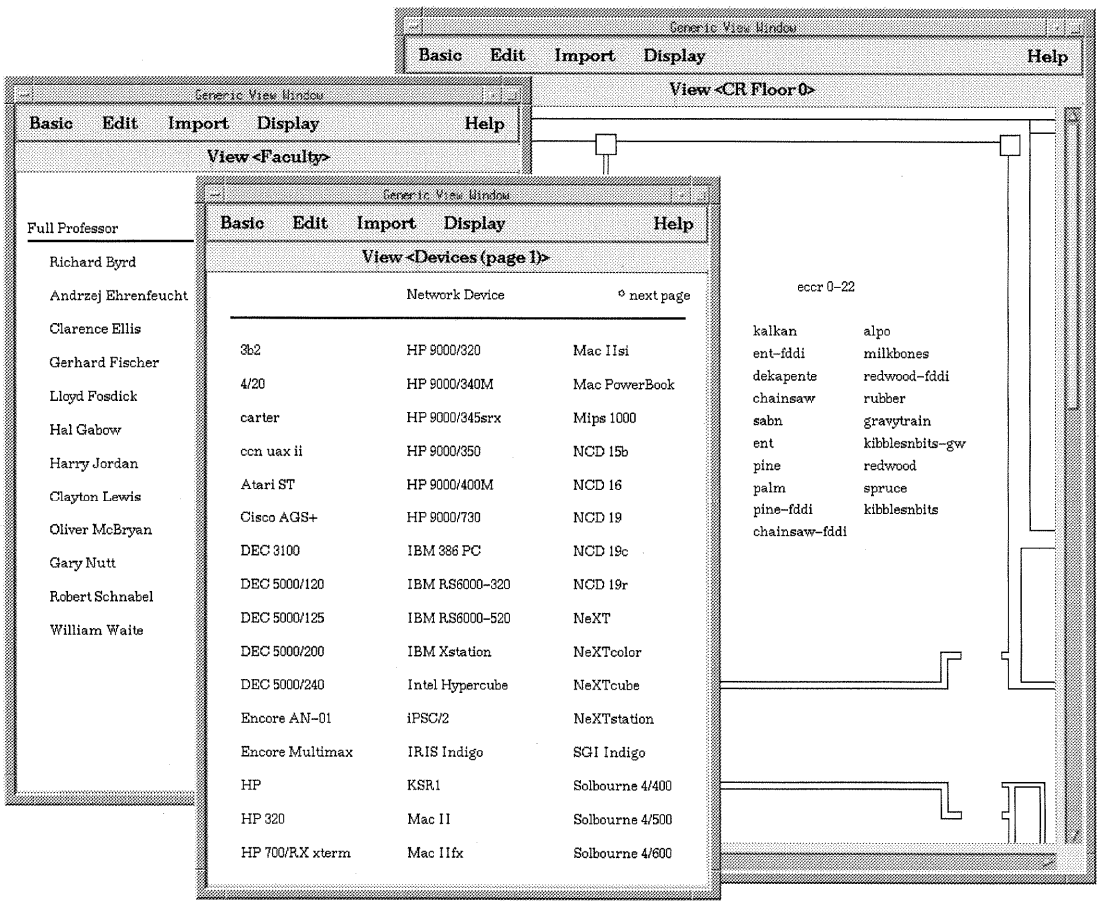



**Figure 23. Views Containing CU Network Information.**

These three views show information from the CU network information space. One view contains the faculty of the department, another some of the device types that are part of the network, and the third shows the physical locations of devices.

functionality. Both are of use in other domains, but they are crucial to integrating XNetwork into the current activities of network designers.

The shell object is an extension to the basic set of display types for objects. A shell object's display is defined by a Unix command. When the shell object is displayed the Unix command associated with the object is evaluated and the results become the textual display of the object. Shell objects provide access to standard Unix commands as well as any shell scripts or custom software that the network designers may have created which return textual output. An example of the use of shell objects is shown in Figure 22. Besides dynamically displaying information from the Unix shell, shell objects can be used to perform activities which change the software configuration of the network from within XNetwork by using the side-effects of executing Unix commands. An example of such a use is a shell object which removes files from the temporary file partition that have not been modified for a certain period of time and displays the disk space available in that partition.

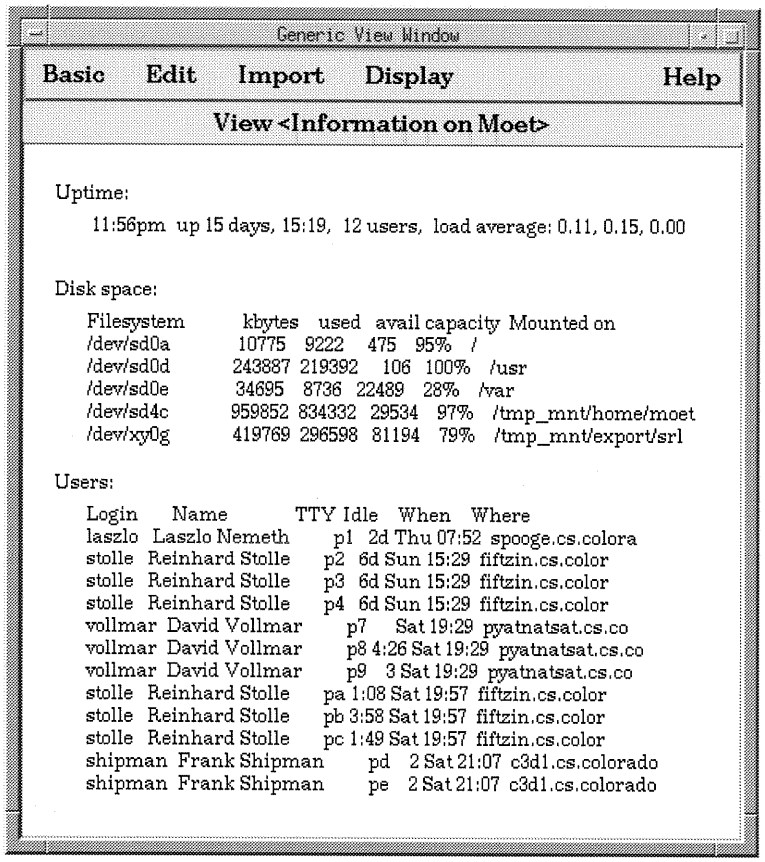



**Figure 22. Shell Objects Providing Information on Computer Utilization.**

This view of information about the workstation moet includes three shell objects providing information on the CPU load, the disk space, and the current users logged into the system.

As was mentioned earlier in this chapter, network designers heavily use electronic means of communication--especially electronic mail and USENET News. This use of these electronic media means that a large amount of information concerning network design is communicated via textual

The generic network information space has been created from information in textbooks, catalogs, and magazines concerning network design and from interaction with domain experts. This information space includes hierarchies of network components, discussion of general networking issues, and prototypical examples of certain network configurations. The network components are organized in inheritance hierarchies from generalized device types, such as workstations, to specific networking products, such as a Sun Microsystems Sparcstation 2 GX. Designers can use any level of specificity they wish when putting together a network--the more specific the more the system can check that constraints have been met. The generic information space provides a starting point for project specific information spaces.

This information space concerning the CU network includes information from the generic information space plus information specific to the CU network. This information includes the approximately 300 hosts on the departmental network, the 60 or so faculty and staff, and the more than a hundred locations within the engineering center that the network connects or is in near. Figure 23 shows views containing faculty, device types, and the physical location of devices.

Part of the process of creating the specific information space was importing formalized information from data files and then further formalizing that data. Much of this formal information was already on-line and could be automatically cross-referenced to create an initial project-specific set of formalized objects. Because of inconsistencies in the format of the source files the resulting set of approximately 1000 objects required some modification before the inheritance hierarchies of people, places, and devices were connected in a consistent manner. Besides building inheritance hierarchies the mechanisms to incorporate the formalized information created semantic relations between the different types of objects. An example is the creation of a "location" relation between the object corresponding to the Symbolics machine bazille and the object for room ECOT 7-12. While all networks contain some data files that could be imported, in this case the site-specific formalisms (agreed upon and maintained by this specific network design team) were also imported.

While the project-specific formal information was semi-automatically incorporated into the information space the informal information presents a larger problem because of the large quantity. There are on the order of 10,000 messages concerning design and maintenance issues on the CU network that have been archived over the last couple years alone. The archived electronic mail is less amenable to the types of batch processing applied to the formal information. As such, only a small set of messages determined interesting were incorporated into the initial project specific information space.

The other specific network-domain information space concerns a situation quite different from the day-to-day maintenance and enhancements that occur in the case of the CU network. The Dartmouth College Computer Science Department is moving to a new location and must install a network into the new location. Monitoring the process of this project has shown that at the design's current state (before the network installation has begun) the discussions are at a higher level of abstraction than most of the tasks followed in the case of the CU network. In the Dartmouth project a floor-plan is provided with the use that each room will serve and discussion concerns how many connections each room requires and how best to provide these connections. Figure 24 shows part of the floor-plan and the specifications for one of the rooms in a property sheet.

Over time the formal information, such as the number of connections to any given room may change several times. As an example the number of connections was increased above current needs when it was decided that planning for the evolution of the network was a primary concern. Because the cost of installing cable is greater than the cost of the cable it is reasonable to provide more connections than are currently required and so reduce the need for rewiring when more connections are needed. In this case informal information from electronic mail discussions led to a modification of the formal information. XNetwork can represent such connections between formal and informal information.

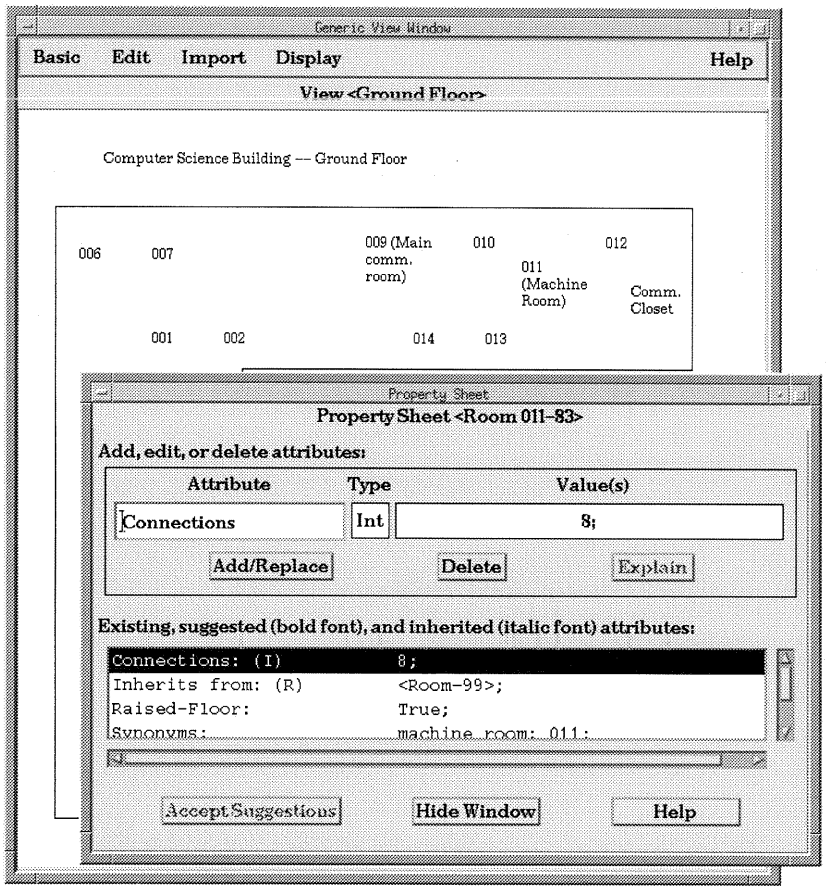



**Figure 24. Floor-plan with Room Specifications.**

The Dartmouth network information space contains information concerning the spatial layout of the building as well as the use and number of connections needed for each room. Here a property sheet shows the specifications for one of the rooms.

These three network-domain information spaces show the ability of HOS to support a variety of different tasks within a domain. The representations of general network information, both formal and informal, can be integrated into a single information space. The incorporation of project specific information into such a generic information space is shown by the CU network information space. The CU information space also shows how HOS can support the work structured around a variety of individual modifications. Finally, the support of the more abstract processes involved in the creation of a new network can be supported as shown in the Dartmouth information space.

## 7.4 Summary

Network design is a rapidly changing field where a design will pass from designer to designer over the life-span of the network. Because networks are continually upgraded and extended, supporting the design of computer networks means supporting the evolution of the artifact and information about the artifact. Current tools used by network designers can only support part of the process of design, either being generic and not providing much support or providing a lot of support about a specific aspect of network design.

From looking at how network designers interact, the logical map was determined to be the primary artifact discussed. Network designers vary the map based upon the current task, tailoring the view to remove unneeded detail. Beyond the use of the logical map in face-to-face meetings, network designers make heavy use of electronic asynchronous means of communication, such as electronic mail and USENET News. Integrating the informal communication with the formal representation of the design was easily supported with HOS.

HOS is a substrate to support many of these information and interaction properties. The substrate was applied to create XNetwork, a domain-oriented design environment for supporting network design, has been created using HOS. Extensions, such as the addition of a construction kit window and PHI argumentation mode, were made to HOS to better support design. HOS's integration of hypermedia and knowledge-based representations enables the integration of information separated in previous domain-oriented design environments, the creation of multiple views of the same artifact, and the use of varying levels of abstraction in design.

One generic and two case-specific network design information spaces have been created. The generic information space contains hierarchies of network devices and discussion of issues concerning network design. The first specific network design information space concerns an existing network connecting more than 300 devices. This information space includes information about the specific devices, the locations, and the main users of the network. The second specific information space concerns the plans for a new network. This information space includes the evolving specification and discussion concerning the specification of the new network. These three information spaces show the flexibility in tasks supported by XNetwork.

The creation of XNetwork provides an example of the process of creating a domain-specific application with HOS. Not surprisingly, since network design was chosen as an application domain early on, HOS was well matched to this domain both in representation and in interface. HOS's use of first-class objects, especially in the case of compound objects, and the use of prototype inheritance supports the representation of different abstractions within network design. The ability to rearrange inheritance hierarchies and to add new attributes and relations was heavily used in the creation of the three information spaces. The hypermedia interface provided an easy way to organize and annotate the formal information. Encoding design rationale as argumentation combines the hypermedia and representation capabilities.

