The Design of a Concept Mapping Environment for Knowledge Acquisition and Knowledge Representation

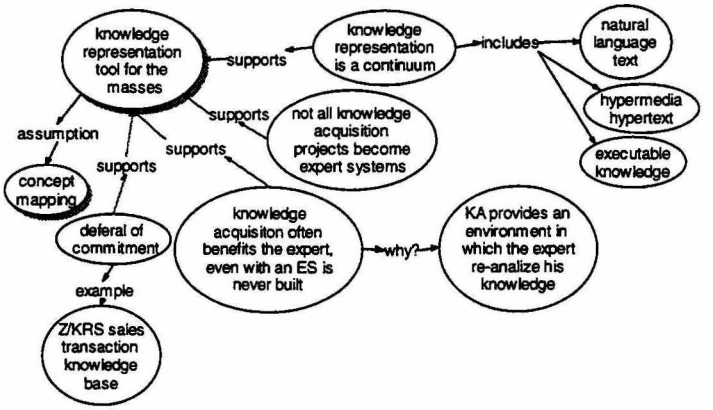
Rob Kremer
Knowledge Science Institute
University of Calgary
Calgary, Alberta Canada T2N 1N4
kremer@cpsc.ucalgary.ca

Abstract

Knowledge representation ranges from very informal to very formal. Informal representations are usually very easy to use and understand for humans, but not for computers. Formal representations are computational, but humans usually find them hard to understand and use. A knowledge acquisition and representation tool which handles both kinds, and the transition from informal to formal, would be very useful. Accord is such a tool, and is currently under development. Accord combines concept mapping and hypermedia to allow the ease of use of an informal system but also provides a set of "formalization operators" which facilitate the conversion of an informal system of concept maps to a formal system which is amenable to computational support.

1. Knowledge Representation for Everybody

Knowledge can be represented by a variety of means, running the gamete from plain natural language text to multimedia hypertext to executable rules and frames. These three examples each represent a broad range of *formality*, that is, all these examples are part of a larger continuum of knowledge formality. For example, natural language text can come in the form of the story "The Boy Who Cried Wolf." This text can be thought of as an encoding for the rule "don't issue false alarms". The text is very informal, in that it does not explicitly state the rule, and one might well miss the rule altogether on hearing the story (as children often do). On the other hand, another piece of natural language text might be the step-by-step instructions for shutting down a nuclear power plant. Presumably, this text would me much more formal in that the rules it encodes are stated very explicitly and unambiguously.



Just as natural language represents a range of formality, multimedia hypertext offers a range of formality. Multimedia adds to natural language the ability to express knowledge in a variety of forms, as appropriate to the particular knowledge, such as diagrams, pictures, animation, audio, and video. Hypertext offers a means of navigation through the knowledge that breaks away form the linear straight jacket of plain text[1]. Just as plain text can be arranged to be formal or informal, so can multimedia hypertext (hypermedia). But multimedia hypertext

[1] Actually, natural language text is not strictly linear: Text forms contain many non-linear devices such as tables of contexts, indexes, references to other sections within the work, and references to other works [McKnight 91]. However, these are required non-linear forms imposed on the underlying linearity, and are not particularly well supported by the media.

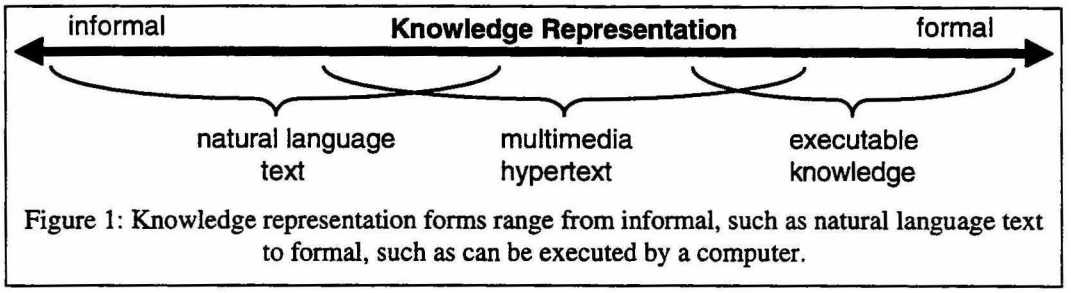


Figure 1: Knowledge representation forms range from informal, such as natural language text to formal, such as can be executed by a computer.

can offer potentially better representational forms and superior navigational abilities that allow the user to obtain a required piece of knowledge more directly and with greater ease.

