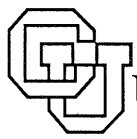
Supporting Knowledge-Base Evolution using Multiple Degrees of Formality

Frank M. Shipman III

CU-CS-592-92 April 1992



University of Colorado at Boulder
DEPARTMENT OF COMPUTER SCIENCE

Supporting Knowledge-Base Evolution using Multiple Degrees of Formality

Frank M. Shipman III

Ph.D. Dissertation Proposal
Department of Computer Science and Institute for Cognitive Science
University of Colorado, Boulder
shipman@cs.colorado.edu
(303) 492 - 1218

Introduction:

The topic of this Ph.D. proposal is the use of representations of varying degrees of formality in supporting knowledge-base evolution. The work will be done in the context of a computer network design environment. In the way I will use the terms, representations are formal/informal in relation to a type of knowledge. Text strings are formal representations for knowledge about the order of a set of characters but are informal in relation to the semantic content of sentences.

There are three classes of representations with regards to the degree of formality that I will differentiate between: informal, formal, and semiformal. Informal knowledge is knowledge in a form not processable by the computer without using heuristic techniques. Examples of informal knowledge are the semantic content of text, images, and audio or video recordings. Formal knowledge is knowledge which is computer processable, such as objects in inheritance hierarchies, causal information in production rules, and the order of characters in a text string. Semiformal knowledge representations combine informal and formal knowledge. Examples of semiformal representations are those used by design rationale and hypermedia systems, in which chunks of text and/or other media are connected by a set of machine interpretable relations.

In order to better support the evolutionary process of design, as described below, I will provide tools to support the evolution of knowledge from less formal representations to more formal representations. These tools will use the limited domain of the system (in my case computer network design) to partially understand informal knowledge in or being added to the system. This limited understanding will be used to aid the designer in integrating the information in the informal knowledge with formal knowledge already in the knowledge base. Support for the conversion of knowledge from user-friendly to machine-processable representations facilitates domain experts in modifying the knowledge in knowledge-based systems. Conversion in the other direction, from machine-processable to user-friendly representations is also needed to support the comprehension of knowledge, but my focus will be on supporting formalization.

The next section contains a discussion of the problem and theoretical framework. This is followed by a description of related work and how it differs from my approach. Then comes a description of the system as it currently exists, the basic components that still need to be added, and the integration of tools supporting knowledge-base evolution. Finally there is are two scenarios and a discussion of the evaluation methods. An appendix has been added to provide more detail about the knowledge and mechanisms used.

Statement of the Problem:

Design environments, which consist of a number of components integrated to support the process of design, have been discussed in [Fischer et al.1989] and [Fischer et al. 1991]. These design environments include a number of different mechanisms for representing knowledge, including semiformal and formal knowledge representations with respect to domain knowledge (see Figure 1).

The argumentation mechanism in design environments includes a semiformal knowledge representation in the form of the Procedural Hierarchy of Issues (PHI) method [McCall 1987] of issue-based deliberation [Kunz, Rittel 1970]. PHI contains three basic types of nodes: issues, answers, and arguments. The palette of design items in a design environment is represented as an inheritance hierarchy of objects with attributes and values. Critics, which link the construction and argumentation components of a design environment, are represented as rules like those found in rule-based systems but whose actions are limited to providing the designer with messages and links into argumentation.

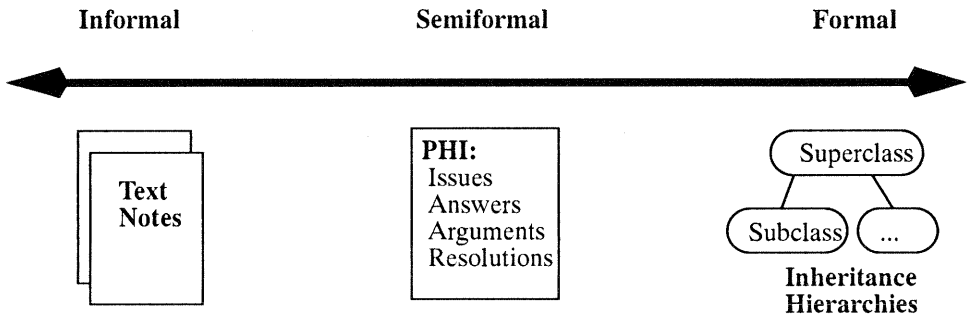


FIGURE 1. Knowledge in our system is represented in a variety of methods ranging from informal, like text notes, to formal, like inheritance hierarchies and production rules.

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. 1981]. Tools supporting the designer in modifying and creating palette items and critic rules have been developed [Fischer, Girgensohn 1990] and general principles for achieving end-user modifiability have been outlined [Girgensohn, Shipman 1992]. No one has examined how knowledge represented in one representation can be transferred to other representations in these design environments. My goal is to look at mechanisms to support such transference.

The different representations used in a design environment were chosen to be appropriate to a type of information needed by the designer or system. By having multiple representations designers have the choice to deal with information in different ways. Allowing this choice becomes critical when designers decide whether to add to or modify the information in the knowledge base. When the designer can choose a representation that matches the information, the "accidental complexity" [Brooks 1987] of the modification is reduced.

The difficulty of changing the knowledge base is too great, from the knowledge acquisition point of view, when the designers will not put forth the effort required to correct errors or add new information to the

system. The work I propose attempts to reduce the difficulty of adding knowledge to the system by allowing the initial addition of knowledge in representations that do not create accidental complexity.

Design as an evolutionary process.

The 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 1981]. Their understanding of the design issues in a task gradually evolves along with their design [Suchman 1987]. 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.

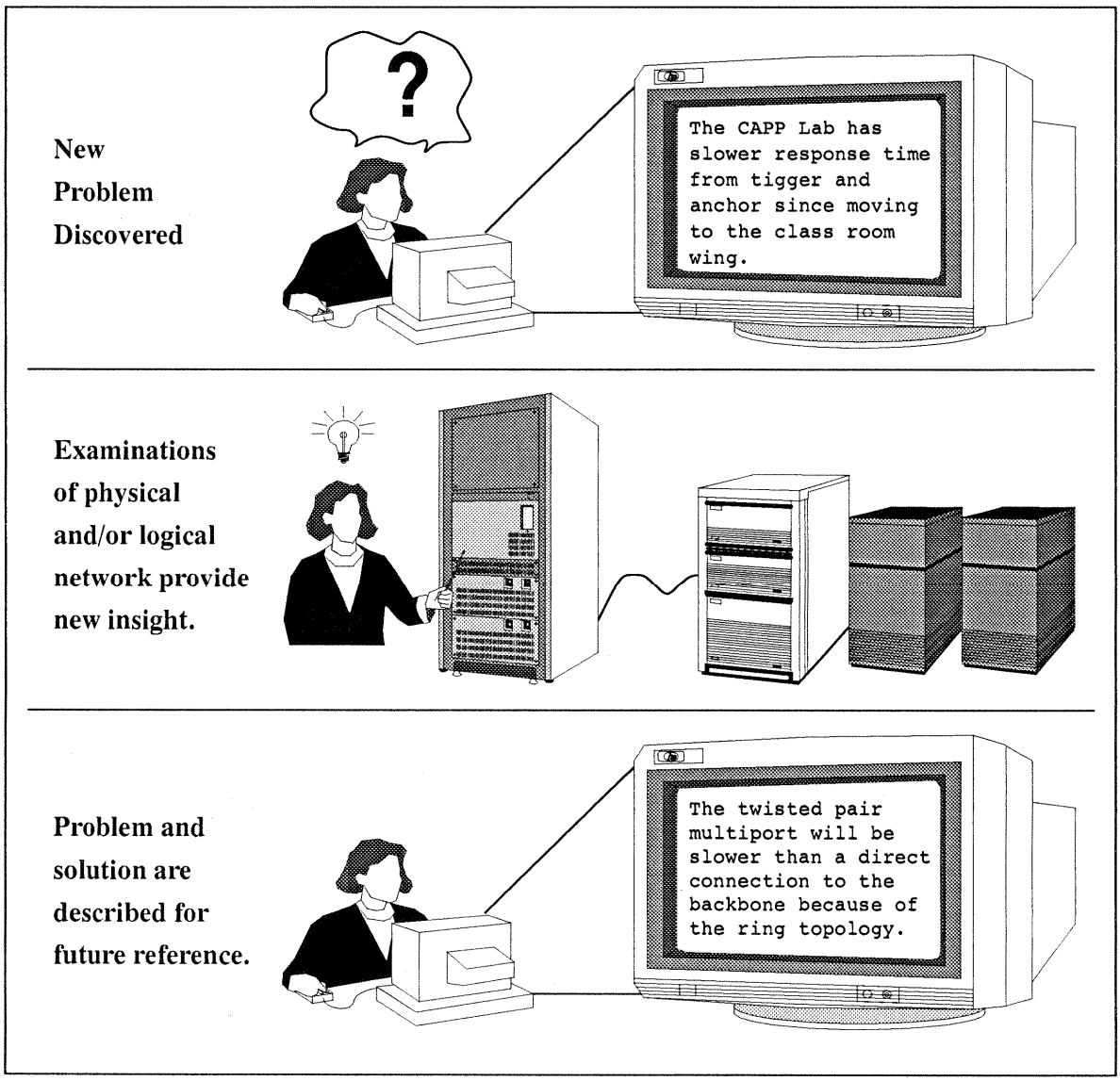


FIGURE 2. When problems are detected, in either a simulated or real network, the symptoms may not provide enough information to know what the problem really is. Over time the designer/administrator will gain more information on the problem and be able to come up with solutions.

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 1986]. 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 1983]. As time passes, the "story" the designer will tell about why the decision was made will change to reflect the designer's more recent experiences [Schank 1990].

Designers will not be able to immediately enter new knowledge in its final form into the design environment unless the issue involved is very simple. Because of (1) the limited understanding the designer may have of the new issue, 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 appropriate to provide for use since designers will have much experience with them and since they allow for much information to remain implicit.

As the designer encounters (and hopefully solves) unforeseen problems in the design task the designer will come to understand more about the domain and the relationships between knowledge within the design environment. As this occurs designers must be able to modify the information previously entered in all of the representations provided to reflect the change in their knowledge. Support for the incremental modification of knowledge, not just within a single representation, but also across representations, is needed to support this knowledge acquisition process.

Evolution from informal to formal representations.

As already noted, work already has been done focussing on the evolution of knowledge within a single knowledge representation. Evolution across representations will most likely occur from informal to formal representations, as the designers gain understanding of their task and the problems they encounter. The advantage for the designers in converting knowledge from informal to formal representations is that the system will be able to provide better support services to the designer. The important threshold is where the designer sees the effort of adding new information or formality to the knowledge base as being outweighed by the benefit provided by the system [Grudin 1988]. By allowing the gradual addition of new knowledge and formality to knowledge already in the system, and by providing tools to aid the designer in this process, the system will support knowledge-base modification through a number of small-effort steps.

One difficulty in this evolution is that the transfer of knowledge from informal to formal representations is likely to be not purely syntactic. This is because informal representations allow more information to remain implicit, and converting this knowledge to formal representations will likely require some of this implicit information to be reevaluated while being made explicit. As the designer is forced to be more explicit, the effort of the designer required to modify knowledge base is likely to increase. This is similar to the difference between explaining conceptually how an algorithm works and actually writing the code which implements the algorithm. In this case the explanation allows many details to remain unstated, while the code requires the algorithm to be specified exactly.

The implicit nature of informal representations of information means that the informal representations should not be "tossed out" upon the information's evolution to more formal representations. The informal representations may be preferred by other designers and will contain information whose effort to formalize was not considered worthwhile. An example is that each use of an informal representation like text can imply much about its author. Formal representations of information will often not contain such implicit

information. Informal representations of information need to be kept so that a designer can better understand and evaluate a piece of formally represented information.