Of course the use of a system by its developer for a planned task is not a true test of the generality of a system's functionality. The following chapter will discuss the use of HOS to create other domain-oriented applications and use by people not involved in the HOS development process.

# *Chapter 8: Observations on the Use of HOS*

The process of incremental formalization in HOS was evaluated by observing and monitoring the development of specific, domain-oriented information spaces. Specifically, an early version of HOS was used by the developer and by an undergraduate computer science student to develop information spaces in the computer network domain. A revised version was used by two students in a knowledge systems class to develop information spaces in the domains of archeology and neurosciences.

The first part of this chapter reviews the goals of HOS. Next comes some observations about hypermedia and design uses of HOS. The remainder of the chapter will discuss the use of HOS in the class projects. This begins with a description of the class projects and then discusses how HOS's functionality was appropriate, inappropriate, or lacking with respect to the projects. This will be followed by a discussion of the evolution of the project information spaces and how the support provided by other systems would have been different for these projects.

## 8.1 Goals for the Use of HOS

The primary goal of HOS is to be a domain-independent substrate which enables and supports incremental formalization. This goal can be broken up into a number of subgoals:

- Information that enters in an informal representation should be able to be formalized "in place."
- Formal representational capabilities should not interfere or discourage the use of informal representational capabilities.
- The formalization process should be actively supported by the system.
- Users' should be able to reflect their changing goals and understanding by modifying existing formalizations in the system.
- The system should be general enough for use in a variety of domains and tasks.

The following sections discuss the uses of HOS with a focus on whether these goals were met. Besides observations linked to the above goals, the discussion will mention aspects of the use situations which relate to the overall need for incremental formalization, whether anticipated within HOS or not.

## 8.2 Reflections on the Use of HOS

During the development of HOS a number of information spaces were created which helped to point out needed improvements in functionality and needed modifications to the interface. This initial usage served to develop the system to the point that large-scale projects could be undertaken with HOS by people other than the original developer. The first discussion of this use concerns the initial development of XNetwork with HOS. Following this is a discussion of a variety of other uses of HOS.

### 8.2.1 Reflections on use during the creation of XNetwork

The first use of HOS was to create XNetwork, the computer network design environment discussed in the last chapter. The creation of XNetwork was primarily performed by the developer, but also included some use by an undergraduate computer science student working with the departmental network design team. This undergraduate used HOS to collect information about general network design and about the specific network within the department.

Creating XNetwork helped to identify more general usages of HOS's design. For example, the flexibility of the prototype inheritance mechanism could be used for more than the traditional "IS-A" hierarchies. Other uses for the inheritance mechanism are for creating equivalence classes of objects (discussed in Chapter 5) and the ability to use inheritance to implement aspects of SmallTalk's model-view controller approach [Krasner, Pope 88] in the display of objects. This latter ability required one object to act as a "model" and others as "view" objects to inherit the formal information from the model while having different display methods.

Creating XNetwork also showed that the suggestion mechanisms could be useful beyond the formalization of the specific information: they could also act as shortcuts in adding information. For example, while building the inheritance hierarchy of device types, HOS 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.

The suggestion mechanisms also can aid the user in finding mistakes or in learning about relevant information within the information space. In particular, once the user begins to develop expectations of when and what types of suggestions will appear, then 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 also 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 or by program tools. These suggestions acted like notifications of this other information's existence.

### 8.2.2 Use of HOS in other domains

Besides the creation of XNetwork, HOS was used for a number of both traditional hypermedia tasks as well as for rapidly creating other domain-oriented design environments. These uses exemplified the variety of tasks and domains for which HOS can be used.

The traditional hypermedia uses of HOS include a trip report discussing the 1992 CSCW conference, a hyperdocument about modern strategic and international affairs based on books by the historian Paul Kennedy [Kennedy 88][Kennedy 93], and the development of the initial outline for this dissertation. These hyperdocuments use pages of text and graphics objects with hyperlinks connecting the pages. Figure 25 shows two sample pages from the CSCW trip report and four other pages listed in the bookmark window.

The experience of these idea organization and report writing tasks indicates success in limiting the interference of HOS's functionality. HOS's hypermedia abilities, while not being as polished as those in commercial systems, were not hindered by the system's functionality concerned with more formal representations. HOS could be used as a hypermedia system without any knowledge of the knowledge-based system functionality.

HOS was also used to build small design environments in the domains of kitchen and bookshelf design. These environments were used primarily to show the functionality of HOS and that it could be used to quickly prototype design environments in many domains. The creation of the bookshelf design environment was done during an hour-long demonstration without being rehearsed. Figure 26 shows the annotated design of a kitchen with a HOS agent notifying the user of a problem in the design. These rapidly prototyped design environments help show the domain-independent nature of the formal mechanisms in HOS.

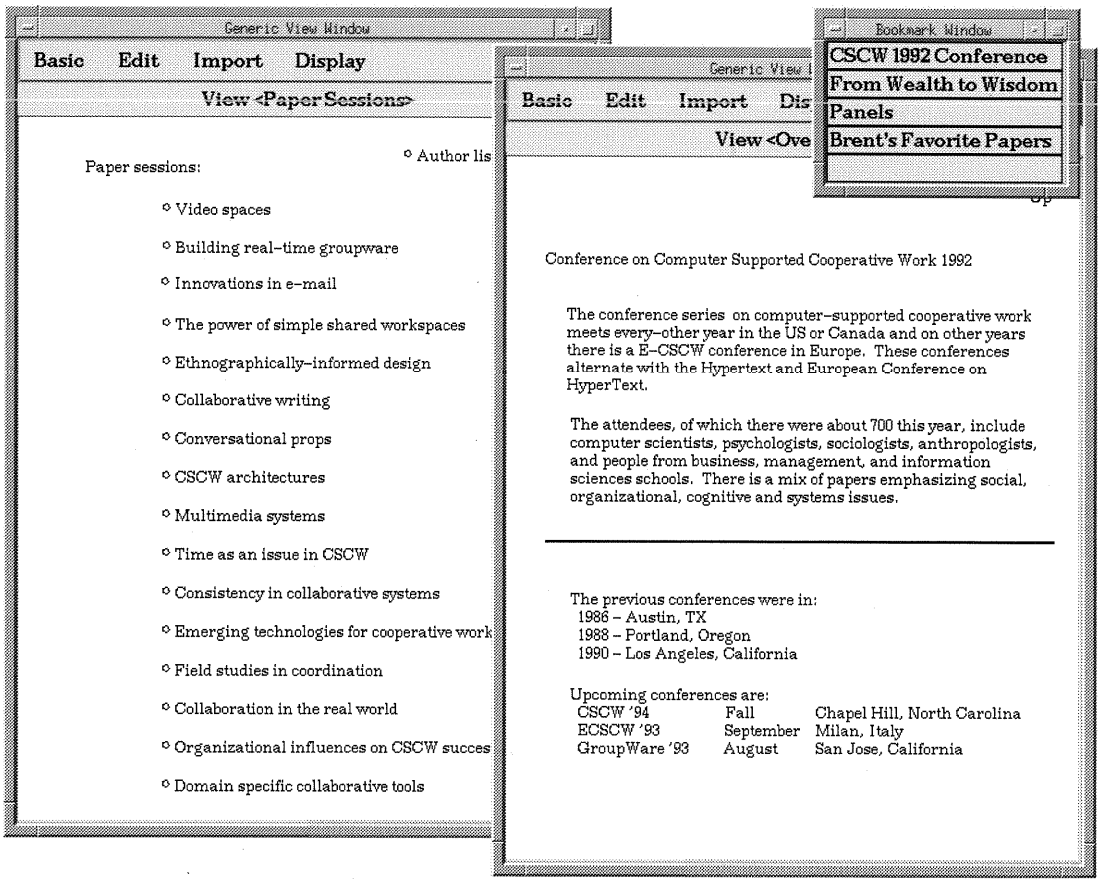



**Figure 25. Use of HOS for Trip Report on Conference**

This figure shows two pages and bookmarks to other pages of a trip report for the CSCW '92 conference that was written in HOS. HOS can be used as a page-oriented hypermedia system without any knowledge of the knowledge-based system capabilities.

## 8.3 Use of HOS in Knowledge Systems Class Projects

Besides the use of HOS for creating hyperdocuments and design environments, the system was used for two projects in a graduate-level knowledge systems class. The goal of the class projects was to create a knowledge-based system for a domain and task of the student's choice. The one-person projects were worked on for a period of about two months. Individual weekly meetings were held to discuss progress and any problems that might arise. Electronic mail was also used to communicate problems, suggestions, and solutions concerning HOS.

During the development of the class projects, information concerning the use of different functionality within HOS was collected using two methods. First, HOS creates backup versions of the database each time a user connects to a database to ensure that information will not be lost due to a problem with HOS. This provides a set of "snapshots" of the information in the system over the period of the semester. For each of the class projects there were approximately fifty of these backups. Second, HOS keeps a log of

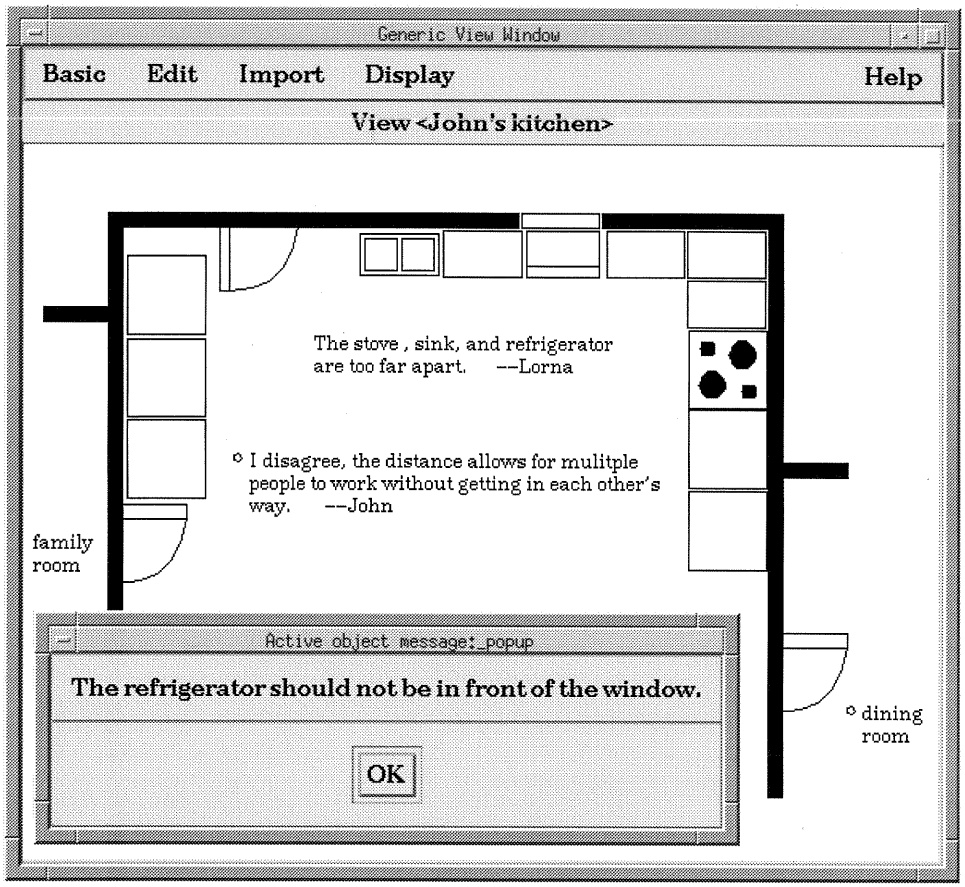



**Figure 26. Kitchen Design Environment Created with HOS**

HOS's knowledge-based system functionality is domain-independent. Here is a view of a kitchen design with a dialog box with a message presented by a HOS agent watching for the occurrence of a certain design situation.

object creations and user-performed attribute modifications. This data provides the "age" of objects when attributes are created or modified.

Because these users were motivated by their class to create domain-oriented knowledge-based systems the fact that they used HOS cannot, by itself, be testimony to the usability and appropriateness of the system. The students' use of HOS can provide interesting scenarios that can be compared to intended modes of HOS usage and to the functionality of other systems. The data from the class projects and the students' project reports can also be used to interpret what functionality was of most use, what was of little use, and what was missing. Finally, the evolutionary nature of problem solving can be examined, with an eye towards appropriate system support or lack thereof, in the development of these two projects. Before discussing the use of HOS is a brief description of each project.

### 8.3.1 Archeological Site Analysis Environment (ASAE)

One of the class projects using HOS resulted in the Archeological Site Analysis Environment (ASAE). Archeological "digs" involve teams with experts in different topics sharing and analyzing large amounts of

information. The purpose of ASAE, as described in the 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."

To try to meet these goals, ASAE combines formal and informal information about archeology in general as well as information about the specific site. General archeological knowledge is represented in a hypermedia document discussing the life-style and signs of habitation left by different cultures, common types of artifacts found in archeological digs, and analytical methods for interpreting a site, such as how trace element analysis can determine the origin of obsidian. Information specific to the site includes the positions, composition and state of the artifacts found, along with the reports of the various archeologists describing test results or discussing possible interpretations of the site.