Of perhaps greatest value are executable forms of knowledge representation. These forms are intrinsically more formal than either natural language text or multimedia hypertext. Executable knowledge representation is exemplified by the knowledge representation languages behind expert system shells and other artificial intelligence applications, but they are not necessarily limited to these expert system applications. For example, the software agent using an executable knowledge representation may do so only for the sake of looking for inconsistencies, omissions, or redundancies; or to translate the knowledge to a more appropriate form. As with the other forms already mentioned, executable knowledge representation covers a range of values on the knowledge representation formality continuum, although common forms available today are at the extreme formal end. The exceptions include systems such as Euclid [Smolensky 87], which represents "semi-formal" knowledge of arguments, and systems such as KRS [Gaines 91b] which is forgiving of users' violation of Classic semantic constraints.

The formality distinction between natural language text, multimedia hypertext, and executable knowledge representation forms is not discrete. These broad forms overlap (figure 10). For example, some natural language texts are much more formal than some multimedia hypertexts -- compare the nuclear power plant shutdown procedures manual and a child's multimedia hypermedia story book. Furthermore, the distinction is further blurred by the fact that some systems are hybrids between the various forms -- most hypertexts contain natural language text passages, and some executable knowledge systems blend the forms. For example, KRS implements a Classic knowledge base, and can make use of the HyperCard [Goodman 87] multimedia hypertext system as an application user interface [Gaines 91c].

Given this theory that these various knowledge representation forms constitute a continuum in formality, it seems natural to ask if some single tool might support the entire range of possibilities. That is, could a knowledge representation tool support easy-to-use informal knowledge representation forms as well as more difficult-to-use formal knowledge representation forms at the same time? Such a tool might support a process of refining informal knowledge to more desirable, executable, formal forms. Indeed, the design of such a tool is the focus of this paper. In this paper, a knowledge representation tool that stretches over informality and formality will be called a *flexible* knowledge representation tool.

Further support for the construction of a flexible knowledge representation tool comes from experiences of knowledge engineers working in the corporate world. Here, as well as elsewhere, not all knowledge engineering endeavors eventually become useful (and used) expert systems. But this does not mean that these efforts are without value. The process of knowledge acquisition has often been seen to benefit the expert: the act of "laying out the domain on the table" seems to force the expert into an introspective mode which sometimes allows the expert to re-evaluate the knowledge, sighting inconsistencies, inferring new relationships, or reorganizing the structure. This re-enforcing of the expert's knowledge is of great value.

A flexible knowledge representation tool could be used to re-enforce and extend an expert's knowledge even when it is presumed that the knowledge cannot be formalized to an executable system. The ease-of-use of a flexible tool's informal end would presumable mean that large investments of time by a trained knowledge engineer could be avoided in favour of the expert being able to do the informal (and relatively unconstrained) knowledge representation task with little outside help. If it later turns out that parts of the knowledge could be converted to executable system, the flexible tool in which the knowledge is already embedded can be used to finish the task.

Even if this not case, if the tool is easy enough to work with, it could provide an excellent form of knowledge documentation for others wishing to examine or learn from the expert's knowledge.

Still more support for the use of flexible tool comes from the software engineering concept of deferral of commitment. Selection of a knowledge representation tool implies commitment to a certain style of representation and a certain method of thinking about the knowledge. An informal system forces no such commitment because it imposes no (or at least minimum) constraints on how the knowledge is to be represented. Thus too-early selection of a possibly incorrect representational form can be avoided.

A case study of just such a situation is described in [Kremer 91] where a expert system was attempted for a certain domain using a commercially available tool which could not, in practice, represent the knowledge because the number of required rules became too complex and unwieldy. The domain was then re-analyzed using a the specification language, Z [Hayes 87] which did not impose many constraints on the knowledge representation. This lead to the discovery that the knowledge format was rules-with-exceptions, which was not available in commercial expert system shells, but which was available in Gaines' KRS system. A proof-of-concept system was successfully developed using KRS.

A flexible knowledge representation tool allows for the initial unconstrained initial representation of the knowledge. Once a significant portion of the knowledge has been captured it can be analyzed with minimal prejudice stemming from the representation. Only when the knowledge structure it is understood well enough would a commitment be made to a particular knowledge execution system (which is presumably supported by the formal end of the flexible tool).