Figure 2 shows a designer discovering a problem in a simulated or real network. Figure 3 could describe the actual addition of information to the design environment by the designer. Initially, labelled Nov. 15 in Figure 3, the designer encounters the problem but does not immediately understand what is causing it. At this time the designer might enter a description of the symptoms and create an issue in the design rationale structure concerning the problem. The designer will gain understanding of the problem through examination of the real or simulated network and by locating relevant information in the knowledge base or other information resources. After working with the network over a period of time the designer gains new insight into the cause of the problem. Such insight can be added to the design environment by adding textual descriptions of new information and ideas as well as modifying the PHI structure to better represent the new understanding of the problem. Eventually the problem and solution are found. At this point the designer adds some final text describing the test confirming the cause of the problem, adds notation to the PHI structure marking a solution as having been chosen, and the creation of a critic rule and object attributes to monitor the system for the occurrence of this problem.

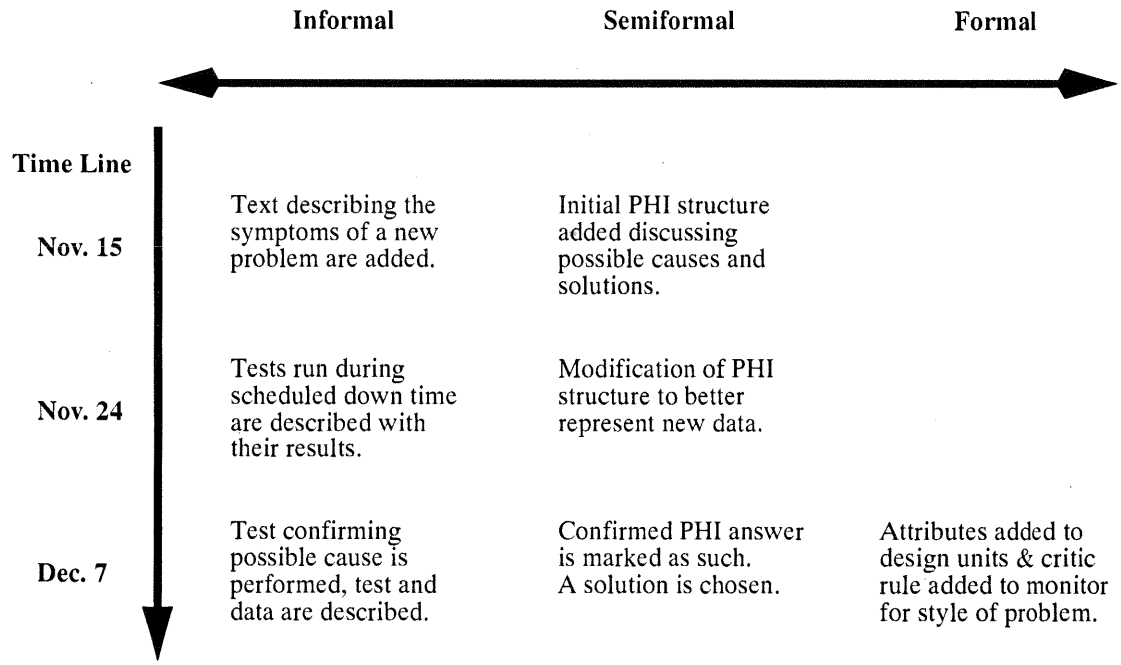


FIGURE 3. As a designer's understanding of a new problem changes the different representations will be used to store this knowledge. This time line shows the postulated inclusion of knowledge about a new problem. In this case the designer changes the knowledge in the system three times. The different representations are used for different types of knowledge which are available at different times during the solution process.

The discussion to here has focussed on a single designer interacting with the system. With multiple designers using the system the evolution can be expected to occur at a faster rate than with a single designer. A designer who has more experience in a particular aspect of the domain (like the use of AppleTalk networks) may understand how a new problem, encountered by another designer, is related to information already in more formal representations. This not only allows the dispersal of design issues to

be handled by the people who can answer them but also provides for discussion between designers to occur in different formats.

Perceived costs are especially important in the acceptance of groupware systems [Grudin 1988; Markus, Connoly 1990; Grudin 1992]. The goal of lowering the costs associated with adding knowledge to a knowledge-based system would increase the possibility of acceptance of the knowledge-based system. This work will not address problems associated with difficulties that arise from social interactions such as the incorrect assumptions about conversational patterns built into Cognoter and discussed in [Tatar et al. 1991].

Problems associated with multiple designers modifying the information space in a single design environment include access and modification rights, and notification of changes. While access and modification rights are difficult problems, I believe that techniques used or being explored in existing hypermedia [Berlin, O'Day 1990; Hahn et al. 1991] and groupware systems [Ellis et al. 1991] are sufficient for my task. The necessity for a notification mechanism to provide designers with an overview of changes is difficult and there should be domain-oriented, as well as general purpose, mechanisms for supporting this in the design environment, but this is not part of my planned dissertation research.

Related Work:

As described, this topic intersects with a number of different research communities. The work which is most closely related to my research falls into three main categories: knowledge acquisition, hypermedia, and design rationale.

Knowledge acquisition.

Within the knowledge acquisition community, my work is related to work on automated knowledge acquisition. This area focuses on tools for knowledge engineers and possibly end-users, but is generally limited to supporting only formal knowledge representations. One of the older and most well known systems in this area, Teirasias [Davis 1984], can be thought of as an expert system on knowledge-base design. More recent systems such as MOLE [Eshelman et al. 1987], OPAL [Musen 1989], and the HITS Knowledge Editor [Terveen et al. 1991] improve on this approach through the use of a presupposed problem solving method, an explicit domain model, and cooperative problem solving respectively.

Creating domain-oriented knowledge acquisition tools, such as OPAL, is time-consuming and difficult. Meta-level tools, such as PROTEGE [Musen 1989] and DOTS [Eriksson 1991], 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 1991], created are then meant to be usable by the domain experts.

An important difference between this work and mine is that these knowledge acquisition systems are limited to use with formal knowledge representations and do not provide support for the transfer of knowledge between representations. Another difference is that my work focuses on supporting designers that are in the process of design, rather than as a separate task that is performed in isolation.

Hypermedia.

Hypermedia systems use semiformal representations and multiple medias (text, graphics, audio, video) to support the authoring and browsing of information for many purposes [Conklin 1987]. One of Frank Halasz's "Seven issues for the Next Generation of Hypermedia Systems" [Halasz 1988; Halasz 1991] is how to integrate computation into hypermedia systems and how to perform computation over hypermedia. Since Halasz's initial call for the investigation of this issue, many people have examined the connections between knowledge representation and hypermedia [Russel 1990; Schwabe et al. 1990; Kaindl, Snaprud 1991]. Systems like CONCORDE [Hofmann et al. 1990], SPRINT [Carlson, Ram 1990], and RelType [Barman 1991] allow for formal knowledge representation in addition to the informal and semiformal information. The knowledge representation scheme I plan to use has similarities to the representation used by the Virtual Notebook System [Shipman et al. 1989], which I have worked on previously.

One system with similar goals to my work is Aquanet. Aquanet is a generic hypermedia system which focuses on "knowledge structuring tasks" [Marshall et al. 1991]. Aquanet supports the creation of structure "schemas" in a schema editor. These schemas are then used as building blocks in structuring the knowledge in the system. In Aquanet there is no domain orientation other than that provided by the schemas created by the user.

One difference between my proposed work and the work done on these systems is my emphasis on domain-oriented tools to aid the user in placing new knowledge within the existing knowledge base and in the transfer of knowledge between representations. This emphasis on supporting authoring will have to be of more concern in future hypermedia systems since the information location problem becomes worse (and it already is a difficult problem for large applications) in poorly organized hyperdocuments. Another difference between my proposed work and the above systems is that these systems are not integrated with the system supporting design construction.

Design rationale.

Design rationale systems are specialized hypertext systems that are primarily concerned with supporting the capture and use of design rationale. This support ranges from systems which provide an interface through which the user can create/edit design rationale to systems which actively elicit design rationale by asking questions about decisions the user has made. Design rationale is normally captured in a semiformal representation with a possibly extensible set of node and link types [Jarczyk et al. 1992]. There is no attempt by these systems to gain any understanding of the informal knowledge captured within nodes.

Design rationale systems vary in the level of formality that they support with SIBYL [Lee 1990] being the system which provides the most support for formal representations while systems/models like gIBIS [Conklin, Begeman 1988], DR [MacLean et al. 1989], and AAA [Schuler, Smith 1990] have favored the simpler and easier to use representations. PHIDIAS [McCall et al. 1990] provides typed links between textual or CAD-object nodes and a query language which can then process over the links (structure-based queries) as well as the graphical content of the nodes (content-based queries).

When design rationale systems support more formal representations they do not provide any of the support of the knowledge acquisition tools discussed above in helping the user take advantage of this functionality. Other differences from my work are that these systems provide no aid to the user for transforming rationale already entered from one form to another, and that most are stand-alone systems separated from the systems supporting design construction.

The Implementation of XNetwork:

In this section I describe how my system implementation will support the evolution of a knowledge base. First I describe XNetwork, the X Windows-based design environment for computer network design, as it currently exists. Then I will discuss the basic components of the system that still need to be implemented and how the Evolving Formality Tools (EFT) will be integrated into the existing system.

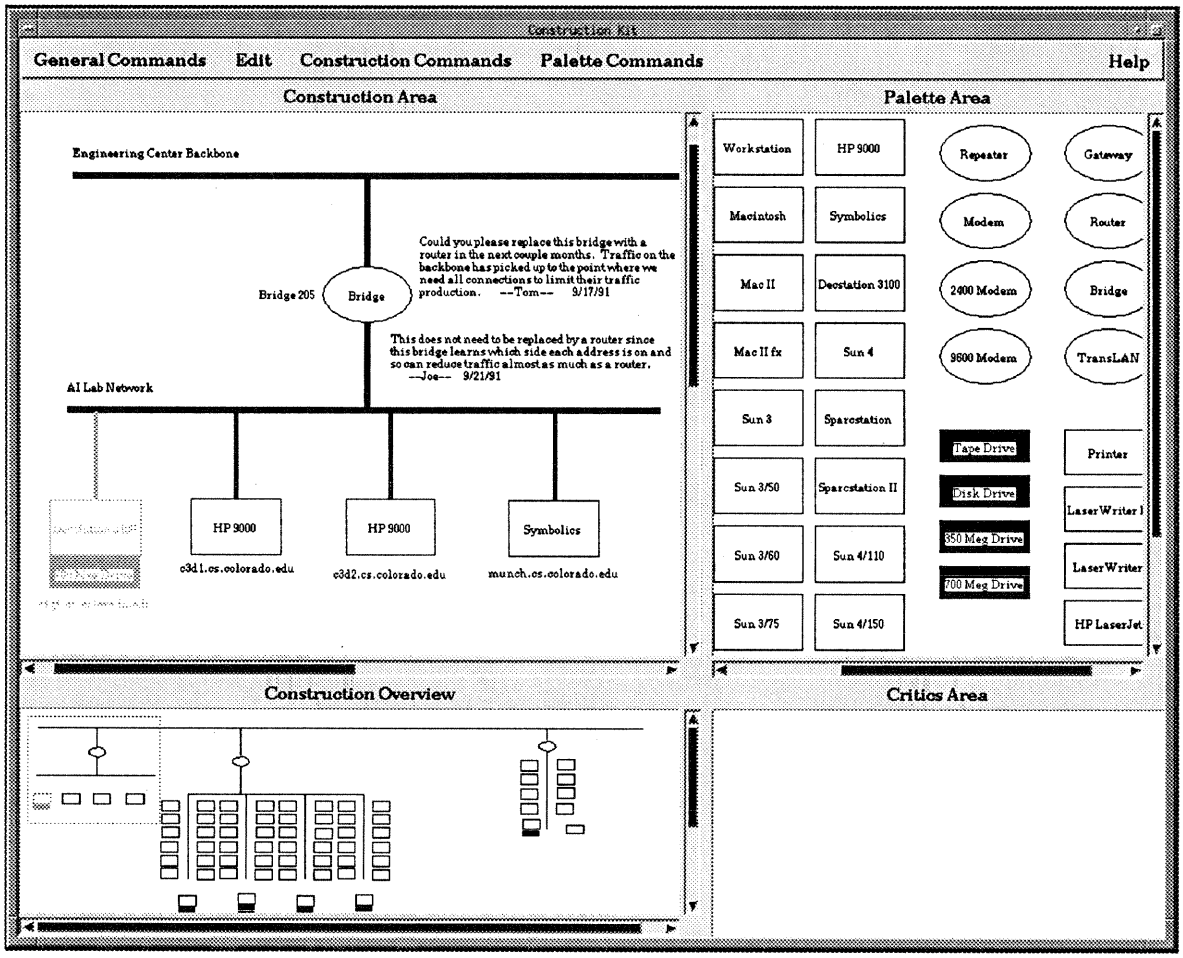


FIGURE 4. XNetwork provides a work area, an overview area, a palette, and a (so far non-functional) critics area. The work area shows discussion around a bridge that connects the engineering center backbone cable to the AI lab network cable and the machines in the AI lab. All objects in all aspects of XNetwork are treated the same, providing flexibility in creating tools to integrate the various representations.