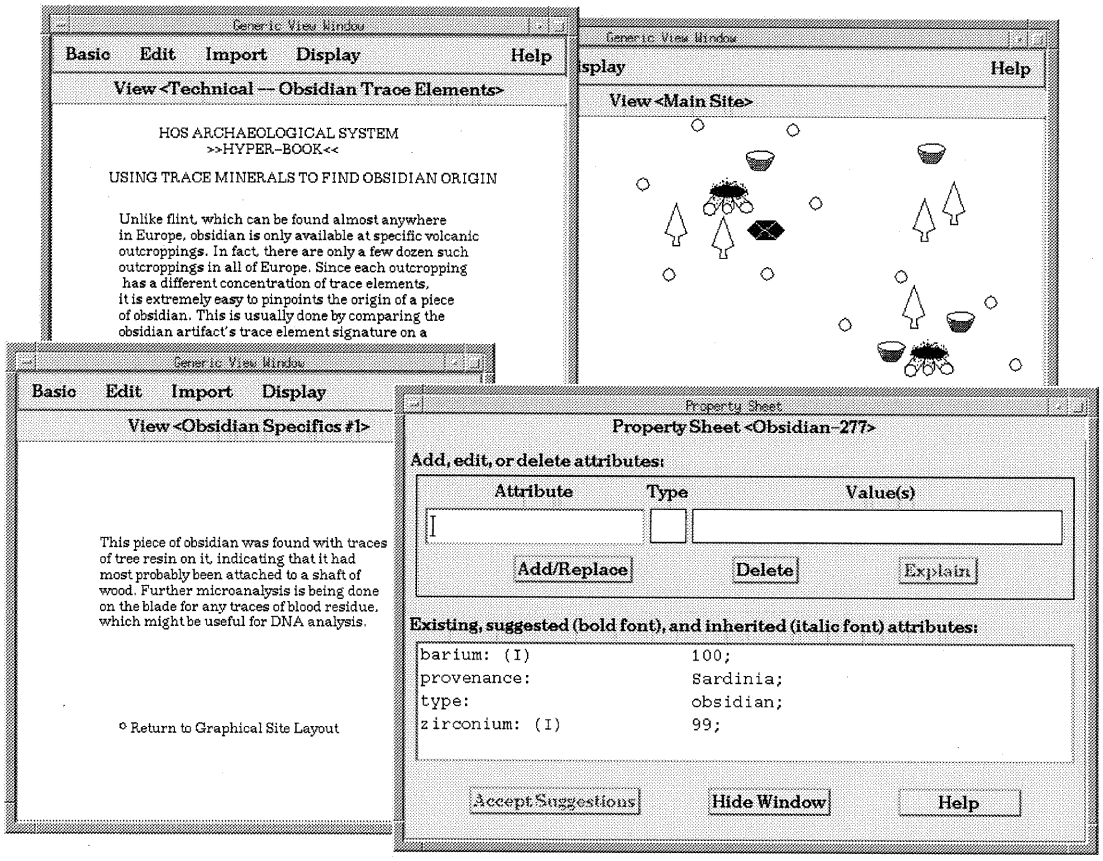



**Figure 27. The Archeological Site Analysis Environment (ASAE)**

Here the archeological site is shown behind one page discussing the use of trace elements to determine the source of obsidian and another page describing tests being performed on this specific piece of obsidian found. Also shown is the property sheet for the piece of obsidian.

ASAE uses one HOS view object to represent the site being excavated. Objects from the site are represented by graphical objects placed within the view object relative to where they are found at the site. The common attributes of the objects, such as their composition, size and shape, are represented as attributes attached to the graphical objects. HOS's hyperlink mechanism is used to link the graphical

objects to textually represented information, such as discussions concerning their features. Figure 27 shows the view representing the site of a dig, a page of the general archeological hyperdocument discussing trace-element analysis of obsidian, a page discussing the specifics of a piece of obsidian found at the site, and the obsidian object's property sheet.

The formally represented information, such as the size, shape, composition, and location of objects found at the site, are used by agents designed to support the interpretation process. Two types of agents are used within ASAE: the standard HOS agents volunteer information based on recent actions of the user, and analysis agents, programmed specifically for the class project. Analysis agents use the formal information to perform tedious tasks and provide initial analyses and suggestions about the inhabitants of the site. An example is that the "provenance" attribute shown in the property sheet in Figure 27 was attached to the obsidian based on the trace-element analysis done by one of the analysis agents.

### 8.3.2 Interactive Neuroscience Notebook (INN)

The other knowledge systems class project using HOS developed the Interactive Neuroscience Notebook (INN), shown in Figure 28. The purpose of INN is to provide an environment for student neuroscientists to collect and organize their knowledge about neuroscience within the context of a pre-authored set of information concerning neurosciences. The resulting role of INN can be seen as a combination of textbook and personal notebook.

The use of pre-authored information on neurosciences provides the student with an initial framework for organizing his/her notes. While this is a task that could largely be supported by standard hypermedia, the goal of INN is to use formally represented information to aid the students by volunteering information that may be of value to the student. By providing a seeded repository for information, including a number of information volunteering agents, INN takes a more active role in the student's learning than a textbook, but does not control the interaction, as is the case with tutoring systems.

The notebook seed 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. Examples of possible uses of this formally represented information querying for cells that have certain properties and looking through an inheritance hierarchy of cell types to understand the relationships and characteristics of different classes of cells.

The formally represented information also provides an existing conceptual framework as well as examples for the student when adding new information. When adding new information, HOS agents monitor the modifications by the student and volunteer information contained in the seed that may be relevant to the current task. An example is that when a student enters a new cell type that has a concentric receptive field, an agent will suggest that X-Cells might be interesting to the student because they also have concentric receptive fields.

### 8.3.3 HOS goals, functionality, and usability

In looking at the use of HOS in class projects a number of questions can be asked that relate to the goals listed at the beginning of this chapter. Was HOS used as was expected for such tasks? Did the students miss taking advantage of features that would have aided the creation of their projects? Also, what functionality missing in HOS would have benefited the students had it been available?

The first of these questions, was HOS used as expected, in general can be answered in the affirmative. When concerned with use of the basic functionality of HOS this probably does not mean much since the

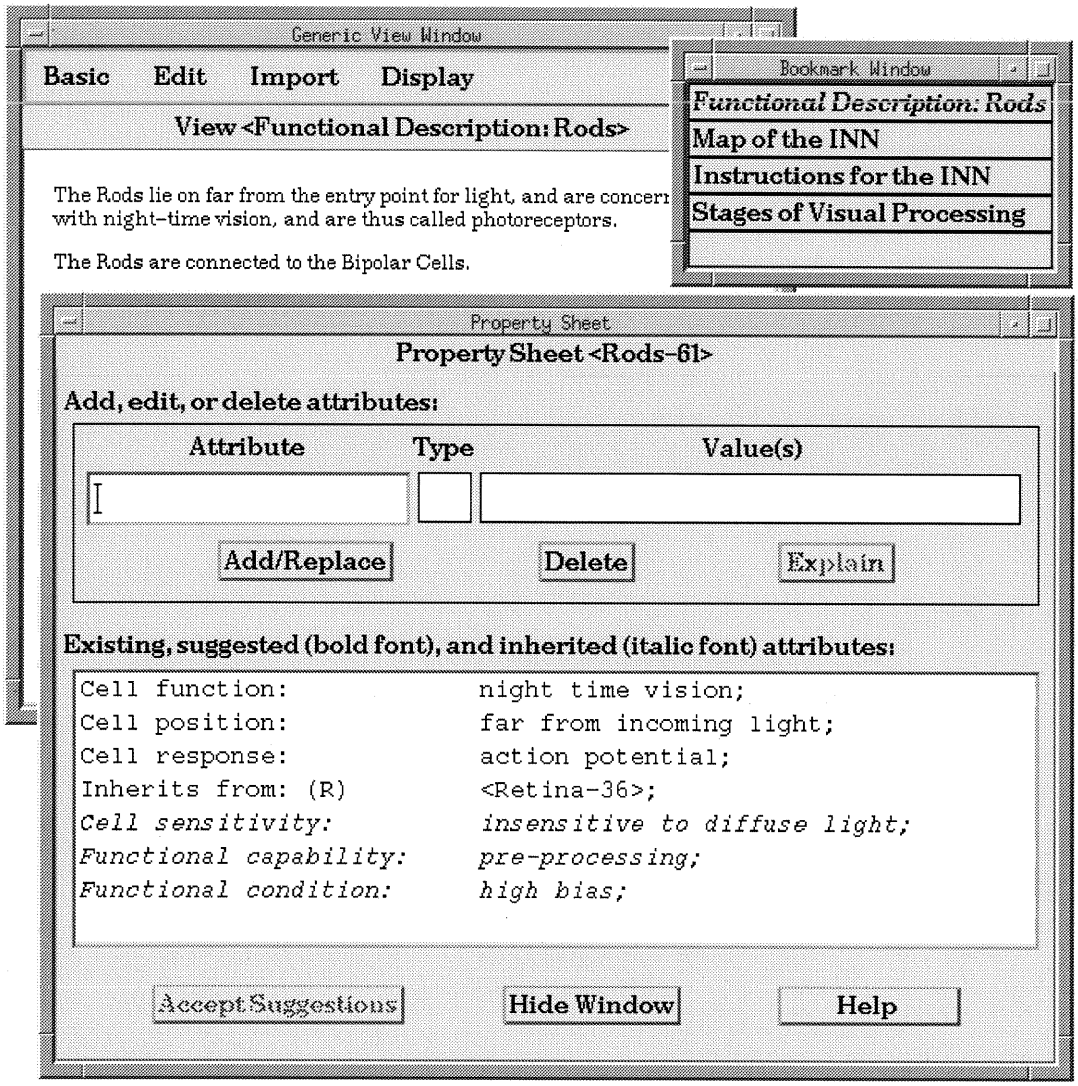



**Figure 28. The Interactive Neurosciences Notebook (INN)**

Here a page of the seeded neurosciences notebook (upper left) suggested by an agent in the bookmark window (upper right) is being displayed along with a property sheet for one of the domain concepts discussed on the page.

students were motivated to explore HOS by their class. Also, because weekly meetings were held with the students, they could regularly ask "How can I do ...?" questions. So, the fact that they used most of the functionality as expected, is to be expected.

The interesting cases where they did not use HOS as expected were primarily in the area of hypermedia authoring. The division of the domain information into chunks was occasionally not as expected, and the organization of hyperlinks was also unexpected. For example, the first paragraph of the view "Functional Description: Rods" shown in Figure 29, is its own HOS object while the rest of the paragraphs in the view

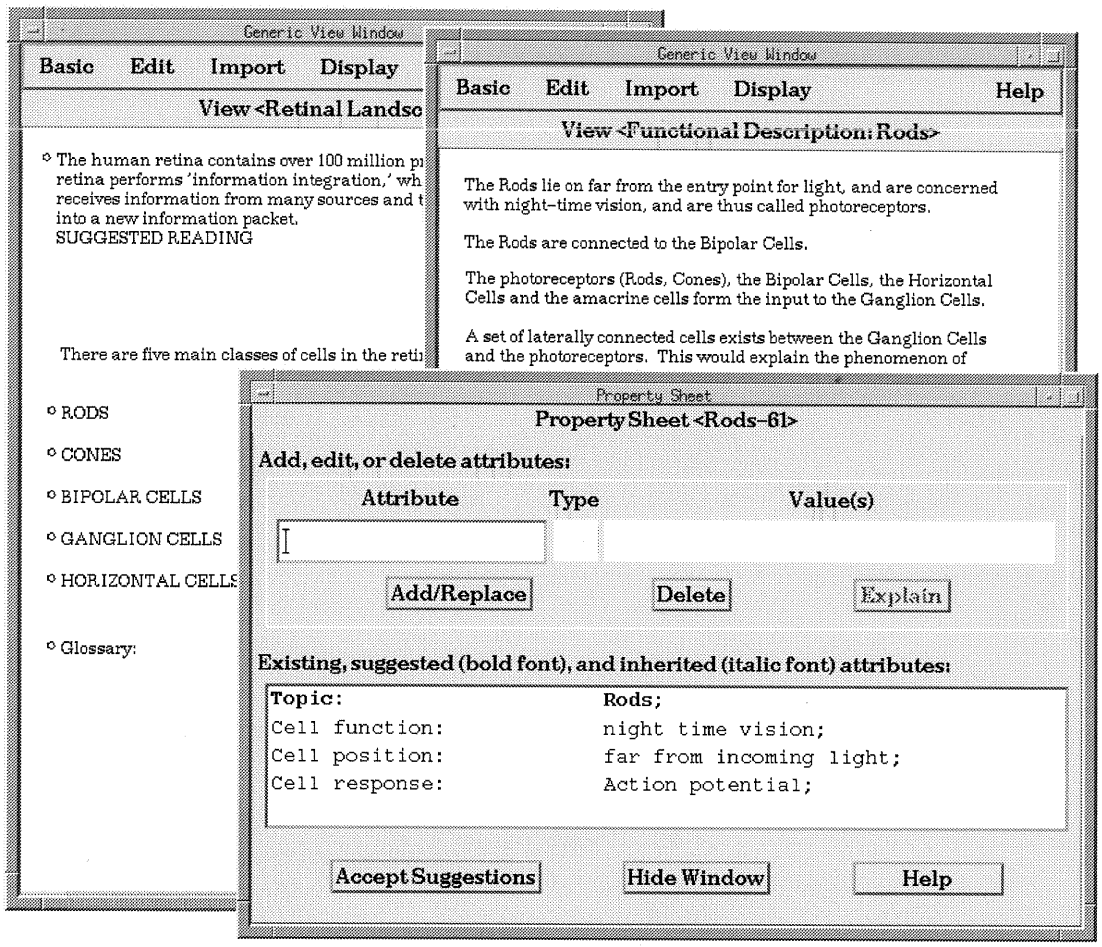



**Figure 29. Status of Concept Object in Unfinished Version of INN.**

This screendump shows the property sheet for the same object as shown in Fig 28 but earlier in the creation of INN. At this point the object had fewer attributes and did not take part in an inheritance relation. The property sheet also shows HOS suggesting the attribute "Topic" to the student.

are all one object. This experience confirms earlier experiences with authoring hyperdocuments that people often disagree about the chunking, linking, and labelling of information.

Another unexpected occurrence, even though it is the expected result of incremental formalization, was the use of paragraphs of text as concept objects. In particular, paragraphs of text concerning a domain concept were sometimes used to represent that concept, meaning they would be named and have attributes attached appropriate to the concept rather than attributes that described the discussion that made up the display of the object. For example, the object representing the concept of "Rods" in the INN (shown in Figure 28 and Figure 29) is the top paragraph on the page and 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." Figure 29 shows the attributes the object had before the project was completed. Figure 28 shows that by the completion of the project the object was place in an inheritance relation 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--matching the steps shown in the diagram of incremental formalization in Figure 1.

In other uses of HOS, objects have generally been used either to represent concepts or to act as discussion, but rarely both. This use as both leads to concept objects being difficult to distinguish from other objects, as in the case of the "Rods" object from Figure 29. To locate the objects representing domain concepts one would have to check every textual object or use queries to locate objects with certain attributes. This experience shows the ability in HOS to formalize information "in place" was used but also indicates some of the difficulties that can result when this occurs.

The students used most of the functionality appropriate to their tasks. The one example of functionality that might have been considered appropriate for greater use was the relations mechanism. In both of the projects few relations were defined other than for inheritance relations and hyperlinks. This result reemphasizes the costs associated in formalization. HOS's current knowledge utilization mechanisms (agents, hyperlinks, and inheritance) do not add any benefits to relations other than inheritance and hyperlinks that is not available to the other types of attributes.