But what is to basis of such a flexible knowledge representation tool? Natural language certainly stretches to the informal end of the knowledge representation spectrum, but it is somewhat deficient when it comes to formal end of the spectrum. Executable languages fulfill the formal end of the spectrum, but they don't fulfill the ease-of-use requirement of the informal end the spectrum. A reasonable starting basis must lay somewhere in the middle inside the multimedia hypertext section of the spectrum. The best attributes of hypertext are its navigational abilities: users can easily move about it in the information, usually by simple point-and-click operations. However, unconstrained hypertext have the drawback of users getting "lost in hyperspace" [McKnight 91] -- the disorienting effect of following interesting links and completely loosing one's perspective of overall structure of the information. While the convenience of hypertext linking should be included in a flexible tool, it is not enough. Multimedia is also a great convenience in that its many forms allow for a wide choice of knowledge expression to humans, however, multimedia doesn't stretch to the formal end of the spectrum; for example, current technology doesn't handle knowledge extraction from video clips well at all.

What is needed is a notational form that can handle informality, formality, can be naturally navigated by hypertext links, can help to mitigate the "lost in hyperspace" problem, and can be annotated in various forms, including hypermedia. One technique that can fulfill these criteria is concept mapping. Concept maps have been used for a long time in various disciplines, they have been applied in very informal situations as well as in very formal situations, they have nodes which can act as hypertext link anchors, they constitute graphs which can act as a view of hypertext links and nodes, and they can be easily annotated in various ways.

The next two sections describe hypertext and concept mapping, respectively.

2. Hypertext

While there are no universally accepted definitions of hypertext or hypermedia[2], Akscyn provides a good definition of its characteristic features [Akscyn 88]:

- "Information is "chunked" into small units, variously called *notecards, frames, notes,* etc. Units may contain textual information. In *hypermedia* systems, units also contain other forms of information such as video graphics, bit mapped images, sound and animation.

[2] This paper uses the word *hypermedia* as the more general term to include both of what is referred to in the literature as *hypertext* and *hypermedia.*