Figure 4 shows the designer's workspace in XNetwork. The top right part of the window contains a palette of design units which can be used in the design. By choosing a palette item a copy of that item appears in the work space to the right of the palette. This work space is the area where the design units are arranged and discussed. Below the work space is the construction overview, which provides a wider angle view of the construction space to provide the designer with a feel for where they are in relation to the rest of the design. The bottom right hand corner contains a critics area, which the system will use to notify the designer of any messages that are provided by critics, which currently have not been implemented in the system.

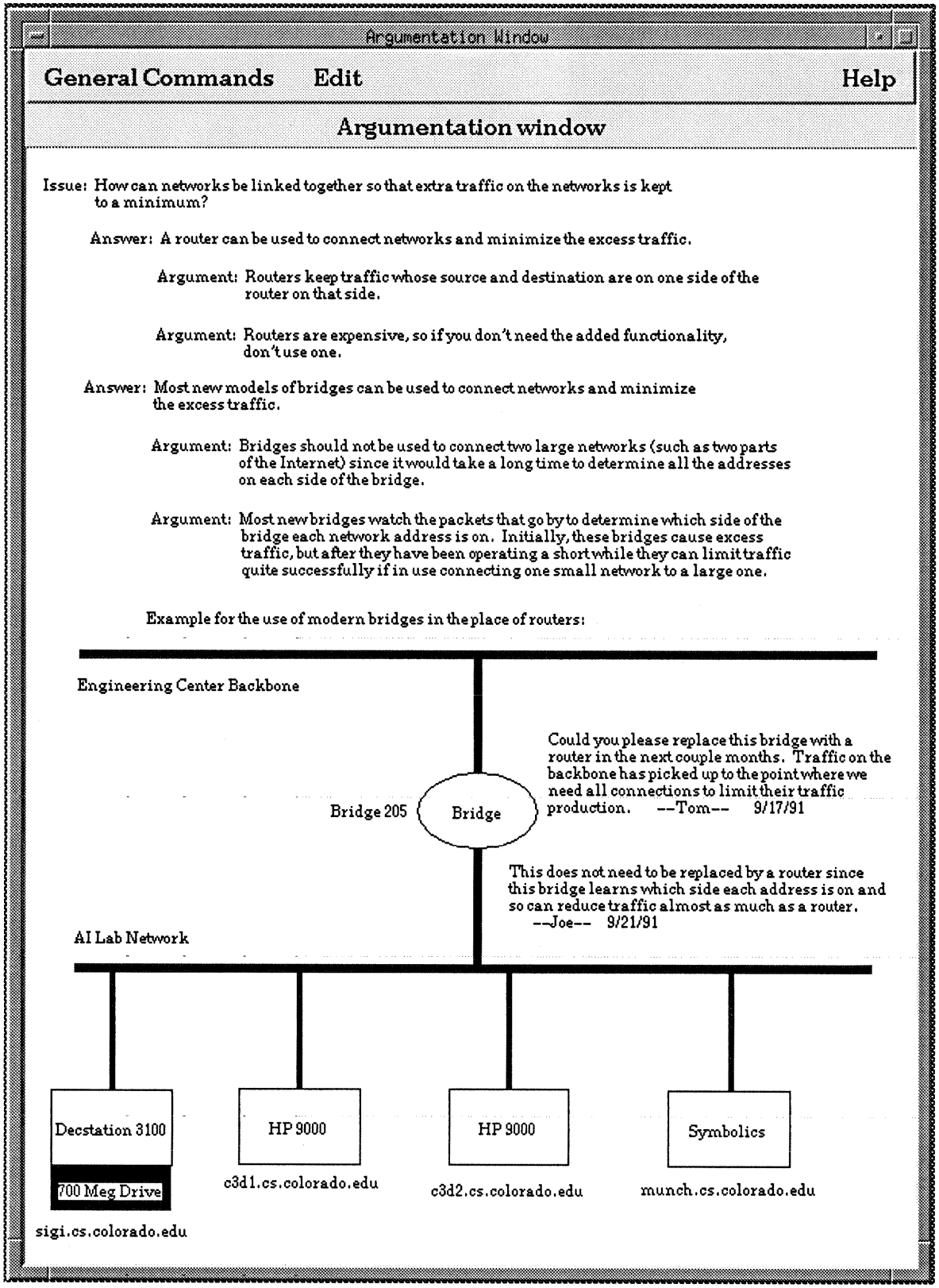


FIGURE 5. **Argumentation pages in XNetwork can be used to collect design rationale along with pieces of actual designs. This page shows a part of the design seen in Figure 4 being used as an example of an answer to the issue being discussed in the argumentation.**

Figure 5 shows an argumentation page in XNetwork. User interaction in the argumentation pages operate similarly to the workspace area of the main window. Argumentation pages can contain all types of objects and no particular support for the PHI style of argumentation has yet been provided in the system. Figure 5 shows a piece of the design shown in Figure 4 being used to illustrate the use of bridges in connecting networks without overly increasing traffic on the networks, which is the topic of the discussion on this page. This page shows a combination of informal text notes, PHI design rationale, and formally represented and connected design units. In the following discussion I will focus on the differences between XNetwork and JANUS [Fischer et al. 1989], a kitchen design environment, and why these differences are important.

Current state of the system.

My implementation has integrated the different representations in the design environment (text, semiformal design rationale, and formal knowledge including objects with attached properties) into a single knowledge base. One purpose of this is to simplify the connections between the components in design environment. In describing the Human Interface Tool Suite, [Hollan et al. 1991, p. 294] claim that:

> "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."

Since the components in XNetwork access the same knowledge base, rather than each component having its own storage means for knowledge, as it is in JANUS, changes made in one component are immediately accessible by the other components. This practical consideration for implementation becomes critical for my tools (EFT) which need to be able to read and modify all of the various representations and possibly use knowledge-base consistency as one means for providing suggestions. Figure 6 diagrams the use of the integrated representation as an underlying layer in the design environment architecture.

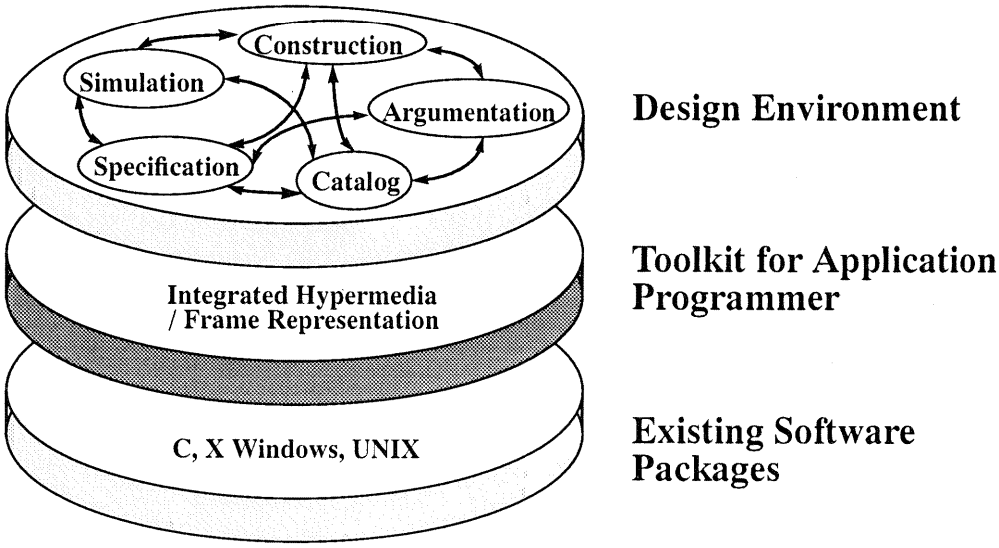


FIGURE 6. The work up to now on XNetwork has focussed on providing a toolkit for design environment developers which allows easier creation and integration of components in the design environment.

To integrate the various types of knowledge in the design environment, all knowledge is kept in a single database of objects similar to Minsky's frames [Minsky 1975]. Each object in XNetwork has a display and attribute/value pairs. The display method is a set of graphics primitives, like draw a circle of a certain size, and can be created and edited by users in a graphical editor similar to MacDraw. (JANUS requires users to use a lisp-like language to modify object displays.)

Attributes and values of objects can be created and edited in a property sheet, as seen in Figure 7. The user can choose between various types for attributes, which will then be enforced. Attributes of type *link* have other objects as values. Knowledge about relationships between objects is added by creating new links. Design units are represented by objects with their technical specifications encoded as attributes and values. Textual notes are represented as objects with a textual display component. Links between objects allow for combinations of formal and informal knowledge in the system.

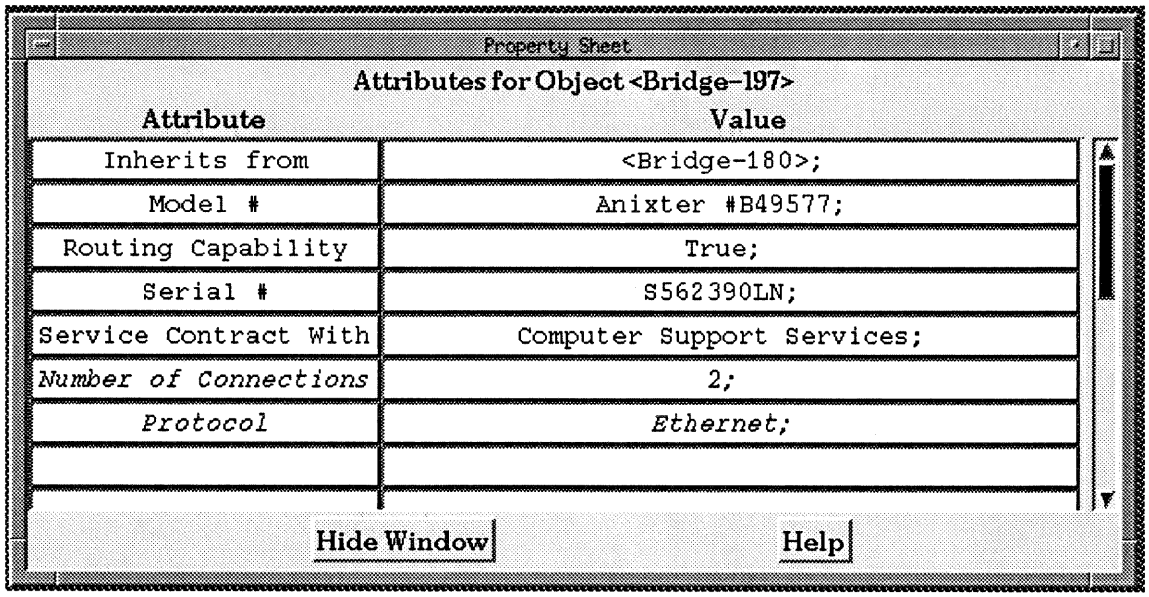


FIGURE 7. Each object in XNetwork, whether a text note or a palette item, has attributes and values which can be edited in a property sheet. Italics indicates that the attribute is inherited.