The question of what additional functionality would have been the most use to the students can be answered, at least in part, by their project reports. The most obvious lack of functionality has to do with utilizing the formally represented knowledge. Both project reports noted the lack of expressiveness in defining HOS agents. Another difficulty was that the objects could only be modified individually, some type of set-based data manipulation language (like SQL, but for an object-oriented database) would have made some of the modifications easier. Finally, there were requests for functionality found in commercial software, such as having the copy, cut and paste mechanism work at the attribute as well as the object level.

### 8.3.4 Supporting the evolution of knowledge

Another set of questions that can be asked about the use of HOS concerns the evolution of the projects. Did the problems of cognitive overhead, tacit knowledge, premature structure and situational structure, described in Chapter 2, appear in these projects? If so, how was HOS successful and not successful in addressing these problems? To answer these questions the information collected over the course of the projects' evolution must be examined.

The close to fifty snapshots of each project during the semester and the log of attribute modifications can be used 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 in the content of their information spaces, the rest of the time experiencing slow growth. Several of these jumps were accompanied by the modification of attributes for objects already in the system.

The occurrence of these rapid growth periods leads to the question of was there a cause or was it just the students working harder on their projects. In looking at what information was added at these jumps and using the weekly discussions to consider motivation, these large changes came when the students redefined an aspect of their task or started a new phase in their projects. The class projects were created without any detailed specification in how the final system would work. The goals were set in the initial report, but continued to be refined throughout the projects. An example of one of the larger jumps was the addition of agents to volunteer information in INN. This jump included the addition of some agent objects, but also included a large increase in the number of attributes within the information space as formal information was added that could be used by the agent objects.

A success for supporting the evolution of the information spaces in HOS was that a single database was used throughout each project. Had the system not been able to adapt to the changing goals the students

could have easily created a new database for the new organization. This use of a single evolving information space in each project and the continual addition and modification of attributes up to the end of the projects shows that the addition and modification was allowed to occur "in place". In allowing for the changes in formalism within the system, HOS was successful.

Another question is whether HOS succeeded in actively supporting the changes in formalization. Because of the suggestion mechanisms' requirement of named objects (the lexicon they use is created from these names) it was assumed that they would not be of use during the initial development of new applications. This being the case, their existence was not mentioned to the students. About midway through the projects, the suggestion mechanisms did become noticed by one of the students. In one of the weekly meetings, a student asked about 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 mechanisms work. 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 29. 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 mechanisms work, that is they will frequently make suggestions, with a small set of named objects to act as a lexicon. 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 29, of questionable value. In the end, only a couple of suggestions were accepted during the projects.

One way in which the evolution of the knowledge bases could have been better supported was shown by a period of redesign in the creation of ASAE. During this period the student changed the method in which hyperlinks were used to connect objects at the site to discussion about them. This led to the removal of about forty hyperlinks which were no longer appropriate. To remove these links the student had to remove the links individually, this effort would have been greatly reduced if it was possible to perform operations on sets of objects.

### 8.3.5 Comparison to potential for development in other substrates

One of the comparisons that can be made between HOS and other systems is determining how domain-oriented applications similar to those built in the class projects could be built with the other systems. While each system has its own set of goals and features to support these goals, a comparison of how a number of systems might have supported these class projects may be of use. Systems that will be compared to HOS are the Virtual Notebook System (VNS) [Gorry et al. 91], OVAL [Malone et al. 92], Aquanet [Marshall et al. 91], Janus-Modifier [Girgensohn 92], and a "typical" relational database system.

In short, while the VNS could be considered the closest system in interaction style, it could not support the formal aspects (agents and inheritance) without significant amounts of programming. Users of the VNS could create textual objects and add attributes with values to these objects. These attributes would be limited to textual (untyped) values and would not allow the explicit expression of relations between objects, other than hyperlinks. Thus, while information could be incrementally formalized up to the point of objects with attributes, the user of the VNS would not be able to use typed relations or inheritance without programming. Also, the VNS would not actively support the creation of attributes or be able to provide knowledge-based support with this information.

At the opposite side of the spectrum of formality among this set of systems is the relational database. The comparison of the relational database to the other systems is made more difficult because databases are systems designed for very different users. In fact, both the VNS and Aquanet are implemented on top of a relational database and HOS includes a contains an object-oriented database for data storage. Without the

effort that was put forth to create the database schemas which underlie these other systems, the user of the relational database is stuck with a lot of up-front work with a language like SQL to begin to create the type of knowledge bases developed in the class projects. Beyond this initial start-up cost, even adding a new piece of textual information to the information space the relational database system would require the user to add a row to a table using SQL. Particularly problematic is the creation of objects with unique sets of attributes. In the class projects there were few cases of a set of objects having the same set of attributes. Of the objects that had attributes, most had their own unique set of attributes which changed over the course of the project. One approach to allowing each piece of information to maintain a constantly evolving set of attributes and relations requires a separate table for each object and the modification of the table schema each time the attribute list is modified. Alternative approaches of just storing pointers within the object table to rows of an attribute table require significant expertise with SQL and schema construction. (This took on the order of a programmer year in the case of developing the schema for the VNS.) Thus, while relational databases "allow" incremental formalization, they require their users to be fluent in languages like SQL for any modification or use of the information.

The other three systems, OVAL, Aquanet, and Janus-Modifier, are all similar with respect to their capabilities. All use a basic class-instance object representation which can be modified through dialog boxes and property sheets. Deficiencies towards meeting the goals of the class projects include Aquanet's lack of any agent or critic mechanism, and Janus-Modifier's lack of the ability to attach attributes to text objects without defining them as elements in the design unit inheritance hierarchy. Similar to the case of the relational database, all of these systems require more effort than the VNS or HOS to work with informal information because of their emphasis on supporting formal representations. In each case the definition or choice of object class must precede the creation of an object. Also, this decision on a particular object class influences what attributes and relations may later be defined for the object without having to modify the object class before changing the object instance. To allow the uniqueness of object attributes found in the class projects, a class-instance representation would require the definition of an object class for each object instance. Again, incremental formalization can occur as long as the user is willing to accept the overhead of defining and modifying classes when their real goal is modifying the instance. The experience of Aquanet usage, described in the next chapter, shows that expecting this effort from users can result in either not getting the information or it not being entered through the mechanisms provided.

When considering the use of less formal information, other than the VNS, none of the systems would easily enable the authoring of hypermedia pages which could be used for sharing information among archeologists or act as pages in a student's electronic notebook. In short, the combination of free-form hypermedia authoring with knowledge-based support mechanisms is required for these tasks and is not provided by these other systems.

To be fair, none of these other systems were built with the goals of incremental formalization or knowledge-base evolution in mind, and they each include functionality not found in HOS. The comparison is still valid because, other than the relational database, these systems are intended for use by non-computer scientists and, other than the VNS, require domain information to be formalized by their users. In the case of Aquanet the problems that users had with working with the formalisms was recognized and work is being done to address this problem. The next chapter describes work on adding actively supported incremental formalization to Aquanet.

## 8.4 Summary

Observations have shown that HOS is applicable to a wide variety of domains and tasks. Use of the system in both hypermedia and design tasks indicates success in integrating formal representational capabilities into a hypermedia interface without adding overhead to working with informal information.

Further evidence of the variety of domains and tasks supported was the use of HOS in class projects to create the Archeological Site Analysis Environment and the Interactive Neuroscience Notebook. The patterns of information space development during these class projects show that HOS's representation does enable the "in place" evolution of information as project goals evolved.

Patterns of usage reinforced the assumption that formal representations must be allowed to evolve without restriction. Although they were single person two month projects, both projects evolved significantly over the entire period they were being created. Most of the major revisions to the project information spaces were triggered by conceptual changes about the domain and task being supported and the addition of new functionality.

The projects also reinforced the inherent nature of problems concerning cognitive overhead, tacit knowledge, premature structure, and situational structure in many formalization tasks. The evolution of project goals directly led to problems of situational structure. The effect of a relatively large cognitive overhead for little benefit received could be seen in the little use of relations other than for hyperlink and inheritance purposes. The non-restrictive nature of the HOS representation was used to add formality on demand, thus limiting the danger of premature structure.

While some other systems integrate formal and informal information, they rarely support the "in place" evolution of information from informal to formal that is found in HOS. Most of the systems would require the user to interact with programming languages, such as knowledge representation languages or SQL, rather than interacting with the objects through direct manipulation, property sheets, and dialog boxes. The closest systems, such as OVAL, Aquanet, and Janus-Modifier, require the user to interact with class-instance object models. These systems cannot support the incremental formalization of individual objects without each object being defined to be of a unique class and the formality required by their object model and interface makes working with informal information cumbersome.

# *Chapter 9: Application to the Aquanet Tool for Knowledge Structuring*

The preceding four chapters have described the design and use of HOS, a system created to enable and support incremental formalization. This chapter examines how a second system describes the application of incremental formalization by the author to a second system that was already in use. The second system, Aquanet, was designed to support knowledge structuring tasks. The work on Aquanet is interesting for two reasons: the use of domain-independent spatial mechanisms for supporting formalization, and as an example of the addition of incremental formalization to an existing systems where use had shown formalization as a problem.

The changes made to Aquanet can be divided into two parts: changing Aquanet's representation and interface to enable incremental formalization and building suggestion mechanisms to support formalization. This chapter focuses on the second task, the creation of a mechanism which suggests structure based on the analysis of spatial layouts created in Aquanet. This mechanism's lack of using domain information in determining suggestions differentiates it from the textual-analysis suggestion mechanisms in HOS. This domain-independent mechanism has been successful at recognizing implicit structure in both computational and non-computational layouts. Once recognized, this structure may be used to support the user in formalizing the information.

Aquanet was discussed in Chapter 2 with respect to situations in which users chose not to use the more formal aspects of the system. These problems led to the revision of Aquanet to enable and support incremental formalization. Before describing the work on inferring structure, this chapter presents a brief overview of Aquanet and some of the experiences from its use which led to the incorporation of an incremental formalization mechanism. Following this overview is an analysis of a set of spatial layouts from various paper-based and computational media and a discussion of the layout analysis mechanisms built to recognize structure in spatial layouts and how these mechanisms performed. This chapter concludes with a discussion of how these mechanisms can be used to actively support incremental formalization.

## 9.1 Aquanet

Aquanet is a generic hypermedia system developed at Xerox's Palo Alto Research Center [Marshall et al. 91]. Aquanet is designed to support users who are "trying to interpret information and organize their ideas, either individually or in groups." The information collected by users performing "knowledge structuring tasks" is put into information objects which have types defined in a structure "schema." Aquanet supports the creation of structure schemas in a schema editor. Schemas consist of a set of object and relation type definitions. These definitions describe the presentation of the object or relation, the attributes of the object or relation, and the types of values these attributes may have. These schemas are then used as templates providing the structure in which information can be put into the system.

Aquanet is domain independent, providing a frame-based knowledge representation [Minsky 75] similar to HOS and allowing information ranging along the spectrum of formality. Informal representations in Aquanet include the textual contents of nodes, and the spatial layout of these textual objects. Formal representations include the definition of object types and the relations in which objects are connected.

Starting a new information space in Aquanet, called a "discussion," requires the user to pick a schema from a set of pre-defined schemas or to define a new schema in which to work. After a schema is selected the user can begin to enter information into the two and a half dimensional browser that is central to Aquanet's

interface. (See Figure 30.) Users can add new information objects and define the values for the attributes of the objects in templates based on the object's type. Also, users graphically arrange the information objects in the browser.

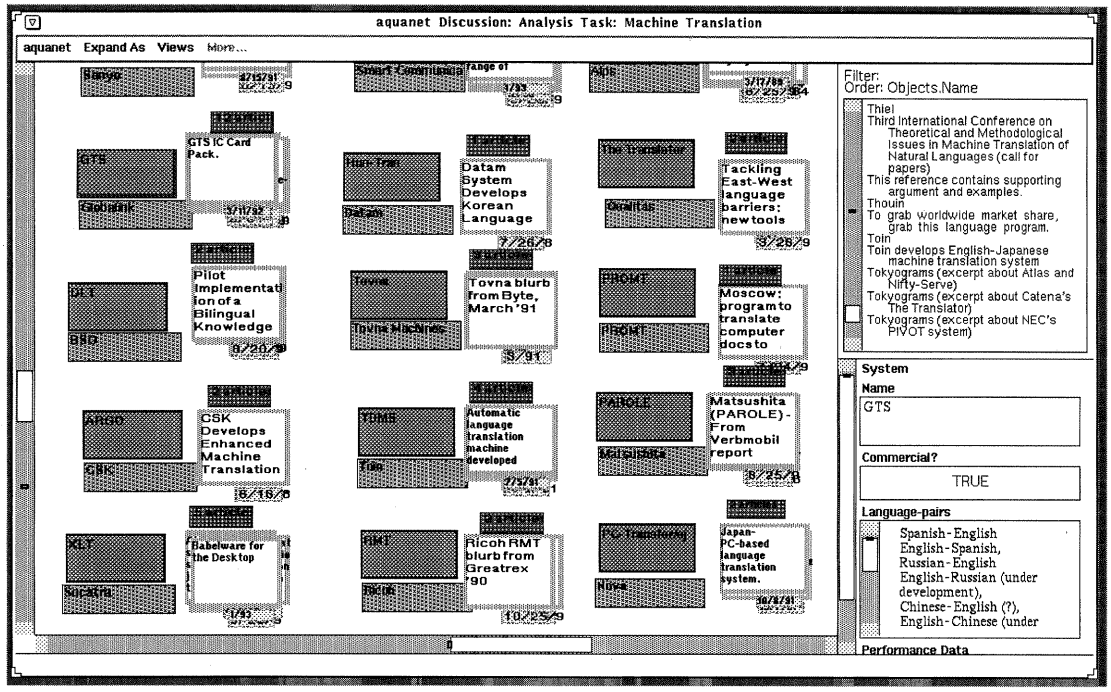



**Figure 30. Screen Image of Aquanet**

The Aquanet interface is organized around a two and a half dimensional browser, shown on the left. This Aquanet discussion concerned with machine translation systems shows four visually distinctive types of objects that have been arranged by the user in patterns. The template in the lower-right corner shows a particular system object's contents. Above the object template is the query and filter mechanism for locating objects that have certain properties.

Each object's presentation in the browser is partially defined by its object type (which defines the color and shape of the object) and partially defined by the object's attributes. In general, only a small part of the information contained in an object is visible in the browser. The browser is more an interface for organizing collections of objects, as shown in Figure 30. By selecting an object in the browser, that object's attributes are shown in an object template

Relations between objects can also be displayed in the browser. The graphical presentation of relations is defined in the schema. Examples of presentations for relations are colored lines or arrows connecting the related objects and rectangles or ovals surrounding the objects. In the original design of Aquanet, it was expected that users would use the relations mechanism to represent the interconnections between information objects explicitly.