Objects can move between components and can be displayed in multiple components at once. Objects can be grouped into compound objects, which have their own attributes and values, much in the way one groups pieces of a drawing in MacDraw. Compound objects allow for subassemblies, such as a subnetwork or a combination of a workstation with its peripherals and drop cable. Compound objects can also be used to blur the distinction between the catalog 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 specific to this example. Figure 5 shows such an example in the context of an argumentation page. Such a catalog example can be represented by a compound object which contains a configuration of design objects, comments, formal knowledge about the configuration, and links to related argumentation. The easy creation and use of compound objects is critical to supporting the construction of large design artifacts efficiently and to allowing discussion to focus on a group of design units as an intermediate abstraction.

XNetwork includes the inheritance of attributes between objects. The top attribute in Figure 7 shows that the object <Bridge-197> inherits attributes from <Bridge-180>. 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; there is no class/instance distinction. Removing this distinction allows the network designers to use the inheritance mechanism without having to learn about these knowledge engineering concepts. The creation of objects that act as classes can still be done with this mechanism, although the system will make no distinction in how such objects can be otherwise used. When adding the initial domain knowledge "seed", that the designers will then modify, the developers of the design environment will use knowledge-representation techniques (such as concept hierarchies) most often associated with class/instance inheritance mechanisms.

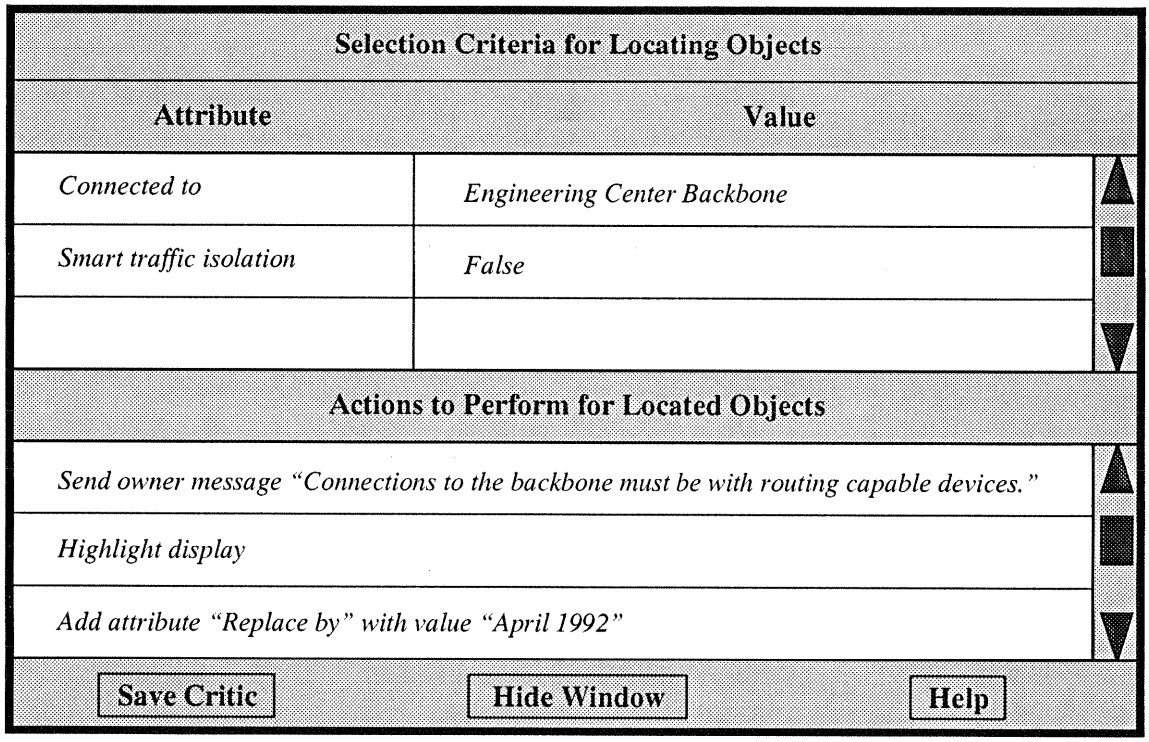


FIGURE 8. A diagram of what a critic rule editing interface might look like. The top part of the interface is where the designer would define the applicability condition and the lower part is where the designer specifies the actions to take when the condition is met.

Additions to basic system.

Still missing from the basic XNetwork implementation are a query mechanism for the retrieval of knowledge from the knowledge base and a simple action language to operate on the knowledge base. The query mechanism will primarily be for internal use within the system. Interaction between the users and the query mechanism will be through intermediate interfaces, like a variation on property sheets. By filling out a form which includes a property sheet the user will be able to retrieve all objects whose properties match those in the property sheet. This form will take advantage of the user's knowledge of how to use the basic property sheets, while adding controls for greater expressiveness as deemed necessary.

An action language will be added so that users can perform operations on the knowledge base, such as displaying knowledge to the user and modification of knowledge. The action language, like the query

mechanism, is meant to be an intermediate layer in the system which would often be hidden from the user. The action language will include domain specific operations.

A generic critic mechanism, which is important for linking the components of the design environment, will consist of a query (using the query mechanism) and an action (using the action language). A critic's query and action will be stored and constantly active unless the designer has asked for the critic to be turned off. Here is an example of what such a critic might be:

Query: Retrieve all objects where the attribute 'Connected to' includes the value 'Engineering center backbone' and where the attribute 'Smart traffic isolation' has the value 'False'.
Action: Provide the user with a list of the objects retrieved.

Besides providing a list of these objects, the action of the critic could have included the addition of a new attribute, such as 'Replace by' with a value of 'April 1992'. This mechanism would allow critics to include a combination of actions including notifications to the user, as well as automatic modifications to the knowledge base. Figure 8 shows a critic similar to the one just described. This more general notion of critics, where they may perform knowledge-base modifications automatically, might better described as 'agents'. By providing the user with the ability to create and edit these critics the user has control over which tasks the system performs automatically, and which tasks the system leaves to the user.

Evolving Formality Tools (EFT).

So far the discussion focussed on additions to the system which are to be used as the building blocks for tools to support the primary goal of knowledge-base evolution. These tools will include process-support tools, which will aim to provide the designer with the information they require to make changes to the knowledge base. This section describes a number of tools to give an idea of the range of tools that might be of use in supporting the formalization of knowledge. I plan on implementing some subset of the tools described in this section, plus tools similar to these whose usefulness appears later in the research.

Concept Browser. A concept browser provides some view of the domain concepts and relationships among them. One interface a concept browser might have is that of an outline processor, where opening up a concept shows more specialized concepts in such a system. This style of interface has been used to provide access to the Medical Subject Headings (MeSH) and as an alternative browsing mechanism to the Virtual Notebook System (VNS) [Burger et al. 1991]. Graphical browsers could also be of use, although they require more screen space and can become cluttered quickly for large networks of nodes and links being graphed.

The concept browser, as envisioned, would also allow modifications to the structure of the concept hierarchies. In MeSH the terminology is updated periodically (at least annually) by the National Institute of Health. For the network design environment there needs to be methods for designers to modify the hierarchies of concepts.

External Information Includer. An external information includer would take information from other computerized information resources and help include them in the system. One example would be taking an electronic mail message, striping off the header, and creating a text note in the design environment containing the body of the email. This includer could also parse the header and would include this information as attributes/values of the text-note object. I believe that electronic mail is the primary candidate since it is network designer's main mode of communication, other than face-to-face.

Versions of this tool could also be built for incorporating articles posted to the network newsgroups, and reports from specific programs or databases. SA-Tool, a proposed tool to aid in network administration [Nemeth 1991], suggests the need for information from the Simple Network Management Protocol [Case et al. 1990] in order to provide active information on the network. As I develop an understanding of which information sources are used frequently and contain information that it is critical to capture, I will create tools to integrate information from these information resources. Work at Baylor College of Medicine on an Integrated Academic Information Management System (IAIMS) [Gorry et al. 1988; Shipman et al. 1989] has created tools such as these to provide medical researchers access to a variety of information through a seamless environment.

Knowledge Elicitor. A knowledge elicitor would query a designer for information by leading a dialog with the designer that is based on a decision tree of prepared alternatives. The purpose of this tool is to elicit knowledge to be included in the knowledge base. This approach was used by Ray McCall in his dissertation work to elicit argumentation [McCall 1979]. By posing questions like "What is the problem with this design?" and "How could this problem be detected in the future?" specific types of knowledge can be elicited about which relationships can be inferred.

Several knowledge acquisition tools for knowledge-based systems have used interviewing techniques to elicit knowledge. KNACK [Klinker et al. 1987] uses an initial interrogation session to elicit concept descriptions and vocabulary for a domain, ASKE [Patel 1989] leads a dialog with the domain expert to build an initial model of the task, and SALT [Marcus, McDermott 1989] uses this method to acquire procedural control knowledge. The Knowledge Elicitor would try to get less formal information from the designers than these tools, with the assumption that designers would be more willing to use the tool if it does not ask them to formalize the information they are adding to the system.

This interrogation method shows promise for getting less formally represented knowledge into the system in a semi-structured format. The knowledge elicitor can automatically create relations between pieces of text through assumptions made about the relations between answers to different questions. Active elicitation schemes have the possibility of annoying the designer, and so must be able to be turned off or might only become active due to requests by the designer.

Suggestion tools. There is a variety of information that the tool will have available to aid in making suggestions. The more formal domain knowledge, along with the placement, textual content, and textual attribute values can be used by the tool. For example one simple tool could look for vocabulary in textual values of attributes which might relate to other objects and suggest replacement (or augmentation) by a object relationship between the two objects. An example of this would be a workstation in the design (c3d2) that has an attribute "disk server" with the value "c3d1" as a text string. This tool would suggest the recasting of this attribute to be a relation, instead of a string, with a value of the object in the design that represents the device c3d1. Such a transformation may be trivial but it gives the system information needed to provide services, such as simulation.

XNetwork Connector. More ambitious tools could make use of partial understanding of informally represented knowledge to locate information within the knowledge base which is likely to be related. The XNetwork Connector suggests relationships to knowledge already in the system. The following text, which is taken from an electronic mail message between network designers, will be used to discuss this tool.

"dec will not be putting thick on the loaner decstations. i ordered one thick card for the lab. on machines you borrow temporarily from lloyd you will have to make do. the ugrad lab is thinnet. one thick card will igve us the gateway we need."

This text (including typos and lack of capitalization) is directly from the original email message. Assuming such communication would sometimes occur in the design environment as text annotations [Reeves 1991], text like this will be available.

There are a number of domain concepts in this piece of text: dec, thick (thicknet), loaner, decstations, card lab, machines, lloyd, ugrad lab, thinnet, gateway.

Assuming that some of these concepts like thicknet, decstation, card and ugrad lab are already in the knowledge base, the tool could retrieve objects within the system that relate these concepts. This type of suggestion could be done with a key-word matching algorithm. Beyond what could be considered key-word matching, the processing of the text should be able to suggest that thicknet, decstations, and the ugrad lab is the topic of this text because of these concepts' recurrence and positions in the sentences. Using this information the tool can locate objects related to thicknet and decstations in relation to the ugrad lab. Issues in the issue base and other text objects which discuss these topics would then be suggested as related to the piece of text.

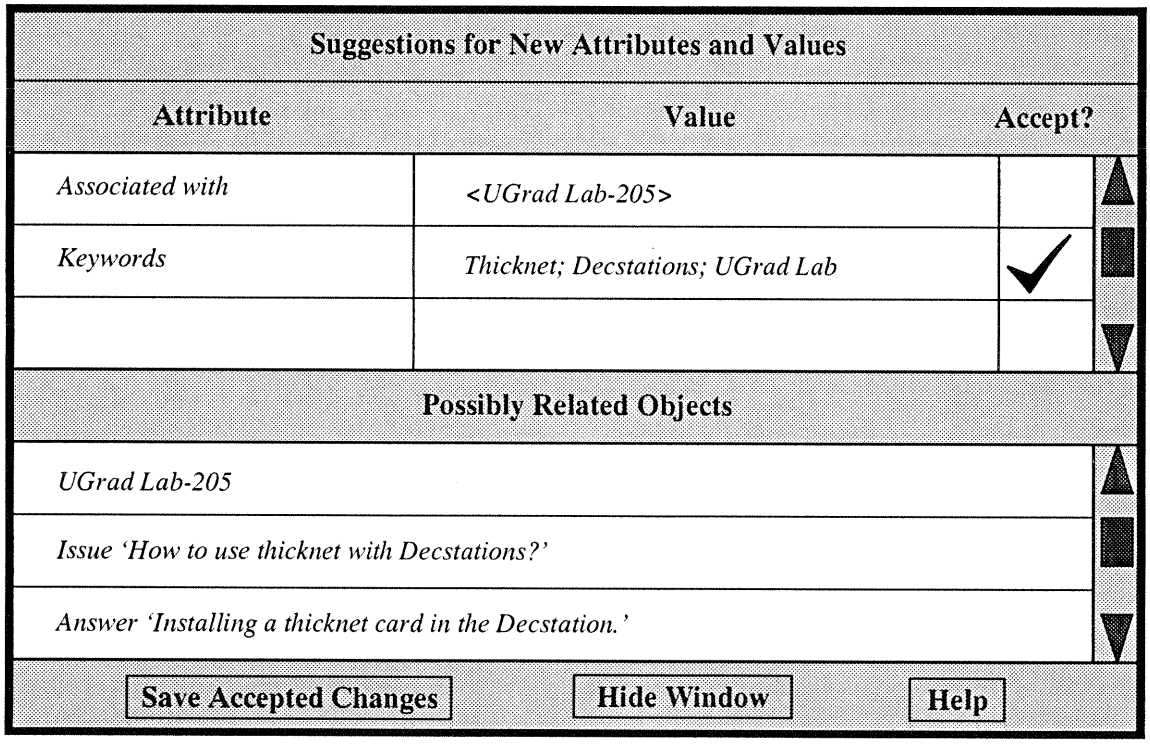


FIGURE 9. A diagram of what the XNetwork Connector might look like. Suggestions for new attributes and values for an object are in the top part of display while a list of possibly related objects appears in the lower section. Designers can accept attributes that they agree should be added and view objects that might contain related information.

Figure 9 shows what this tool might look like. This tool provides a list of suggested attributes and values that categorize the text and create relations to other objects, which provide information on the object when chosen by the user. The user can modify the suggestions made by the tool, accept the suggestions, or ignore the suggestions.

By analyzing network-domain text and interviewing network designers I hope to determine the most frequently occurring domain concepts. Sources for network-domain text include email to the trouble alias, email between the network administrators, and text in books on network design and administration. One problem with key-word and similar approaches is how to recognize synonyms, different uses of the same word, and misspellings (such as the typo 'igve' for 'give'). If significant problems in the acceptability of suggestions result from these types of problems, I can use techniques such as described in the Knowledge Elicitor section to ask the user for keywords, instead of having the system infer them.

XNetwork Formalizer. The XNetwork Formalizer will use domain-oriented techniques similar to those described for the XNetwork Connector to provide suggestions. While the XNetwork Connector suggests associative connections between pieces of knowledge in the system, the XNetwork Formalizer will suggest possible reformations of the knowledge in other representations. This would include suggestions about converting textual annotations to PHI-style argumentation as well as text in annotations or argumentation to attributes, values, and critic rules. This tool will look much like the XNetwork Connector shown in Figure 9.

To continue with the example of the piece of the email message between network designers, the XNetwork Formalizer could retrieve the ugrad lab description and suggest that attributes related to cabling or workstations might be modified. Now let us assume that the text processing could determine more about the content of the text, such as the fact that the ugrad lab is using thinnet, from the text "the ugrad lab is thinnet." The tool might then suggest that a new attribute of the ugrad lab description be created, if it does not already exist, which describes the type of cable being used. This particular knowledge can be very crucial since thinnet has very different characteristics than thicknet, which is used in most of the department's network.

Also in this text is the knowledge that one thicknet card has been ordered and will be used as a gateway for the subnetwork under discussion. For the system to suggest this requires more global level knowledge of the network, such as the backbone cable is thicknet. The system will certainly not be able to suggest formalizations for all the knowledge in a piece of text, and as was argued before, the text will still be of use. This piece of text contains the notion that some of the work being discussed is a temporary setup, which will have to be redone when the new machines arrive. In this case the designers may not want to make the effort to encode much of the content since it may be out of date in a couple weeks.

Summary. There are many types of tools to support the evolution of knowledge from informal representations to more formal representations. Some of the tools are designed to support the process by providing interfaces appropriate for browsing information in the design environment and initially getting information into the environment. Other tools use the system's knowledge of the domain to make suggestions about possible formalizations that could be added to the knowledge base.

An example using text from an actual email message between network designers was used to show some benefits from a simple mechanism for processing informally represented information that was very close to keyword matching. As the ability to interpret the informal knowledge increases, the suggestions become

more detailed and I hope to investigate if they become more useful. Even the most ambitious tools described does not rely on fully understanding a piece of informal information, but instead gains most of its effectiveness by using the topic(s) of a piece of informal information. Suggestions made by the tools will not be completely accurate and the assumption/approach is that they provide a basis for reformulation by the human designer, such as with the suggestion illustrated in Figure 9. The usefulness of automatic translation programs as preprocessors for professional translators provides support for this approach.

While the example shows text as the informal representation, processing which could provide similar understanding of other informal representations would act similarly. I expect that I will use knowledge of the drawing layout, such as distance between objects in the work area, as another source of informal knowledge that will be used by my tools. An example is that the text annotations in Figure 4 are near network cables, some drop cables, the bridge, and some workstations. Based on the knowledge that both pieces of text discuss bridges, routers, and traffic, and the knowledge of the text's position in the layout, a better suggestion can be made than with either piece of knowledge alone.

The above discussion of the tools have included small examples. To better communicate how I view the tools being used I have included the two following scenarios. The first scenario describes the addition of a new connection to the network, while the second scenario describes the solution of a problem with the network. The section of the first scenario describing the use of the XNetwork Formalizer has been expanded in Appendix 1 to include more description on the types of knowledge and mechanisms required by the system.

A Network Design Scenario:

One of the goals of XNetwork is to support collaboration and communication between designers. The following scenario shows the above tools aiding in formalizing information contained in a discussion between network designers. This scenario is based on actual email messages between three network designers involved in adding a connection to a departmental network. The network designers were not using XNetwork and actions taken by the designers in the scenario are only postulated. The names in the messages have been changed and one grammatical mistake was corrected.

Adding new request. The system designers for a department get a request to connect a large instructional room which is not under control of the department to the departmental network. The email request states:

> "please make sure there is an item in your queue to get a net drop to cr2-whatever... the room with the very large screen projection system in the north end of the 2nd floor of CR."

The email request is received by Sal, who brings the message into the design environment through an Information Includer, which parses the mail header to automatically create attributes such as sender, subject, time sent, and time of arrival. At this time no other understanding or representation of the message is attempted.

Integrating request. Sal also happens to be the designer who normally handles such requests. Sal selects the request and asks for the XNetwork Connector to suggest relations to knowledge already in the knowledge base. The Connector recognizes the concepts 'drop cable', '2nd floor', and 'CR wing' and notifies Sal that there is no information about the combination of these three concepts. Sal then specifies to list objects which match two of the three concepts. Looking at what the Connector did locate, Sal finds that there is information on network connections in the CR wing, but no connections exist on the second floor.

Consulting another designer. Sal decides that a senior designer must be asked to verify that the request can be fulfilled, and if so how. To do this Sal places the original request and a new note near the CR part of the network diagram. Sal also specifies for XNetwork to notify the senior network designers of the new note's existence. In particular the note to the other designers says:

> "we allowed to do this? i don't think we own anything on the 2nd floor. AND how are we supposed to get a drop cable up to the second floor in this area? there is the elevator shaft but i don't think we can make it (lenght wise). it will also require a dedicated tap. So could one of you higher-ups please respond to my concerns over this."

Gaining permission. A senior designer, Pat, reads the note from Sal and uses the design environment to get information on who controls the access to places where the new cable is likely to be placed in this part of the building. The design environment, having a complete floor plan of the building with some information about what rooms are used for, shows that the math department and media services use several rooms in the area. Pat sends email to representatives to each of these groups to make sure that the addition of such a connection will be acceptable. A reply from media services okays the connection and details where the connection should be. After

approval has been granted, Pat edits the message from media services and adds it next to the message from Sal. Pat specifies for XNetwork to notify Sal of this new message:

> "we have permission from mary caldwell, boss at media services, to install the network connections we want in cr2-28 and have them terminate in the media closet. you can pick up a key to the closet by calling x8470 and going over to get it. you can keep the key for a couple of weeks after the installation to be sure everything is up and working properly."

Evaluating possible solutions. A third designer, Paul, is put in charge of actually physically adding the cable and devices to the network. Paul decides to look into different possibilities for getting the connection to the classroom. Paul selects the three text nodes about the project and asks for suggestions of related information from the XNetwork Connector.

Because of the large number of topics found, the Connector provides a list of topics that the system feels might be important topics to the discussion. These topics are: department network, drop cable, second floor, CR wing, CR cable, elevator shaft, and dedicated tap. Paul selects a subset of these topics as being of interest: drop cable, CR cable, and dedicated tap. Paul looks at information found by the Connector about previous dedicated tap connections to the CR cable, including examples in the current design, issues on how best to do add dedicated tap connections, and notes about particular installations of dedicated taps.

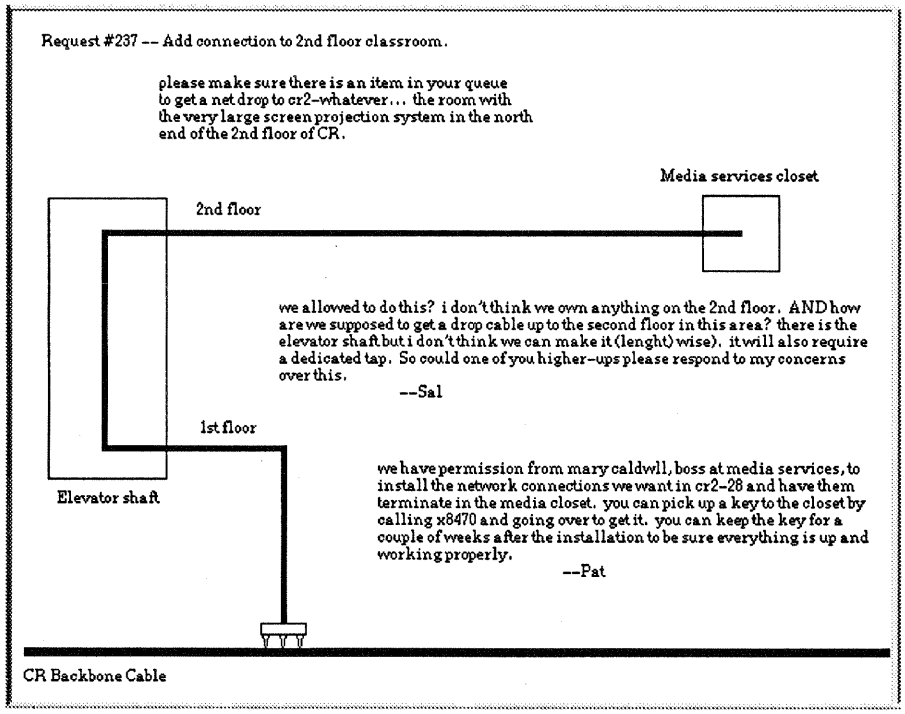


FIGURE 10. The current project is made into an example which can later be examined when similar tasks arise. This view shows the plan for adding the drop cable to the second floor and the discussion about the task.