In actuality, Aquanet users did not use relations for most tasks. Instead, they defined visually distinctive object types and then spatially co-located related objects, sometimes creating visibly distinct areas clustering different types of information [Marshall, Rogers 92].

The observation of this use of the system led to work on creating mechanisms to recognize and use the information implicit in the spatial arrangement of information objects. The goal of deducing formal information from the analysis of informally represented information unites this work with work on the suggestion mechanisms in HOS. The difference from HOS is that instead of using mechanisms for analyzing text, the mechanisms built for Aquanet analyze spatial layouts. The last part of this chapter describes how these mechanisms can be used to support the incremental creation of Aquanet relations.

## 9.2 Analysis of Spatial Arrangements

People often use spatial methods for organizing information. White-boards and bulletin boards as well as some computer systems require people to lay out information in a two dimensional space. Sometimes the information is laid out in a seemingly random pattern but often spatial relationships between pieces of information are indicators of implicit relationships. People both follow conventions and develop new strategies for using these media to organize their information.

To recognize these spatial strategies and conventions and the structures that result from their application, one goal is to look for what structures are common across spatial information spaces. In order to avoid building mechanisms that are particular to a single user or to Aquanet an analysis of a number of spatial layouts from both computational and non-computational environments was performed.

Previous studies of how people organize material have been used to influence the design of user interface metaphors. For example, Malone's study of how people organize documents on their desktops [Malone 83] is realized by Mander et al.'s stack metaphor [Mander et al. 92]. This study was conducted with a different purpose in mind: instead of using an interface metaphor to make explicit actions natural to the user, the goal was to investigate ways of identifying and using implicit structure for the user's benefit without requiring that the structure ever be made explicit.

### 9.2.1 Data collection

The material for the analysis below was gathered from layouts people created while they were engaged in a variety of real problem-solving tasks. Data was collected from two non-electronic sources and eight electronic sources. Both non-electronic sources involved wall-sized spaces with cards (3x5 cards for the most part) pinned or taped to them. The electronic sources were from three different computer systems designed to support information structuring: Aquanet, NoteCards [Halasz et al. 87], and the Virtual Notebook System (VNS) [Gorry et al. 91]. All of these layouts had been created prior to the beginning of this analysis of common spatial strategies.

The first non-electronic layout was created by two researchers discussing the differences between a pair of computer-based editing paradigms. Their discussion resulted in over 70 3x5 cards pinned to a wall-sized bulletin board. The second wall-sized layout was the product of a facilitated meeting and included about 300 3x5 cards of different colors, PostIts, and sketches attached to a large sheet of butcher paper. Figure 31 shows this layout.

The VNS example involved an electronic notebook page where the system's developers tracked reported bugs. The NoteCards example involved three different spatial layouts an anthropologist used in analyzing data from his field work. Four different Aquanet examples were selected. Two involved information analysis tasks assessing machine translation systems, one was the discussion the developers engaged about system bugs and features (thus making it an example roughly equivalent to the VNS page), and the final example was generated by four participants during the early phase of authoring a paper.

A uniform representation for describing the spatial data was needed since each of these systems stores slightly different information about graphical objects used in a spatial layout, and because of the need to

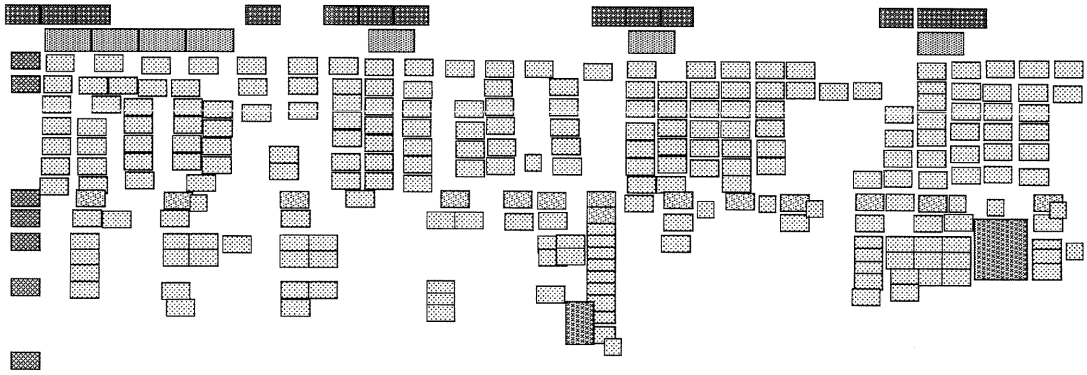


**Figure 31. Arrangement of Cards on a Wall**

The arrangement of a set of cards on a wall that were used to analyze consumer behavior for a product design. Shades of cards represent visually distinctive categories of cards.

represent the data from the two non-electronic sources. Individual information objects from each data source were encoded according to their x and y position in the arbitrary frame of reference imposed by each system, or by measured coordinates in a physical layout. Object extents were also recorded, along with an object type. For the non-electronic sources, type was either clarified by the author, or assigned on the basis of visual distinctiveness (different color notecards, different kinds of media). A few of the original sources included other graphical indicators of relations, like lines between two objects to signal a relationship; these graphics were omitted from this analysis.

For some of the electronic data, extracting position, extent, and type required conversion from its existing form. For example, Aquanet stores explicitly typed objects with an x-y position and an extent. However, the textual information objects extracted from NoteCards and VNS were not explicitly typed, nor did the NoteCards objects have system-assigned extents. These objects were assigned extents based on the length of their content (text objects that were three lines long were assigned three times the y extent as single-line objects), and type according to differences in font (like italics) or differences in visual markings (like boxes around the text). In the case of VNS the location data source differs slightly from the others since the objects were aligned with a snap-to-grid function.

### 9.2.2 Data characteristics

Table 2 summarizes characteristics of the collected data according to the uniform position-extent-type representation. The characteristics of the data for the organizing tasks varied considerably. Some were small layouts, assembled quickly; others were large layouts that emerged over several months, or in one case, several years. Six of the layouts resulted from collaborative tasks; the others were individual efforts. Five of the layouts--predominantly those used for an extended period of time--used a rich set of visually distinguishable types; the other five exhibited minimal object typing.

In addition to describing the quantitative characteristics of the data, the authors of the layouts were also solicited to provide their explanations of its layout. The largest example, an assessment of machine translation technology that had taken place over a two-year period, had its 1500 nodes partitioned into ten task-related areas where different structuring paradigms were applied; most of the other tasks had only one or two major areas, and many used a single structuring paradigm. With the authors' help, the arrangements were also assessed as to how structured they were--whether the structures were easily visually interpreted,

**Table 2: Summary of Spatial Layout Data Characteristics**

| task | task span | source | # of authors | # of objects | # of types |
|---|---|---|---|---|---|
| comparing editors | 1 day | non-electronic | 2 | 72 | 3 |
| product design analysis | 2-3 months | non-electronic | 2-3 | 284 | 8 |
| anthropological data analysis | 2-3 weeks | NoteCards | 1 | 36 | 3 |
| anthropological data analysis | 2-3 weeks | NoteCards | 1 | 20 | 3 |
| anthropological data analysis | 2-3 weeks | NoteCards | 1 | 50 | 3 |
| bug tracking (VNS) | 6 months | VNS | 7-8 | 63 | 6 |
| bug tracking (Aquanet) | 6 months | Aquanet | 4 | 130 | 6 |
| analysis of m.t. field | 2 years | Aquanet | 1 | 1506 | 14 |
| analysis of m.t. software | 2 months | Aquanet | 2 | 193 | 6 |
| selecting topics for a paper | 2 months | Aquanet | 4 | 74 | 3 |

or whether they were too loose to be understood by an outside reader. Table 3 summarizes the structure intended by the authors of the spatial layouts.

*Lists* were the predominant structure in the spatial arrangements; most of the lists were vertically aligned, and many were labelled. The recurring pattern of labels over or to the left of lists, and other spatially recurring patterns of object types formed another major category of structures identified, called *groups*. The use of matrices and some unstructured clustering of like objects was also observed. In a few places, objects were placed in *stacks* to save space at the expense of being able to see all of them at once. Examples of the three basic patterns, stacks, groups, and lists, are shown in Figure 32.

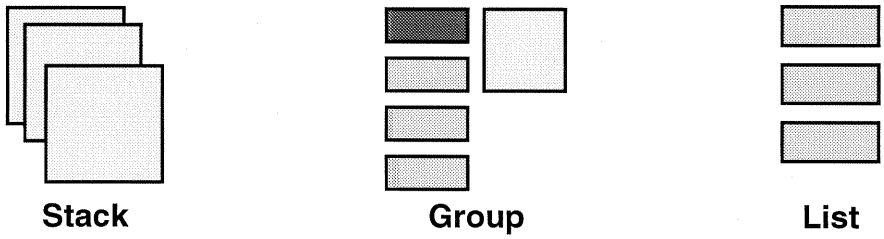



**Figure 32. Examples of Spatial Structure Classifications**

Stacks and lists have homogeneous components while groups are recurring patterns of heterogeneous components.

### 9.2.3 Grammar describing spatial structures

A descriptive grammar was developed to recognize or parse two-dimensional structure. This grammar uses a syntactic analysis of simple visual properties of graphic elements to guide automatic recognition and parsing of spatial structures.

**Table 3: Qualitative Spatial Layout Data Characteristics**

| task | author's organization | # of areas | degree of structure |
|---|---|---|---|
| comparing editors | Matrix (2 columns, 11 rows, multiple elements in each cell) and one labelled list whose elements are in horizontal pairs. | 2 | loosely structured |
| product design analysis | Matrix (23 columns, 7 rows, lists in some cells, 4 layers of labelling for the columns). | 1 | highly structured |
| anthropological data analysis | Five labelled vertical lists scattered in a labelled area. Each list has single annotation. | 1 | structured |
| anthropological data analysis | Five labelled lists with an overall horizontal tendency. Comment at the bottom summarizes entire area. | 1 | structured |
| anthropological data analysis | Two loosely structured columns with several imbedded lists. Area is labelled | 1 | loosely structured |
| bug tracking (VNS) | A work area containing 8 labelled lists laid out on a grid. Several other lists and individual objects labels and annotate the work area. | 1 | highly structured |
| bug tracking (Aquanet) | Nine loosely structured labelled lists, two unstructured, labelled clusters of like objects. | 3 | loosely structured |
| analysis of m.t. field | Ten areas, many containing lists of labelled lists or recurring patterns of structure. Some objects in stacks. | 10 | structured |
| analysis of m.t. software | Cluster of 16 like groups of objects and three unlabeled lists. Label summarizes larger of the two areas. | 2 | structured |
| selecting topics for a paper | Sequentially ordered row of 8 labelled lists. Items in the lists, the lists, and the area are annotated. | 1 | highly structured |

Spatial structures are composed of individual information objects based on a syntactic analysis of their visual properties. To perform a bottom-up parse of these structures, three general attributes of their constituent information elements are considered: type, location, and extent. Type assignment may be based on any distinguishing visual feature, such as color or shape. Location and extent are used to determine secondary characteristics, such as overlap and alignment. By combining this notion of visually distinguishable types and their spatial characteristics, it is possible to parse collections of lower-level objects into abstract structures.

Visual structures are recognized through spatial contiguity of their elements and the types of their elements. The structures that are recognized are the stacks, lists, and groups that were found to be common in the data collected. Stacks are homogeneous with respect to component object types and can be recognized based on the compactness or overlap of these components. Both lists and groups can be recognized based on the alignment of the component objects. Lists are composed of a homogeneous type while groups are composed of recurring mixtures of object types.

Intermediate visual structures can be composed of previously identified structures. Identifying intermediate structures requires reapplication of the same recognition algorithms that were used to induce the initial structures. Examples of intermediate structures are lists of lists or lists of groups. In recognizing intermediate level structure stacks and lists are considered to be of the same type as their component

objects while groups are assigned new types. Some kinds of complex or heterogeneous structures require user intervention to disambiguate them, since they are not readily syntactically recognized (for example, tables and matrices that involve overlapping vertical and horizontal lists).

The target macro-structures of the parse are the major areas of activity described by the authors. Areas are spatially contiguous x, y extents, although they may be non-rectangular and they may overlap. Heterogeneous areas will require user intervention to identify them (or they can be assumed to include the entire space). Homogeneous areas can be recognized automatically. Thus visual macrostructures will include "clumps", homogeneous areas of a single type that don't match alignment or overlap criteria for identifying them as lower level structures.

### 9.2.4 A sample parse of a spatial layout

Figure 33 shows a sample spatial layout of eight individual objects with three distinct types: Heading, Element, and Comment. It consists of a single area containing two vertical lists, one with three Elements and one with two, with Headings above them, and one Comment beside the second list.

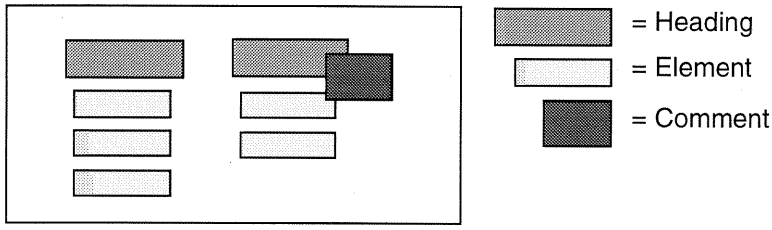



**Figure 33. A Sample Graphic Layout**

This spatial layout consists of eight objects, each being one of three object types.

Figure 34 shows a parse of the spatial layout from Figure 33. Non-terminal nodes of the graph reflect the primitive and intermediate structures that may be identified through analysis of spatial layout, including Horizontal list, Vertical list, and Group (which generates a new type). Terminal nodes show graphic depictions of the original types from the layout.

## 9.3 HairDo: Implementing a Spatial Grammar in Context of Aquanet

The data analysis and the resulting spatial grammar support the idea that automatic recognition of implicit structure is possible, albeit dependent on some degree of user intervention. A prototype parser was built to test some recognition algorithms based on the spatial grammar and to experiment with user interactions that guide the recognition process.

Part of the problem in allowing a computer system to take advantage of the previous analysis is the difficulty in recognizing the structure. There is no single "right" grouping and even individuals are inconsistent in their use of spatial layout to organize information. Objects are not always carefully aligned and may overlap. Different people may have a different opinion of what groupings, if any, are appropriate for a particular layout. The system can only conjecture one interpretation and operate under this interpretation until it is altered by a person.