Making modification to network layout. Paul checks the current state of the part of the CR backbone that would likely be tapped to locate a site that will work for adding the new connection.

Once chosen, Paul adds the connection to the design in the design environment to let the other designers know of the current plan for connecting the classroom. Paul creates a new view in XNetwork including all the modifications to the network made and the discussion about the project to create an example when similar projects come up. This new view is shown in Figure 10. Paul then asks the XNetwork Formalizer to make suggestions for formal descriptors of this view.

Formalizing description of view. The Formalizer suggests attributes/values based on the design configuration in the view and the discussion about the design configuration. Based on the text and layout shown in Figure 10 the Formalizer picks out values for common attributes that are needed in the domain. The Formalizer attempts to suggest values for attributes encoded by the design environment developer, who also provides algorithms making suggestions based on the current state of the knowledge base. The suggestions that the Formalizer provides to Paul are shown in Figure 11. In this case the Formalizer has suggested attributes to be attached to this project view which specify the physical devices used, the affected locations, people involved in the design, and the request number. Figure 11 also shows the suggested new issues to the argumentation space which are based on prefabricated issues which are important for the domain. The two issues shown in Figure 11 are based on the general domain issue "How to install a connection to <location>."

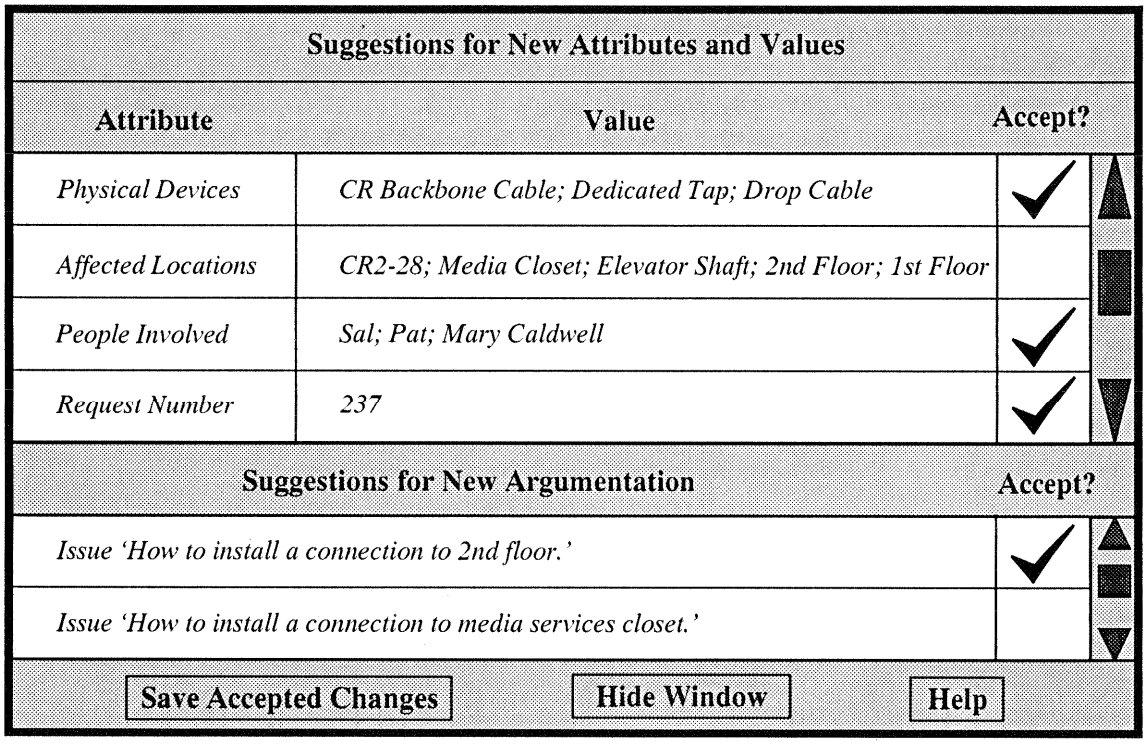


FIGURE 11. A diagram showing the XNetwork Formalizer after Paul asks for suggestions based on the knowledge about the project he has been working on. Paul has accepted three of the four suggested attribute/value pairs and one of the two suggested argumentation nodes shown.

Describing state of project. After the cable has been initially installed, Paul adds a message in the design environment describing the current state of the project. The status message is shown in the top right-hand side of Figure 12. The message says:

"one cable and one tap are in there. gary gave me a hand (thanks!) i made a mistake because i thought we had one spool of cable and we got another one hour before we started to work on it, so i have to go back and run the second one later."

Refining knowledge about project. Now that the installation task has been partially completed, Paul also adds more detailed information on this project of connecting the classroom. Changes to the design that were made during the physical process of laying the cable need to be added to the network layout in the design environment. Paul adds text and changes the project view to show that in order to get through a fire wall the drop cable had to be routed through the room CR 2-34.

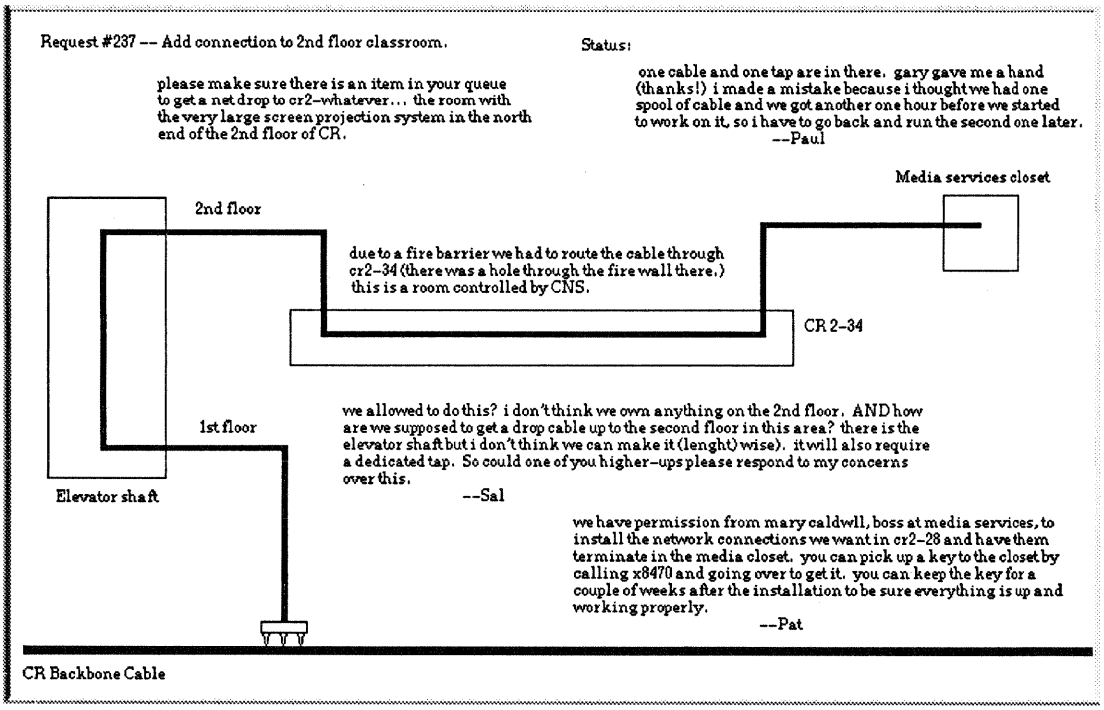


FIGURE 12. After the installation task has initially been accomplished the designer goes back and changes the planned design to include changes that were made. Now the project status has been added to the project view.

Paul asks the Formalizer for suggestions based on the new project view. The Formalizer suggests the addition of "CR2-34" to the value of the attribute "Affected Locations", and the addition of "Paul" and "Gary" to the value of the attribute "People Involved". Paul accepts these additions but rejects the Formalizer's suggestion to add a new issue "How to install a connection to CR2-34" to the argumentation.

This scenario describes the support of a few network designers working on a project. In this scenario a single designer has done most of the formalizing of knowledge in the discussion after the whole conversation has taken place. This is the case because my work is not specifically concerned with issues arising out of multiple users of the system. I expect XNetwork to include support for some style of directed messaging, such as providing a list of new messages specified to be to a particular designer to that designer, but this is not part of my proposed thesis work.

A Network Administration Scenario:

The following scenario shows the tools aiding in a normal network administration task, handling email complaints. The message used below is again an actual message that was received by network designers from a student (XX is a workstation named after a Mexican beer).

Request arrives. The system administrator at a university gets an email message from a student that there appears to be some problem in communication between the workstation that the student uses and the department's network. In particular the message says:

> "Looks like there is a break in the connection between xx (in the CAPP lab) and the concentrator in the machine room. XX refuses to reboot (unable to obtain internet address) and the concentrator shows no carrier on that line."

Adding request to system. Upon receiving this message the system administrator copies the message into the system as a text object. This can occur either by selecting the email message from another program with the mouse and pasting it into a text object, or by a specialized tool, such as the External Information Includer. The latter would have the advantage that it would parse and attach the contents of the mail header as attributes and values of the text object. At this point the text object does not have any relations to other information in the system

Connecting related information. The designer selects the text object and asks the XNetwork Connector for suggesting related information. XNetwork already has knowledge about the concepts XX, CAPP lab, and concentrator. The knowledge base includes the information that XX is a workstation in the CAPP lab, and that the CAPP lab uses a twisted-pair concentrator for its subnetwork. By using this knowledge the system can suggest links from the new text object into the argumentation and the network layout based on the connectivity between XX and the concentrator. Specifically, the system can suggest issues in the issue base, examples, and other textual annotations which discuss problems that might be the cause of a workstation loosing its connection to a twisted-pair concentrator. Beyond the use of the machine names and types the system should recognize some of the other key-words from the domain, like connection, reboot, internet address, no carrier, and line. Using these the Connector tool could further order or prune its suggestions to emphasize the information found which also discusses these concepts. The network designer decides that none of the suggested connections discuss exactly the problem reported but adds connections between the new request and a few general issues concerning twisted-pair connections in the issue base.

Solving the problem. After examining the network layout the administrator decides that the problem is most likely with the concentrator and less likely with a physical connection, such as a twisted pair wire. By viewing the attributes of the concentrator in a property sheet the system administrator decides that resetting the concentrator should fix the problem, and proceeds to try this fix. After resetting the concentrator the workstation XX can again reboot and the problem has been fixed.

Recording the solution. After fixing the problem, the network designer switches back to the design environment and starts to formalize the symptoms and solution to the problem. Now the designer uses the Knowledge Elicitor (KE) to create PHI-structured argumentation about the problem that

occurred and the solutions that were attempted. Upon invoking the Elicitor the designer specifies that the new argumentation will be about a new problem and solution. The Elicitor uses this information to pick the predefined script it will use to ask questions of the designer. The following dialog occurs:

KE:	Please describe the problem encountered.
Designer:	A workstation lost its connection to a twisted-pair concentrator.
KE:	What was the first solution tried?
Designer:	Resetting the concentrator.
KE:	Did this fix the problem? (yes, no, in part)
Designer:	Yes.
KE:	When will this solution fix the problem?
Designer:	When the concentrator is in an incorrect state.
KE:	When would this solution not work for this problem?
Designer:	If the problem had been with the wiring or with the workstation.
KE:	Were any other solutions tried? (yes, no)
Designer:	No.
KE:	Were any other solutions considered? (yes, no)
Designer:	No.
KE:	Thank you for the information. Please review the argumentation produced for mistakes.