The parsing process begins by looking for primitive visual structures, such as stacks, lists, and groups. The process then attempts to build composite structures by applying the same primitive recognition algorithms

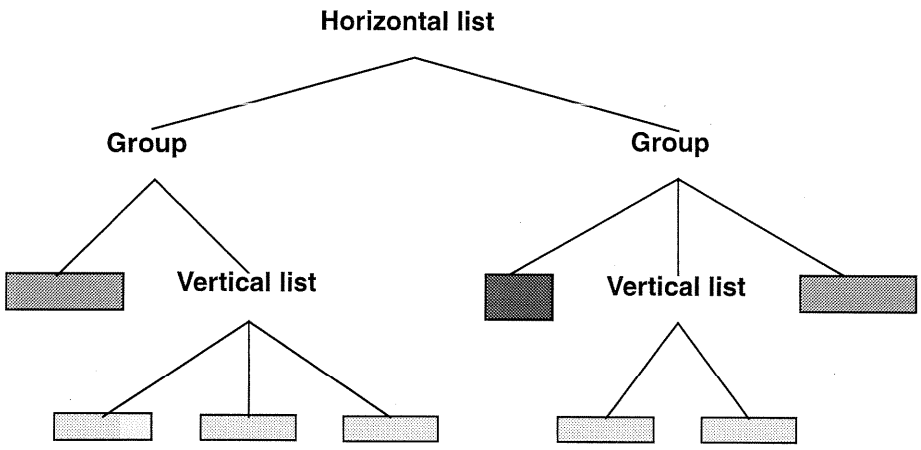



**Figure 34. The Parse Tree Generated using Visual Structure**

This is one possible parse of the diagram in Figure 33. Objects have been grouped into a hierarchy of relations based on spatial layout and object type.

to previously recognized structures. This bottom-up approach also takes advantage of some characteristics of the overall space such as average object size.

The recognition algorithms use the same visual and spatial characteristics discussed earlier, location, extent and type of each object. Types are only used in comparing whether two objects are of the same or a different type. This use of only very basic information stems from the belief that such mechanisms could be used in a wide variety of systems which use a two dimensional information space. Figure 35 shows the prototype system developed for evaluating the recognition algorithms and for investigating interaction mechanisms for enabling users to modify the recognition algorithms. The shading represents different types of recognized structure.

### 9.3.1 Comparing authors' intentions to computational results

As discussed earlier, the authors of the information spaces used the techniques to make the information more understandable for themselves, their colleagues, and for future readers. A question that remains is whether the recognition algorithms recognize the intended structure in the layouts. The author's intended structure is shown in the second column of Table 3. Comparing the results of the parser with the author's descriptions is made difficult because the authors discussed the high-level structure, such as "five labelled vertical lists", of the layouts while the parser's results include a large amount of the low or intermediate level structures. Qualitatively the parser was quite accurate at locating low to intermediate level structures. Lists and headed lists, stacks, and groups at the lowest level were most often recognized. The parser was successful, but less so, at combining these elements into the higher level structures intended. Interpretations of higher level structures were most often confounded by inconsistencies in the composition or spacing of intended structures.

Two types of recognition mistakes are possible: assuming structure that was not intended, and missing structure that was intended. The parser was generally conservative in its recognition of structure, and so did not often propose structure when none was intended. The conservative nature of the recognition

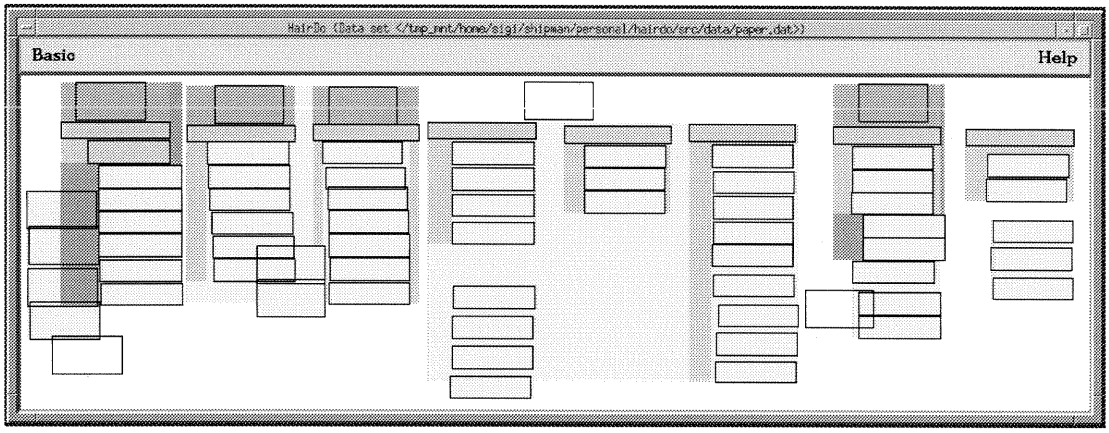



**Figure 35. Results of Parse of an Aquanet Discussion**

The original Aquanet objects are shown as wire frames. Differing types of inferred structure in the Aquanet discussion are shown by the different background shading.

algorithms was the result of the compounding problem of inferring structure from incorrectly inferred lower-level structures.

To provide a contrast to the human-organized layouts, a series of electronically-redistributed layouts were also created using the graphic object types and extents from each of the real layouts and used a random number generator to determine their new spatial locations within the area originally occupied by the objects. These random layouts were used to see if the automatic recognition algorithms found them as unstructured as they were to the human eye. Indeed, automatic recognition uncovered very little structure in these machine-generated layouts.

### 9.3.2 Interactions to correct recognition

By comparing the inferred structures with authors' characterizations of the organization of their layouts, it could be seen that the recognition algorithms often find intentional structure. But sometimes they do not. In the case where the user notices the system making the wrong assumption, the user needs the ability to correct what is wrong.

The user must be able, through minimal interaction, be able to change the recognition results. In the current implementation the user may "break" an existing inferred object or create new higher level objects. The user can also change options for the recognition process so that different results will be provided.

### 9.3.3 Limitations of recognition mechanisms

This work has looked at the implicit structure in spatial layouts and how this structure can be recognized by computers. To make this task tractable a number of assumptions were made about the layouts to be analyzed. One important assumption is that there are no lines or other connections between objects or that these can at least be filtered out. Also, layouts where information is not be grouped into neat objects with specified extents (this is the case in the use of white-boards and similar drawing surfaces) are not currently handled without some preprocessing to provide extents. These limitations point towards areas of future work required to produce more general mechanisms for layout understanding.

## 9.4 Using Recognized Structure to Support Knowledge-Base Evolution

What makes this work relevant to the topic of incremental formalization is the possible use of the recognized spatial structure to support the formal representation of relations between objects. A list or stack of objects can imply some common property of the elements in the list or stack. Likewise, a group can imply relations between the objects in the group.

Consider the sample layout shown in Figure 33. Given the parse of this layout shown in Figure 34 the system can now support the task of formally representing relations between the objects. This can be done by asking the user questions about possible interpretations of the diagram.

First, the system's location of groups can be used to ask about a possible relation between the objects in the group. Being the most prolific of the possible relations the system could ask about the relation between objects of type "Heading" and the objects of type "Element" below them. In this case the system could further aid in the formalization process by providing a list of known relation types that are possible for this combination of object types. Given a choice by the user the system can then generate five instances of that relation between the headings and elements. Likewise, the possible relation between the "Comment" object and the "Heading" object could be resolved with user interaction to produce the formal representation shown in Figure 36.

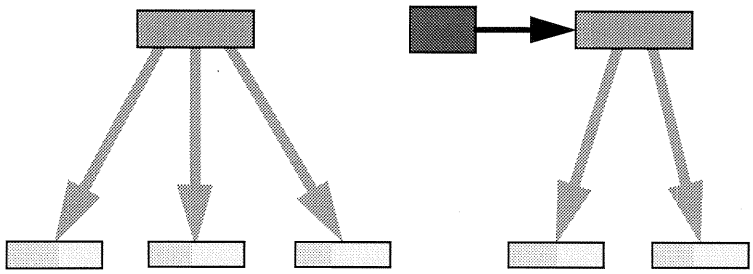


**Figure 36. Resulting Formal Representation of Visual Structure**

Using the parse tree shown in Figure 34, the system could aid the user in formalizing the relationships between the objects in Figure 33. This figure shows one possible set of logical relations that might result from this interaction.

The simple nature of this example layout does not demonstrate other possible cues for structure-based suggestions. Combining the interpretation of visual structure with the information encoded in the objects, suggestions can be made about possible missing information or orderings of objects. In the case of a list or stack of several objects which all but one have a certain formalized characteristic, the system might suggest that the missing characteristic be added to the non-conforming object.

The recognition of visual structure enables the system to propose the existence of relationships, even though the semantics of the relation is not known. For knowledge-based support to use these relationships some understanding of the semantics of the relationship must be elicited from the user. Like with the suggestions for new attributes for objects in HOS, the analysis of the informal information is used to aid the user in formalizing of implicit knowledge.

## 9.5 Summary

Although people are not always willing to explicitly represent the relationships among elements they use to construct a spatial layout, empirical work has shown that people use such structures implicitly. Not only are implicit structuring schemes common, but they are also governed by a set of conventions. That is, similar types of implicit spatial structures can be identified in layouts produced in different electronic systems, and by people using non-electronic media. This survey of ten sources provided a rich set of examples of these structures and underscored their domain-independence in analysis tasks.

Because people arrive at certain common interpretations of the organization two dimensional layouts without needing to understand the layouts' content, heuristic algorithms can be developed to recognize a layout's implicit structure. This work has developed such a set of algorithms, and have shown how they may be recursively applied to recognize not only primitive structures like lists, stacks, and groupings, but also more complex composite structures. These algorithms can be described by a simple grammar that uses some very basic--and very general--properties of a spatial layout. This grammar is based on spatial and visual properties of individual elements, their size and location, and their visually evident homogeneity or heterogeneity. This spatial grammar provides the ability to parse--although not unambiguously--the bulk of the layouts in the set of examples.

Recognition of the implicit structure in spatial layouts can be used to support the formalization of this structure in a manner similar to that used by the suggestion mechanisms in HOS. This can facilitate the evolution of the information space and can help provide the formalized information required for knowledge-based support mechanisms.

# *Chapter 10: Open Questions and Future Directions*

As is often the case, this work on incremental formalization probably raises more questions than it answers. Some of the questions concern extensions to incremental formalization as included in HOS. The formalization of declarative knowledge is supported in HOS, but the question remains as to how procedural knowledge, such as is formalized in agent objects, could be incrementally formalized and how that process could be supported? Perhaps methods from programming by demonstration [Myers 86] and user modelling [Kobsa, Wahlster 89] could be used to provide suggestions for formalizations, but that is a question left for future research. An approach more analogous to the current work is the application of spatial recognition algorithms, like those discussed in Chapter 9, to a visual programming language where spatial relationships between objects can define procedural or functional activity.

Another question about how users can be provided control over suggestion mechanisms. An interface could be added to enable users to add new rules to the suggestion mechanism based on the location of references in HOS. For example, in the case of the neurosciences information space, a new rule could suggest the relation "cells discussed" to the referenced objects with the attribute "type" having the value "cell". While the interactive definition of such suggestion rules would be nice for knowledge engineers seeding the information spaces, having users editing suggestion rule-bases is definitely not the solution to making formalization of domain information easier.

Another set of questions that arise concern system issues, such as what types of tasks cannot be supported using HOS, and what would be required to allow them to be supported by HOS? One answer to this question is that applications requiring computed values or considerable procedural expressiveness, such as simulation, are unlikely to be supported by the current version of HOS. Extensions to HOS's agent mechanisms could provide for greater applicability to these tasks by the inclusion of computed values for attributes, similar to that found in spreadsheets, and an object-oriented query and manipulation language for defining procedures on HOS objects.

Another question regarding systems like HOS, which combine domain-oriented application creation with the actual use of the application, is where is the cut-off between knowledge engineers building a domain-oriented application and the users of that application? The combining of the development and use environments is not unique to HOS. Spreadsheets also provide a substrate for creating domain-specific computations. As with spreadsheets, users of HOS would probably vary from "power users" building and modifying applications to users who avoid learning about most of the formalisms of the system.

A couple of questions arise from comparing the use of incremental formalization in HOS and Aquanet. One of these questions is where is the line between automatically using the inferred relations and waiting for the user to accept the suggestions? Because of the large numbers of structures being recognized within Aquanet, it is not appropriate to expect the user to "okay" every piece of recognized structure but HOS's suggestions are modifications of domain knowledge used to provide knowledge-base support. Some combination of the number and the use of suggestions must be considered in determining when acceptance by a user is a prerequisite for use of inferred structure. Another question arising from comparing HOS and Aquanet is how can textual and spatial recognition methods be integrated to improve suggestions. This is an open question which will hopefully be looked at during future research.

Another question that concerns the suggestion mechanisms is how could the use of natural language processing techniques improve the suggestion mechanisms. HOS's suggestion mechanisms use a lexicon to locate references to other objects, which are assumed to be interesting concepts. While current

suggestion mechanisms do not use natural language techniques, the use of name, place, or topic spotting algorithms for natural language text would likely be of use in helping the user add and modify the system's concept objects, from which the lexicon is created. Again, this is a question that requires more work to be answered.

In the case of work on incremental formalization, the future work has already begun. Currently, the design of a new system, a follow-up to both HOS and Aquanet, is underway. This system is "to allow uninhibited visual and semantic reorganization of the material, to cope with implicit structure, and to incrementally and partially formalize the semantic significance of the graphical expressions [Moran 93]."

The ubiquity of problems concerning formal representations leads to the possible widespread use of mechanisms enabling and supporting incremental formalization. The current trend towards even less formally represented information, such as found sketches made with pen-based interfaces, naturally leads towards the application of some type of incremental formalization in that arena as well.

# *Chapter 11: Conclusions*

Evolution is a critical problem for knowledge-based systems in rapidly changing domains and in domains dealing with ill-defined, design and analysis problems. The problems of cognitive overhead, tacit knowledge, premature structure, and situational structure are inherent in dealing with formal representations and hinder the acquisition of new and the modification of existing formally represented information.

To address these problems, an approach to knowledge-base evolution called incremental formalization has been introduced. In this approach, information is entered into the system in an informal representation and subsequently formalized with computer support. Incremental formalization fits well into the seeding, evolutionary growth, and reseeding paradigm of knowledge base evolution discussed in the context of domain-oriented design environments.

The Hyper-Object Substrate (HOS) supports incremental formalization through the integration of the capabilities of hypermedia and knowledge representation languages. By allowing users to choose the degree of formality for entering information, HOS reduces the up-front costs for users adding knowledge. In particular, domain 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 also supports in-place evolution with a set of tools which aid the user in transforming less formally represented information into formal representations, i.e. representations with which the computer can provide more services. Some of these tools merely provide information useful in formalizing knowledge. Other tools use the recognition of references or patterns within less formally represented information to suggest possible formalizations.

HOS has been used to create a number of domain-oriented systems. The largest of these domain-oriented systems is XNetwork, an environment to support the collaborative design of computer networks. Some extensions to HOS were made to better match the task of supporting design and the domain of computer networks. The development of XNetwork showed HOS's ability to support the different levels of abstraction and different types of domain knowledge needed for network design. The suggestion mechanisms in HOS were 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. These projects show that HOS can be used to build domain-oriented systems in a variety of domains and supporting a variety of tasks with little or no programming. The integration of hypermedia and knowledge representation languages provides interactivity not found in programming and knowledge representation languages combined with expressiveness enabling support not found in standard hypermedia systems.

During the development of these domain-oriented systems, the potential problems of cognitive overhead, tacit knowledge, premature structure, and the situational nature of structure were reiterated. Namely, 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 of previous formalisms. This evolving nature of understanding implies that problems concerned with formal representations cannot be considered just an interface problem.

The generality of incremental formalization was explored in the evaluation of structure recognition within the context of Aquanet, a system to support knowledge structuring. Experiences with Aquanet showed that users were not using the system's formal representational abilities. To compensate, recognition mechanisms were built which can locate structure implicit in the spatial layout of Aquanet discussions. The success of these extensions demonstrated the usefulness of tool support for incremental formalization. Furthermore, the recognition of structures within Aquanet provides an example of the application of incremental formalization in a non-textual interface and a suggestion mechanism that uses knowledge of common patterns of spatial layout rather than domain knowledge in making suggestions. The success of these mechanisms has led to the design of a new system where such recognition and incremental formalization of implicit structure is to be the normal mode of operation.

Some open questions are how can procedural information be incrementally formalized and how can the textual and graphical techniques for creating suggestions be combined. On-going systems development, extending the HOS and Aquanet frameworks, will focus on some of these open questions about incremental formalization and point out additional interesting directions for this line of research.

# *Bibliography*


[Aho et al. 74]
Aho, A.V., Hopcroft, J.E., and Ullman, J.D. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, Mass., 1974, pp. 60-67.

[Akscyn et al. 88]
Akscyn, R.M., McCracken, D.L., and Yoder, E.A. KMS: A Distributed Hypermedia System for Managing Knowledge in Organizations. *Communications of the ACM* 31, 7 (July 1988), pp. 820-835.

[Ballance, Graham 91]
Ballance, R.A. and Graham, S.L. Incremental Consistency Maintenance for Interactive Applications. In *Logic Programming: Proceedings of the Eighth International Conference*, K. Furukawa, Ed. MIT Press, Cambridge, Mass., 1991, pp. 895-909.

[Barman 91]
Barman, D. *RelType: Relaxed Typing for Intelligent Hypermedia Representations.* Brown University Technical Report, CS-91-26, April 1991.

[Berlin, O'Day 90]
Berlin, L., and O'Day, V. Platform and Application Issues in Multi-User Hypertext. In *Multi-User Interfaces and Applications*, S. Gibbs, A. Verrijn, Eds., Amsterdam: North-Holland, 1990, pp. 293-309.

[Bernstein 90]
Bernstein, M. An Apprentice That Discovers Hypertext Links. In *Proceedings of the European Conference on Hypertext (ECHT'90)* (Paris, France, Nov.). 1990, pp. 212-223.

[Brooks 87]
Brooks Jr., F.P. No Silver Bullet: Essence and Accidents of Software Engineering. *IEEE Computer* 20, 4 (April 1987), pp. 10-19.

[Brunet et al. 91]
Brunet, L.W., Morrissey, C.T., and Gorry, G.A. Oral History and Information Technology: Human Voices of Assessment. *Journal of Organizational Computing* 1, 3 (1991), pp. 251-274.

[Bullen, Bennett 90]
Bullen, C.V., and Bennett, J.L. Learning From User Experience With Groupware. In *Proceedings of the Conference on Computer-Supported Cooperative Work (CSCW'90)* (Los Angeles, Calif., Oct. 7-10). ACM, New York, 1990, pp. 291-302.

[Carlson, Ram 90]
Carlson, D., and Ram, S. HyperIntelligence: The Next Frontier. *Communications of the ACM* 33, 3 (March 1990), pp. 311-321.

[Case et al. 90]
Case, J., Fedor, M., Schoffstall, M., and Davin, J. *A Simple Network Management Protocol.* RFC 1157, DDN Network Information Center, SRI International, May 1990.

[Chomsky 56]
Chomsky, N. Three Models for the Description of Language. *IRE Transactions on Information Theory* 2, 3 (1956), pp. 113-124.

[Conklin 87]
Conklin, J. Hypertext: An Introduction and Survey. *IEEE Computer* 20, 9 (Sept. 1987), pp. 17-41.

[Conklin, Begeman 88]
Conklin, J., and Begeman, M. gIBIS: A Hypertext Tool for Exploratory Policy Discussion. In *Proceedings of the Conference on Computer-Supported Cooperative Work (CSCW'88)* (Portland, Oregon, Sept. 26-28). ACM, New York, 1988, pp. 140-152.

[Conklin, Yakemovic 91]
Conklin, E.J., and Yakemovic, K.C. A Process-Oriented Approach to Design Rationale. *Human Computer Interaction* 6, 3-4 (1991), pp. 357-391.

[Davis 84]
Davis, R. Interactive Transfer of Expertise. In *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*, B.G. Buchanan, E.H. Shortliffe, Eds. Addison-Wesley, Reading, Mass., 1984, pp. 171-205.

[Dumais et al. 88]
Dumais, S.T., Furnas, G.W., Landauer, T.K., Deerwester, S., and Harshman, R. Using Latent Semantic Analysis to Improve Access to Textual Information. In *Human Factors in Computing Systems, CHI'88 Conference Proceedings* (Washington D.C., May). ACM, New York, 1988, pp. 281-285.

[Ellis et al. 91]
Ellis, C., Gibbs, S., and Rein, G. GroupWare: Some Issues and Experiences. *Communications of the ACM* 34, 1 (Jan. 1991), pp. 38-58.

[Eriksson 91]
Eriksson, H. *Meta-Tool Support for Knowledge Acquisition.* Linköping Studies in Science and Technology, Dissertations, No. 244, Linköping, Sweden, 1991.

[Eshelman et al. 87]
Eshelman, L., Ehret, D., McDermott, J., and Tan, M. MOLE: A Tenacious Knowledge-Acquisition Tool. *International Journal of Man-Machine Studies* 26, (1987), pp. 41-54.

[Fischer et al. 89]
Fischer, G., McCall, R., and Morch, A. JANUS: Integrating Hypertext with a Knowledge-Based Design Environment. In *Proceedings of Hypertext '89* (Pittsburgh, Penn., Nov. 5-8). ACM, New York, 1989, pp. 105-117.

[Fischer et al. 91]
Fischer, G., Lemke, A.C., McCall, R., and Morch, A. Making Argumentation Serve Design. *Human Computer Interaction* 6, 3-4 (1991), pp. 393-419.

[Fischer et al. 92]
Fischer, G., Grudin, J., Lemke, A.C., McCall, R., Ostwald, J., Reeves, B.N., and Shipman, F. Supporting Indirect, Collaborative Design with Integrated Knowledge-Based Design Environments. *Human Computer Interaction* 7, 3 (Fall 1992), pp. 281-314.

[Fischer, Girgensohn 90]
Fischer, G., and Girgensohn, A. End-User Modifiability in Design Environments. In *Human Factors in Computing Systems, CHI'90 Conference Proceedings* (Seattle, Wash., April). ACM, New York, 1990, pp. 183-191.

[Fischer, Nieper-Lemke 89]
Fischer, G., and Nieper-Lemke, H. HELGON: Extending the Retrieval by Reformulation Paradigm. In *Human Factors in Computing Systems, CHI'89 Conference Proceedings* (Austin, Texas, May). ACM, New York, 1989, pp. 357-362.

[Fischer, Reeves 92]
Fischer, G., and Reeves, B.N. Beyond Intelligent Interfaces: Exploring, Analyzing and Creating Success Models of Cooperative Problem Solving. *Applied Intelligence, Special Issue Intelligent Interfaces* 1, (1992), pp. 311-332.

[Frawley et al. 92]
Frawley, W.J., Piatetsky-Shapiro, G., and Matheus, C.J. Knowledge Discovery in Databases: An Overview. *AI Magazine* 13, 3 (Fall 1992), pp. 57-70.

[Furnas et al. 87]
Furnas, G.W., Landauer, T.K., Gomez, L.M., and Dumais, S.T. The Vocabulary Problem in Human-System Communication. *Communications of the ACM* 30, 11 (Nov. 1987), pp. 964-971.

[Ginsberg et al. 85]
Ginsberg, A., Weiss, S., and Politakis, P. SEEK2: A Generalized Approach to Automatic Knowledge Base Refinement. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI '85).* 1985, pp. 367-374.

[Girgensohn 92]
Girgensohn, A. *End-User Modifiability in Knowledge-Based Design Environments.* Ph.D. Dissertation., Department of Computer Science, University of Colorado, Boulder, Colorado, 1992.

[Girgensohn, Shipman 92]
Girgensohn, A., and Shipman, F. End-User Modifiability: Tools and Representations. In *Proceedings of the Symposium for Applied Computing (SAC '92)* (Kansas City, Missouri, March 1-3). ACM, New York, 1992, pp. 340-348.

[Gorry et al. 78]
Gorry, G.A., Chamberlain, R.M., Price, B.S., DeBakey, M.E., and Gotto, A.M. Communication Patterns in a Biomedical Research Center. *Journal of Medical Education* 53, (1978), pp. 206-208.

[Gorry et al. 88]
Gorry, G.A., Burger, A., Chaney, J., Long, K., and Tausk, C. Computer Support for Biomedical Research Groups. In *Proceedings of the Conference on Computer-Supported Cooperative Work (CSCW'88)* (Portland, Oregon, Sept. 26-28). ACM, New York, 1988, pp. 39-51.

[Gorry et al. 91]
Gorry, G.A., Long, K.B., Burger, A.M., Jung, C.P., and Meyer, B.D. The Virtual Notebook System: An Architecture for Collaborative Work. *Journal of Organizational Computing* 1, 3 (1991), pp. 233-250.

[Grudin 88]
Grudin, J. Why CSCW Applications Fail: Problems in the Design and Evaluation of Organizational Interfaces. In *Proceedings of the Conference on Computer-Supported Cooperative Work (CSCW'88)* (Portland, Oregon, Sept. 26-28). ACM, New York, 1988, pp. 85-93.

[Halasz et al. 87]
Halasz, F.G., Moran, T.P., and Trigg, R.H. NoteCards in a Nutshell. In *Human Factors in Computing Systems and Graphics Interface, CHI+GI'87 Conference Proceedings* (Toronto, Canada, April). ACM, New York, 1987, pp. 45-52.

[Halasz 88]
Halasz, F. Reflections on NoteCards: Seven Issues for the Next Generation of Hypermedia Systems. *Communications of the ACM* 31, 7 (July 1988), pp. 836-852.

[Halasz 91]
Halasz, F. "'Seven Issues': Revisited." Hypertext '91 Keynote Talk, (San Antonio, Texas, Dec. 15-18). 1991.

[Hart 88]
Hart, A. Knowledge Acquisition for Expert Systems. In *Knowledge, Skill, and Artificial Intelligence*, B. Göranzon, I. Josefson, Eds. Springer, Heidelberg, 1988.

[Henninger 93]
Henninger, S. *Locating Relevant Examples for Example-Based Software Design*. Ph.D. Dissertation., Department of Computer Science, University of Colorado, Boulder, Colorado, 1993.

[Hofmann et al. 90]
Hofmann, M., Schreiweis, U., and Langendörfer, H. An Integrated Approach of Knowledge Acquisition by the Hypertext System CONCORDE. In *Hypertext: Concepts Systems and Applications*, A. Rizk, N. Streitz, and J. Andre', Eds. Cambridge University Press, Cambridge, UK, 1990, pp. 166-179.

[Hollan et al. 91]
Hollan, J., Rich, E., Hill, W., Wroblewski, D., Wilner, W., Wittenburg, K., and Grudin, J. An Introduction to HITS: Human Interface Tool Suite. In *Intelligent User Interfaces*, J. Sullivan, S. Tyler, Eds. ACM, New York, 1991, pp. 293-338.

[Hutchins et al. 86]
Hutchins, E., Hollan, J., and Norman, D. Direct Manipulation Interfaces. In *User Centered System Design*, D. Norman, S. Draper, Eds. Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1986, pp. 87-124.

[Jarczyk et al. 92]
Jarczyk, A., Löffler, P., and Shipman, F. Design Rationale for Software Engineering: A Survey. In *Proceedings of the 25th Annual Hawaii International Conference on System Sciences Vol. 2 (HICSS-92)* (Jan.). IEEE, 1992, pp. 577-586.

[Jordan et al. 89]
Jordan, D.S., Russell, D.M., Jensen, A.-M.S., and Rogers, R.A. Facilitating the Development of Representations in Hypertext with IDE. In *Proceedings of Hypertext '89* (Pittsburgh, Penn., Nov. 5-8). ACM, New York, 1989, pp. 93-104.

[Kaindl, Snaprud 91]
Kaindl, H., and Snaprud, M. Hypertext and Structured Object Representation: A Unifying View. In *Proceedings of Hypertext '91* (San Antonio, Texas, Dec. 15-18). ACM, New York, 1991, pp. 345-358.

[Kennedy 88]
Kennedy, P. *The Rise and Fall of the Great Powers*. Random House, New York, 1988.

[Kennedy 93]
Kennedy, P. *Preparing for the Twenty-First Century*. Random House, New York, 1993.

[Kobsa, Wahlster 89]
Kobsa, A. and Wahlster, W. (Eds.) *User Models in Dialog Systems*. Springer, Berlin, Germany, 1989.

[Krasner, Pope 88]
Krasner, G.E., and Pope, S.T. *A Cookbook for using the Model-View-Controller User Interface Paradigm in Smalltalk-80*. Technical Report, ParcPlace Systems, Palo Alto, Calif., January, 1988.