Revising elicited argumentation. The Knowledge Elicitor then provides initial argumentation structure and text to the designer in a new argumentation page (see Figure 13). The designer then edits the argumentation that the Elicitor created to improve readability. Structural errors could also be fixed by the designer at this point, but in this scenario the argumentation created is small and the designer does not consider there to be any mistakes in the structure. The benefit of the Elicitor is that the designer does not have to state PHI relationships between pieces of text, but can react to a suggested structure provided by the Elicitor.

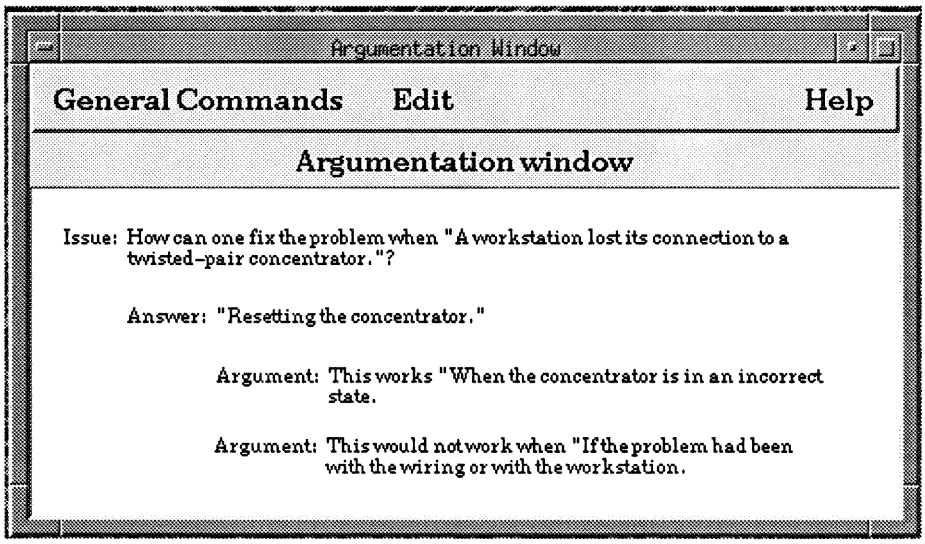


FIGURE 13. Initial argumentation provided by Knowledge Elicitor. The designer is then asked to read over the argumentation and improve it. The Elicitor uses scripts for commonly occurring types of knowledge, such as addressing new problems, or adding a new device to the network.

Integrating the new knowledge. The designer now brings up the XNetwork Connector again to add explicit connections between the knowledge just added to the system and knowledge already in the system. The designer selects the new argumentation and asks the Connector for suggested connections. At the top of the list of objects provided by the Connector are objects related to twisted-pair concentrators, since this is the domain concept most frequently referenced in the argumentation. The designer decides this is too broad of a topic and uses the Concept Browser to refine the concepts used by the Connector. In this case the designer adds the topic broken connection to the topic twisted-pair concentrator. The list of objects that the Connector now suggests are about connection problems with twisted-pair concentrators. The designer picks the original message from the student from the list of suggestions to have connections to the new argumentation page.

The problem described in this scenario is encountered not during a 'design' task, but during routine administration. Network problems often do not occur without specific conditions that are not controllable by the designer. Problems may not be noticed upon immediate implementation of a design, but only later on by a user of the network when such a condition occurs. The tools I describe support the gradual evolution of the knowledge base along with the designers' gradually improving understanding of the domain and the specific design being constructed.

Evaluation of the Mechanisms:

I plan to evaluate my work through both observation of network designers using the system and through experiments. I plan to have two evaluation/user-testing periods, the first of which will aid in locating the aspects of the system that need to be emphasized in revisions to the system implementation. Information on the use of my system in a more natural setting will also be gathered.

Experiments. The first evaluation will occur after the creation of domain-oriented tools for supporting the conversion of knowledge from less formal representations to more formal representations. This evaluation will divide subjects up into two sets, one in which the subjects will use the tools designed to support evolution and the other in which the subjects will not use the tools. The subjects will be given scenarios which contain a problem and solution for the subjects to enter into the system. The tools will be evaluated favorably if the subjects who use the tools put more knowledge from these scenarios in formal representations than those users without the aid of the tools. Beyond the amount of formal knowledge added to the system I will get network designers to critique the organization of the new knowledge in relation to the existing knowledge.

The second round of evaluations will be similar to the first round and will occur after modifications to the system and tools have occurred. These modifications will be based on the outcome of the first round of evaluations, and are meant to remove implementation problems such as interaction style which may be causing the subjects to have difficulties with the tools. If problems with this evaluation process are encountered in the first round of evaluation, changes will be made to try to remove these problems before the second round of evaluations are performed.

Real-use Observations. Beyond the experiments I plan to provide the system to network administrators for their use in real-world network administration tasks. If the system is accepted for real use by network designers I will add a recording mechanism which would provide information concerning the percentage of system suggestions that were accepted. Because of my determination to use standards widely used in the UNIX world, I believe that I will be able to recruit network administrators (including from the computer science departments sysops group) to use my system and provide feedback.

Besides statistics on accepted recommendations I will ask that the network designers send me their knowledge-bases after some use. By comparing the variations of the returned knowledge base I hope to determine which knowledge-base modifications are common across designers and projects and which are different.

Summary:

The problem of knowledge-base evolution is a serious one for the use of knowledge-based systems in rapidly changing domains and in domains dealing with ill-defined problems, such as design. Through using representations of varying degrees of formality I am attempting to reduce the perceived costs for designers adding knowledge to the system. This is to be accomplished by initially allowing knowledge to be added in the format most comfortable to the designer. Over time the designer, with the use of tools, may transform this knowledge into representations with which the computer can computer over and thus provide more services.

Tools to support evolution can merely provide information useful in formalizing knowledge, or can use domain-oriented techniques to make suggestions for the formalization of knowledge in less formal representations. Specifically, the tools I have proposed are to work with textual annotations, PHI-style argumentation, inheritance hierarchies of objects, and critic rules.

This work is being proposed in the context of XNetwork, a knowledge-based design environment to support the collaborative long-term design of computer networks. A prototype of XNetwork exists, but still requires extensions before the basic system is complete. The Evolving Formality Tools will build on top of

the XNetwork basic system. Once implemented the tools will be evaluated through both empirical experiments and observations of real-use situations.

References:

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

[Berlin, O'Day 1990] L. Berlin, V. O'Day, "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.

[Brooks 1987] F. Brooks Jr., "No Silver Bullet: Essence and Accidents of Software Engineering", IEEE Computer, Vol. 20, No. 4, April 1987, pp. 10-19.

[Burger et al. 1991] A. Burger, B. Meyer, C. Jung, K. Long, "Technical Briefing: The Virtual Notebook System", Proceedings of Hypertext '91 (San Antonio, TX), ACM, December 1991, pp. 395-401.

[Carlson, Ram 1990] D. Carlson, S. Ram, "HyperIntelligence: The Next Frontier", Communications of the ACM, Vol. 33, No. 3, March 1990, pp. 311-321.

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

[Conklin 1987] J. Conklin, "Hypertext: An Introduction and Survey", IEEE Computer, Vol. 20, No. 9, September 1987, pp. 17-41.

[Conklin, Begeman 1988] J. Conklin, M. Begeman, "gIBIS: A Hypertext Tool for Exploratory Policy Discussion", Proceedings of the Conference on Computer Supported Cooperative Work (CSCW '88), ACM, September 1988, pp. 140-152.

[Davis 1984] R. Davis, "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 Publishing Company, Reading, MA, 1984, pp. 171-205, ch. 9.

[Ellis et al. 1991] C. Ellis, S. Gibbs, G. Rein, "GroupWare: Some Issues and Experiences", Communications of the ACM, Vol. 34, No. 1, 1991, pp. 38-58.

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

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

[Fischer et al. 1989] G. Fischer, R. McCall, A. Morch, "JANUS: Integrating Hypertext with a Knowledge-Based Design Environment", Proceedings of Hypertext '89 (Pittsburgh, PA), ACM, November 1989, pp. 105-117.

[Fischer et al. 1991] G. Fischer, J. Grudin, A. Lemke, R. McCall, J. Ostwald, B. Reeves, F. Shipman, "Supporting Indirect Collaborative Design with Integrated Knowledge-Based Design Environments", to appear in Human Computer Interaction, Vol. 7, No. 3, 1992.

[Fischer, Girgensohn 1990] G. Fischer, A. Girgensohn, "End-User Modifiability in Design Environments", Human Factors in Computing Systems, CHI '90 Conference Proceedings (Seattle, WA), ACM, 1990.

[Girgensohn, Shipman 1992] A. Girgensohn, F. Shipman, "End-User Modifiability: Tools and Representations", Proceedings of the Symposium for Applied Computing (Kansas City, MO), ACM, March 1992, pp. 340-348.

[Gorry et al. 1988] G. Gorry, A. Burger, J. Chaney, K. Long, C. Tausk, "Computer Support for Biomedical Research Groups", Proceedings of the Conference on Computer Supported Cooperative Work (CSCW '88), ACM, September 1988, pp. 39-51.

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

[Grudin 1992] J. Grudin, "Groupware and social dynamics: Eight challenges for developers", Communications of the ACM, 1992, (in press).

[Hahn et al. 1991] U. Hahn, M. Jarke, S. Eherer, K. Kreplin, "CoAUTHOR - A Hypermedia Group Authoring Environment," in Studies in Computer Supported Cooperative Work, J.M. Bowers, S.D. Benford, eds., Elsevier Science Publishers, North-Holland, 1991, pp. 79-100.

[Halasz 1988] F. Halasz, "Reflections on NoteCards: Seven Issues for the Next Generation of Hypermedia Systems," Communications of the ACM, Vol. 31, No. 7, July 1988, pp. 836-852.

[Halasz 1991] F. Halasz, ""Seven Issues": Revisited", Hypertext '91 Keynote Talk, (San Antonio, TX), ACM, December, 1991.

[Hofmann et al. 1990] M. Hofmann, U. Schreiweis, H. Langendörfer, "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, 1990, pp. 166-179.

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

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

[Kaindl, Snaprud 1991] H. Kaindl, M. Snapru, "Hypertext and Structured Object Representation: A Unifying View", Proceedings of Hypertext '91 (San Antonio, TX), ACM, December, 1991, pp. 345-358.

[Klinker et al. 1987] G. Klinker, J. Bentolila, S. Genetet, M. Grimes, J. McDermott, "KNACK -- Report-Driven Knowledge Acquisition", in International Journal of Man-Machine Studies, Vol. 26, No. 1, 1987, pp. 65-79.

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

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

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

[Marcus, McDermott 1989] S. Marcus, J. McDermott, "SALT: A Knowledge Acquisition Language for Propose-and-Revise Systems", in Artificial Intelligence, No. 39, pp. 1-37.

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

[Marshall et al. 1991] C. Marshall, F. Halasz, R. Rogers, W. Janssen Jr., "Aquanet: a hypertext tool to hold your knowledge in place", Proceedings of Hypertext '91 (San Antonio, TX), ACM, December, 1991, pp. 261-275.

[McCall 1979] R. McCall, "On the Structure and Use of Issue Systems in Design", Doctoral Dissertation (1978), University of California, Berkeley, University Microfilms, 1979.

[McCall 1987] R. McCall, "PHIBIS: Procedurally Hierarchical Issue-Based Information Systems", 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. 1981] R. McCall, I. Mistrik, W. Schuler, "An Integrated Information and Communication System for Problem Solving", Proceedings of the Seventh International CODATA Conference, Pergamon, London, 1981.