[Kunz, Rittel 70]
Kunz, W., and Rittel, H.W.J. *Issues as Elements of Information Systems*. Working Paper 131, Center for Planning and Development Research, University of California, Berkeley, Calif., 1970.

[Lakin 87]
Lakin, F. Visual Grammars for Visual Languages. In *Proceedings of AAAI 87* (Seattle, Wash., July). 1987, pp. 683-688.

[Lee 90]
Lee, J. SIBYL: A Qualitative Decision Management System. In *Artificial Intelligence at MIT: Expanding Frontiers*, P. Winston, S. Shellard, Eds. The MIT Press, Cambridge, Mass., 1990, pp. 104-133.

[Lethbridge, Skuce 92a]
Lethbridge, T.C., and Skuce, D. Informality in Knowledge Exchange. In *Proceedings of the AAAI Workshop on Knowledge Representation Aspects of Knowledge Acquisition* (San Jose, Calif., July). 1992.

[Lethbridge, Skuce 92b]
Lethbridge, T.C., and Skuce, D. *CODE4: Feature Summary*. Technical Report, Department of Computer Science, University of Ottawa, 1992.

[Lieberman 86]
Lieberman, H. Using Prototypical Objects to Implement Shared Behavior in Object-Oriented Systems. In *OOPSLA 1986 Conference Proceedings* (Portland, Oregon, Sept. 29-Oct. 2). 1986, pp. 214-223.

[MacLean et al. 89]
MacLean, A., Young, R., and Moran, T. Design Rationale: The Argument behind the Artifact. In *Human Factors in Computing Systems, CHI '89 Conference Proceedings* (Austin, TX). ACM, New York, 1989, pp. 247-252.

[Malone 83]
Malone, T.W. How do People Organize their Desks? Implications for the Design of Office Information Systems. *ACM Transactions on Office Information Systems* 1, 1 (January 1983), pp. 99-112.

[Malone et al. 86]
Malone, T.W., Grant, K.R., Lai, K.-Y., Rao, R., and Rosenblitt, D. Semi-Structured Messages are Surprisingly Useful for Computer-Supported Coordination. In *Proceedings of the Conference on Computer-Supported Cooperative Work (CSCW'86)* (Austin, Texas, Dec.). 1986, pp. 102-114.

[Malone et al. 92]
Malone, T.W., Lai, K.Y., and Fry, C. Experiments with Oval: A Radically Tailorable Tool for Cooperative Work. In *Proceedings of the Conference on Computer Supported Cooperative Work (CSCW '92)* (Toronto, Canada, Oct. 31-Nov. 4). ACM, New York, 1992, pp. 289-297.

[Mander et al. 92]
Mander, R., Salomon, G., and Wong, Y.Y. A 'Pile' Metaphor for Supporting Organization of Information. In *Human Factors in Computing Systems, CHI '92 Conference Proceedings* (Monterey, Calif., May 3-7). ACM, New York, 1992, pp. 627-634.

[Markus, Connolly 90]
Markus, M., and Connolly, T. "Why CSCW Applications Fail: Problems in the Adoption of Interdependent Work Tools." In *Proceedings of the Conference on Computer-Supported Cooperative Work (CSCW'90)* (Los Angeles, Calif., Oct. 7-10). ACM, New York, 1990, pp. 371-380.

[Marshall 87]
Marshall, C.C. Exploring Representation Problems Using Hypertext. In *Proceedings of Hypertext '87* (Chapel Hill, N.C., Nov. 13-15). ACM, New York, 1987, pp. 253-268.

[Marshall et al. 91]
Marshall, C., Halasz, F., Rogers, R., and Janssen, W. Aquanet: a hypertext tool to hold your knowledge in place. In *Proceedings of Hypertext '91* (San Antonio, Texas, Dec. 15-18). ACM, New York, 1991, pp. 261-275.

[Marshall, Rogers 92]
Marshall, C.C., and Rogers, R.A. Two Years before the Mist: Experiences with Aquanet. In *Proceedings of European Conference on Hypertext (ECHT '92)* (Milano, Italy, Dec. 1992). pp. 53-62.

[McCall 87]
McCall, R. PHIBIS: Procedurally Hierarchical Issue-Based Information Systems. In *Proceedings of the Conference on Architecture at the International Congress on Planning and Design Theory.* American Society of Mechanical Engineers, New York, 1987.

[McCall et al. 81]
McCall, R., Mistrik, I., and Schuler, W. An Integrated Information and Communication System for Problem Solving. In *Proceedings of the Seventh International CODATA Conference.* Pergamon, London, 1981.

[McCall et al. 83]
McCall, R., Schaab, B., and Schuler, W. An Information Station for the Problem Solver: System Concepts. In *Applications of Mini- and Microcomputers in Information, Documentation and Libraries*, C. Keren, L. Perlmutter, Eds. New York: Elsevier, 1983.

[McCall et al. 90]
McCall, R., Bennett, P., d'Oronzio, P., Ostwald, J., Shipman, F., and Wallace, N. PHIDIAS: Integrating CAD Graphics into Dynamic Hypertext. In *Proceedings of the European Conference on Hypertext (ECHT'90)* (Paris, France, Nov.). 1990, pp. 152-165.

[McCall et al. 91]
McCall, R., Ostwald, J., and Shipman, F. Supporting Designers' Access to Information Through Virtually Structured Hypermedia. In *Proceedings of the 1991 Conference on Intelligent Computer Aided Design.* 1991, pp. 116-127.

[Minsky 75]
Minsky, M. A Framework for Representing Knowledge. In *The Psychology of Computer Vision*, P. Winston, Ed. McGraw-Hill Book Company, New York, 1975, pp. 211-277.

[Mittal, Dym 85]
Mittal, S., and Dym, C.L. Knowledge Acquisition from Multiple Experts. *The AI Magazine* (Summer, 1985), pp. 32-36.

[Monty 90]
Monty, M.L. *Issues for Supporting Notetaking and Note Using in the Computer Environment.* Dissertation, Department of Psychology, University of California, San Diego, 1990.

[Moran 93]
Moran, T., Theme: Deformalizing Computer and Communication Systems. In *Proceedings of InterCHI Research Symposium* (Amsterdam, Netherlands, April 23-24). 1993.

[Mundie 91]
Mundie, D.A., and Shultis, J.C. (Eds.) *Proceedings of the Workshop on Informal Computing.* (Santa Cruz, Calif., May). 1991.

[Musen 89]
Musen, M. An Editor for the Conceptual Models of Interactive Knowledge-Acquisition tools. *International Journal of Man-Machine Studies* 31 (1989), pp. 673-698.

[Myers 85]
Myers, W. MCC: Planning the Revolution in Software. *IEEE Software* (November 1985), pp. 68-73.

[Myers 86]
Myers, B.A. Visual Programming, Programming by Example, and Program Visualization: A Taxonomy. In *Human Factors in Computing Systems, CHI '86 Conference Proceedings* (Boston, Mass., April). ACM, New York, 1986, pp. 59-66.

[Nakakoji, Fischer 90]
Nakakoji, K. and Fischer, G. Catalog Explorer: Exploiting the Synergy of Integrated Design Environments. In *Proceedings of Software Symposium '90* (Kyoto, Japan). June, 1990, pp. 264-271.

[Nanard, Nanard 91]
Nanard, J., and Nanard, M. Using Structured Types to incorporate Knowledge in Hypertext. In *Proceedings of Hypertext '91* (San Antonio, Texas, Dec. 15-18). ACM, New York, December, 1991, pp. 329-343.

[Nardi, Miller 90]
Nardi, B.A., and Miller, J.R. An Ethnographic Study of Distributed Problem Solving in Spreadsheet Development. In *Proceedings of the Conference on Computer-Supported Cooperative Work (CSCW'90)* (Los Angeles, Calif., Oct. 7-10). ACM, New York, 1990, pp. 197-208.

[Nelson 67]
Nelson, T. Getting It Out of Our System. In *Information Retrieval: A Critical Review*, G. Scheckter, Ed. Thompson Books, Washington D.C., 1967.

[Nemeth 91]
Nemeth, E. SA-Tool, A System Administrator's Cockpit. In *Proceedings of the Usenix Conference.* 1991, pp. 193-205.

[Norman 86]
Norman, D. Cognitive Engineering. In *User Centered System Design*, D. Norman, S. Draper, Eds. Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1986, pp. 31-61.

[Peper et al. 89]
Peper, G., MacIntyre, C., and Keenan, J. Hypertext: A New Approach for Implementing an Expert System. In *Proceedings of 1989 ITL Expert Systems Conference*, 1989.

[Polanyi 66]
Polanyi, M. *The Tacit Dimension*, Doubleday, Garden City, NY, 1966.

[Politakis, Weiss 84]
Politakis, P., and Weiss, S. Using Empirical Analysis to Refine Expert System Knowledge Bases. *Artificial Intelligence* 22, 1 (Jan. 1984), pp. 23-48.

[Rama, Srinivasan 93]
Rama, D.V., and Srinivasan, P. An Investigation of Content Representation Using Text Grammars. *ACM Transactions on Information Systems* 11, 1 (Jan. 1993), pp. 51-75.

[Reeves 93]
Reeves, B. *Supporting Collaborative Design by Embedding Communication and History in Design Artifacts.* Ph.D. Dissertation., Department of Computer Science, University of Colorado, Boulder, Colorado, 1993.

[Reeves, Shipman 92]
Reeves, B.N., and Shipman, F.M. Supporting Communication between Designers with Artifact-Centered Evolving Information Spaces. In *Proceedings of the Conference on Computer Supported Cooperative Work (CSCW '92)* (Toronto, Canada, Oct. 31-Nov. 4). ACM, New York, 1992, pp. 394-401.

[Rein, Ellis 91]
Rein, G., and Ellis, C. rIBIS: a real-time group hypertext system. *International Journal of Man-Machine Studies* 34, 3 (March 1991), pp. 349-367.

[Remde et al. 87]
Remde, J.R., Gomez, L.M., and Landauer, T.K. SuperBook: An automatic tool for information exploration - hypertext? In *Proceedings of Hypertext '87* (Chapel Hill, N.C., Nov. 13-15). ACM, New York, 1987, pp. 175-188.

[Rittel 84]
Rittel, H. Second-Generation Design Methods. In *Developments in Design Methodology*, N. Cross, Ed. John Wiley & Sons, New York, 1984, pp. 317-327.

[Russell 90]
Russell, D. Hypermedia and Representation. In *Hypertext und Hypermedia: Von Theoretischen Konzepten zur Practischen Anwendung*, P.A. Gloor and N.A. Streitz, Eds. Springer-Verlag, Berlin, 1990, pp. 1-9.

[Schank 90]
Schank, R. *Tell Me A Story: A new look at real and artificial memory.* Charles Scribner's Sons, New York, 1990.

[Schäuble, Glavitsch 90]
Schäuble, P., and Glavitsch, U. A Probabilistic Retrieval Model Based on Hidden Markov Models. In *Proceedings of the Workshop on Intelligent Access to Information Systems* (Darmstadt, Germany). 1990, pp. 43-46.

[Schön 83]
Schön, D. *The Reflective Practitioner: How Professionals Think in Action.* Basic Books, New York, 1983.

[Schwabe et al. 90]
Schwabe, D., Feijo, B., and Krause, W. Intelligent Hypertext for Normative Knowledge in Engineering. In *Hypertext: Concepts Systems and Applications*, A. Rizk, N. Streitz, and J. Andre', Eds. Cambridge University Press, Cambridge, UK, 1990, pp. 123-136.

[Shipman et al. 89]
Shipman, F.M., Chaney, R.J., and Gorry, G.A. Distributed Hypertext for Collaborative Research: The Virtual Notebook System. In *Proceedings of Hypertext '89* (Pittsburgh, Penn., Nov. 5-8). ACM, New York, 1989, pp. 129-135.

[Shipman 93]
Shipman, F.M. *Hyper-Object Substrate (HOS) User's Manual.* To appear as Technical Report, Department of Computer Science, University of Colorado, Boulder, 1993.

[Shipman, Marshall 93]
Shipman, F.M., and Marshall, C.C. *Formality Considered Harmful: Experiences, Emerging Themes, and Directions.* Technical Report CU-CS-648-93, Department of Computer Science, University of Colorado, Boulder, 1993.

[Simon 81]
Simon, H. *The Sciences of the Artificial.* MIT Press, Cambridge, MA, 1981.

[Stahl 93]
Stahl, G. *Interpretation in Design: The Problem of Tacit and Explicit Understanding in Computer Support of Cooperative Design.* Ph.D. Dissertation., Department of Computer Science, University of Colorado, Boulder, Colorado, 1993.

[Stevens 93]
Stevens, C. *Helping Users Locate and Organize Information.* Ph.D. Dissertation., Department of Computer Science, University of Colorado, Boulder, Colorado, 1993.

[Suchman 87]
Suchman, L.A. *Plans and Situated Actions: The problem of human-machine communication.* Cambridge University Press, Cambridge, UK, 1987.

[Terveen et al. 91]
Terveen, L., Wroblewski, D., and Tighe, S. Intelligent Assistance through Collaborative Manipulation. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91)* (Aug.). 1991, pp. 9-14.

[Tesler 81]
Tesler, L. The Smalltalk Environment. *Byte* 6, 8 (Aug. 1981), p. 90.

[Toulmin 58]
Toulmin, S. (Ed.) *The Uses of Argument.* Cambridge University Press, UK, 1958.

[Ungar, Smith 87]
Ungar, D., and Smith, R.B. Self: The Power of Simplicity. In *OOPSLA 1987 Conference Proceedings* (Orlando, Florida, Oct. 4-8). ACM, New York, 1987, pp. 227-242.

[Walker 87]
Walker, J.H. Document Examiner: Delivery Interface for Hypertext Documents. In *Proceedings of Hypertext '87* (Chapel Hill, N.C., Nov. 13-15). ACM, New York, 1987, pp. 307-323.

[Waterman 86]
Waterman, D.A. *A Guide to Expert Systems.* Addison-Wesley, 1986, pp. 3-11.

[Winograd, Flores 86]
Winograd, T., and Flores, F. *Understanding Computers and Cognition: A New Foundation for Design.* Ablex Publishing Corporation, Norwood, NJ, 1986.

[Yakemovic, Conklin 90]
Yakemovic, K.C., and Conklin, E.J. Report of a Development Project Use of an Issue-Based Information System. In *Proceedings of the Conference on Computer-Supported Cooperative Work (CSCW'90)* (Los Angeles, Calif., Oct. 7-10), ACM, New York, 1990, pp. 105-118.

# *Author Index*

# *Subject Index*