[McCall et al. 1990] R. McCall, P. Bennett, P. d'Oronzio, J. Ostwald, F. Shipman, N. Wallace, "PHIDIAS: A PHI-Based Design Environment Integrating CAD Graphics into Dynamic Hypertext", Proceedings of the European Conference on Hypertext (ECHT'90), 1990.

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

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

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

[Patel 1989] J. Patel, "On the Road to Automatic Knowledge Acquisition", in Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI '89), 1989, pp. 628-632.

[Reeves 1991] B. Reeves, "Merging Cooperative Problem Solving and Computer Supported Cooperative Work", Doctoral Dissertation Proposal, Department of Computer Science, University of Colorado, Boulder, 1991.

[Russel 1990] D. Russell, "Hypermedia and Representation," in Hypertext und Hypermedia: Von Theoretischen Konzepten zur Practischen Anweldung, P.A. Gloor and N.A. Streitz, eds., Springer-Verlag, Berlin, 1990, pp. 1-9.

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

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

[Schuler, Smith 1990] W. Schuler, J. Smith, "Authors Argumentation Assistant (AAA): A Hypertext Based Authoring Tool for Argumentative Texts," in Hypertext: Concepts Systems and Applications, A. Rizk, N. Streitz, and J. Andre', eds., Cambridge University Press, 1990, pp. 137-151.

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

[Shipman et al. 1989] F. Shipman, J. Chaney, G. Gorry, "Distributed Hypertext for Collaborative Research: The Virtual Notebook System", Proceedings of Hypertext '89 (Pittsburgh, PA), ACM, November 1989, pp. 129-135.

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

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

[Tatar et al. 1991] D. Tatar, G. Foster, D. Bobrow, "Design for Conversation: Lessons from Cognoter", International Journal of Man-Machine Studies, Vol. 34, March 1991, pp. 185-209.

[Terveen et al. 1991] L. Terveen, D. Wroblewski, S. Tighe, "Intelligent Assistance through Collaborative Manipulation," Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91), August 1991.

[Winograd, Flores 1986] T. Winograd, F. Flores, "Understanding Computers and Cognition: A new foundation on design," Addison-Wesley Publishing Company, Reading, MA, 1986.

Appendix 1:

This appendix tries to provide a detailed look the knowledge required in the system for one segment of the network design scenario in the proposal. Because the XNetwork Formalizer requires the most knowledge of the Evolving Formality Tools, I will look at the section of the scenario in which the Formalizer is used. This is on page 20 and is labeled "Formalizing description of view." This section and the accompanying figure have been reprinted below to reduce the need of looking back to the original scenario.

Formalizing description of view. The Formalizer suggests attributes/values based on the design configuration in the view and the discussion about the design configuration. Based on the text and layout shown in Figure 10 the Formalizer picks out values for common attributes that are needed in the domain. The Formalizer attempts to suggest values for attributes encoded by the design environment developer, who also provides algorithms making suggestions based on the current state of the knowledge base. The suggestions that the Formalizer provides to Paul are shown in Figure 11. In this case the Formalizer has suggested attributes to be attached to this project view which specify the physical devices used, the affected locations, people involved in the design, and the request number. Figure 11 also shows the suggested new issues to the argumentation space which are based on prefabricated issues which are important for the domain. The two issues shown in Figure 11 are based on the general domain issue "How to install a connection to <location>."

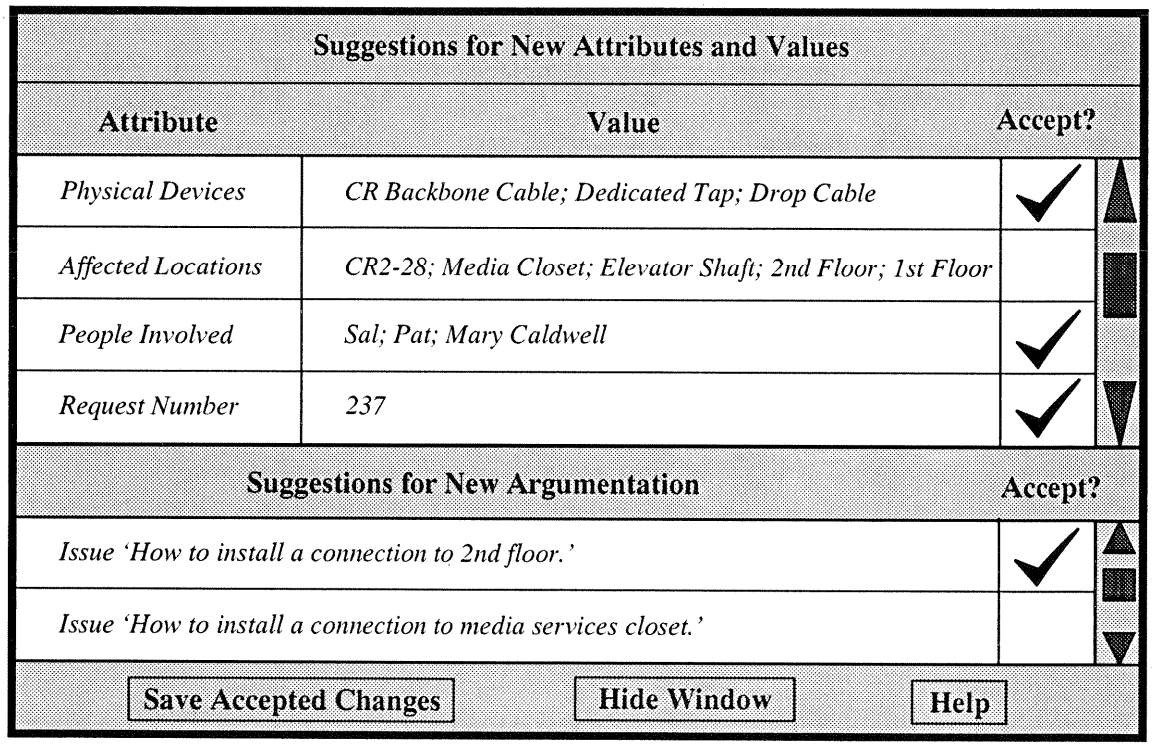


FIGURE 11. A diagram showing the XNetwork Formalizer after Paul asks for suggestions based on the knowledge about the project he has been working on. Paul has accepted three of the four suggested attribute/value pairs and one of the two suggested argumentation nodes shown.

The suggestions made by the Formalizer require a variety of different types of knowledge about the domain of network design, about the building, about the network designers, and about the specific network that is involved. The attributes the Formalizer tries to formulate suggestions for can be thought of as a list of attribute names and methods for computing the suggested value for a particular attribute. This list of attributes and methods is created by knowledge engineers and is meant to support the designers. I will now go through the four suggested attributes shown in Figure 11. Then I will discuss the two issues which the Formalizer suggests being added to the argumentation.

Attribute suggestions. The first suggestion that the Formalizer makes is to create an attribute containing the list of physical devices important for the example. The suggested value is a list of all devices in the example view. This may seem like redundant information since this value could always be computed from the design itself, as it is to create the suggestion. The purpose of having it not be just a computed value is that not all physical devices may be important to a given view they occur in. This separate attribute allows the designer to choose which subset of devices are important enough to be considered as characteristics of the view. The knowledge that the system needs to suggest this attribute's value is the list of all devices in the view and textual descriptions for these devices.

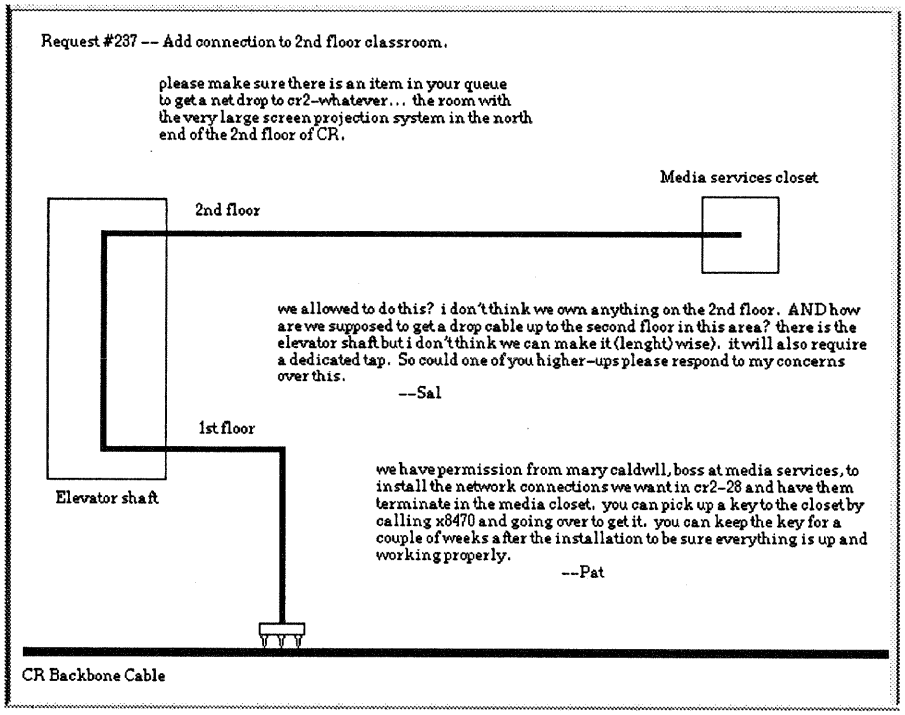


FIGURE 10. The current project is made into an example which can later be examined when similar tasks arise. This view shows the plan for adding the drop cable to the second floor and the discussion about the task.

The second suggestion that the Formalizer makes is to create an attribute that lists the affected locations of the network in the view. To understand the method for computing this attribute's suggested value I have copied Figure 10 from the body of the proposal. The knowledge that is required by this method is to have a

list of phrases that represent locations within the building. This list will need to have synonyms for a single location point to the same conceptual location in the knowledge base. The way in which this method has made its suggestion is to scan all pieces of text in the view looking for these references to locations. In this example there are many such references: 2nd floor, 1st floor, elevator shaft, CR2-28, and media services closet. All references to locations mentioned in the text are then presented as possible locations affected by the design in this view. Because the system is just doing string searches this method will not recognize references to locations not already in its list of references to locations. The designers will be able to add new references to the list. More complicated methods could be used to try to improve the suggestion for this value. Knowledge from the design could be used to add the locations of objects in the view which are not referenced in the text.

The suggestion for the attribute of people involved in this segment of the network is made with a method very similar to the suggestion for affected locations. This time the method uses a list of references to people in the text. New people can be added to the knowledge base by the designer. This method could also use more complicated methods to make its suggestions. The design environment could keep a history of who changes what object and use this information to add to the list of people referenced.

The final suggestion, for request number, is a special attribute which is added for the specific process by which these designers handle requests from users of the network. For this example it is assumed that each request is numbered so that its progress can be tracked. The method suggesting this attribute is a very simple one which looks for a text object labeling the request in this view. One view could have many requests associated with it, so the method should be able to handle such situations.

Argumentation suggestions. The other type of suggestion the Formalizer provides is one for new argumentation. In this example two issues have been suggested by a single mechanism. In general, methods are created by the knowledge engineers which are to suggest new argumentation nodes to the designer based upon the current state of the view and the knowledge base. These methods are likely specialized to create certain types of argumentation nodes, in this example the method creates *issues* about how to connect locations to the network.

This method uses information about where the devices in the view are located, as well as the more general information of all locations in the building which have previously been connected. This information should be computable from the basic design in the design environment. In particular, this method suggests new issues of the form "How to install a connection to <X>" where <X> is each location of a network device in this view which was not previously connected (in other views). While this method suggests new top-level issues other methods might suggest answers, arguments, or examples to already existing chunks of argumentation.

Summary. This appendix has tried to provide a better feel for the types of knowledge and mechanisms required for the most complex of the Evolving Formality Tools discussed in the proposal. It has explained each suggestion individually with respect to the knowledge that the system has and the method for using this information to create a suggestion.