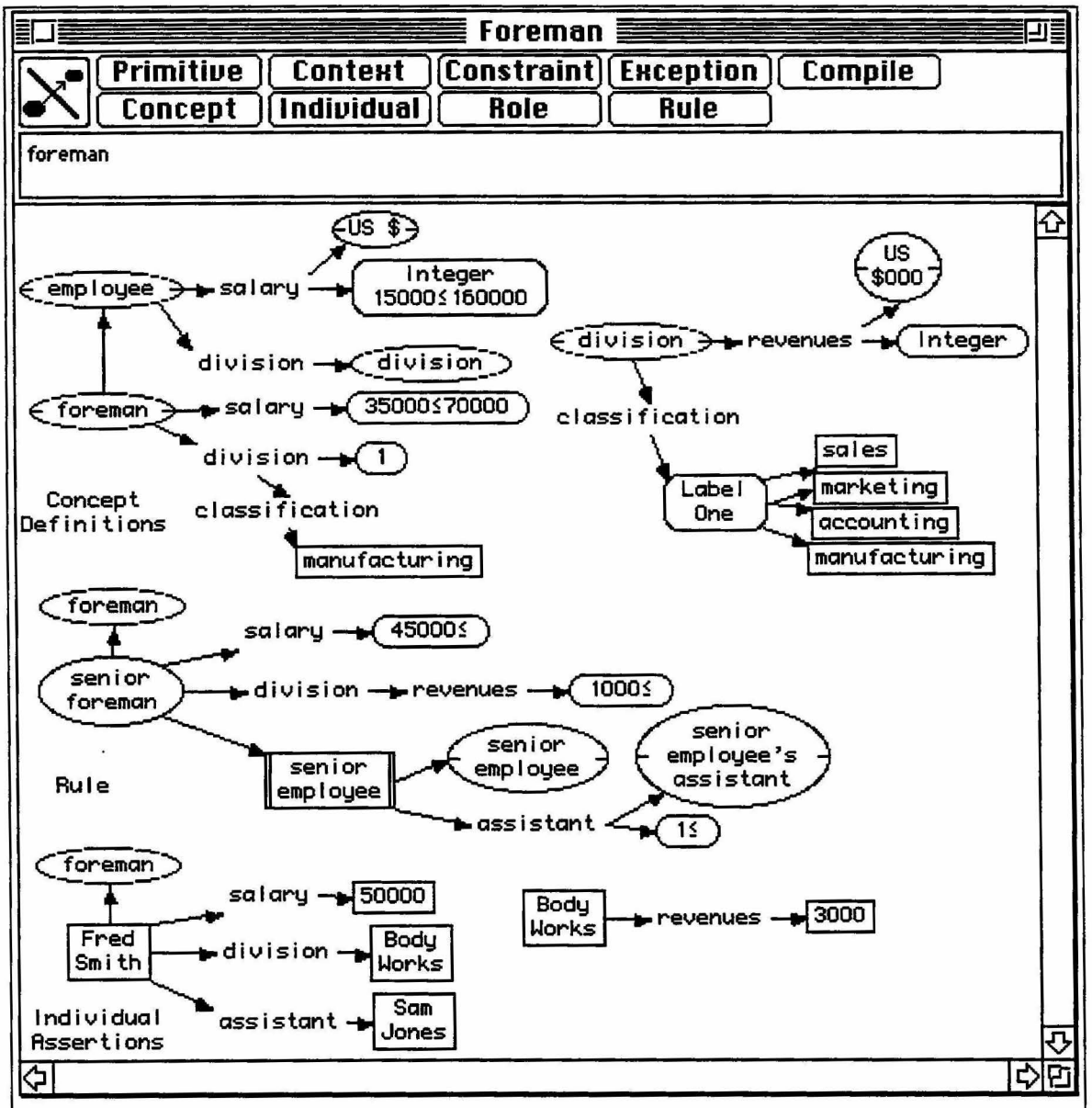


Figure 2: A concept map which is a visual form of a Classic-like language describing rules in an organization.

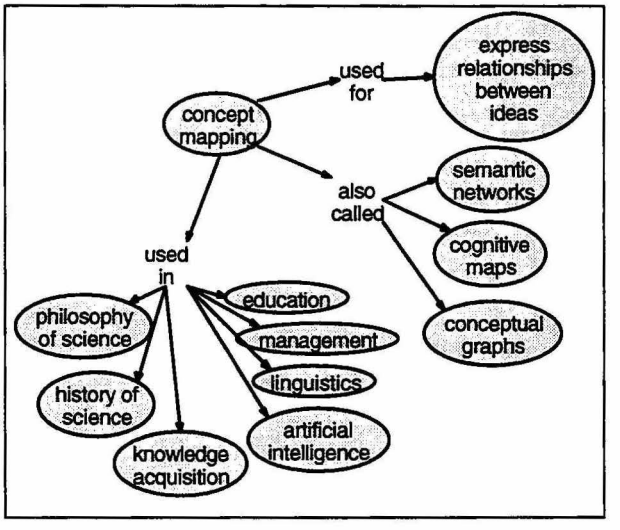
- "Units of information are displayed one per window (systems vary in the number, size and arrangement of windows permitted.)
- "Units of information are interconnected by *links*. Users *navigate* in a hypermedia database by selecting links in order to travel from unit to unit.
- "By creating, editing, and linking units, users build up information structures for various purposes (e.g. authoring documents, developing on-line help systems).
- "In *shared* hypermedia systems, multiple users may simultaneously access the hypermedia database. Shared systems may be implemented as *distributed* systems, in which portions of the database are distributed across multiple workstations and file servers on a network."

Full hypermedia is only now becoming readily and inexpensively available for low-end, desk-top computers. Hypermedia (and its predecessor, hypertext) started in the 1970's on large and expensive machines and eventually propagated to inexpensive desk-top machines, but it has lost some of its functionality (such as network browsing)

along the way. Apple's HyperCard [Goodman 87] on the Macintosh and Asymetrix's Toolbook [Asymetrix 89] on the IBM PC are examples of the recent commercially available systems that contain many powerful features, such as embedded programming languages, but conspicuously lack graphical browsers. Recent public acceptance of hypermedia can be seen by its use in many pervasive software packages, such as the help system in Microsoft's Windows 3.x. More recently, WWW (World Wide Web), a hypermedia system who's hyperlinks span the globe via Internet has enjoyed a huge surge in popularity.

3. Concept Mapping

Concept maps are an intuitive visual knowledge representation technique. They come in a range of forms which are variously referred to as conceptual graphs, semantic networks and cognitive maps. They appear to have more "computational efficiency" [Lambiotte 89] (for humans) than some other forms of knowledge presentation, such as text. Concept maps have been used in artificial intelligence, expert systems and knowledge acquisition, education, management, decision making, linguistics, philosophy of science, and history of science [Gaines 93].



Concept maps are graphs consisting of nodes and arcs, representing relationships between nodes [Lambiotte 89]. The nodes are labeled with descriptive text, representing the "concept", and the arcs are labeled (sometimes only implicitly) with the relationship type.

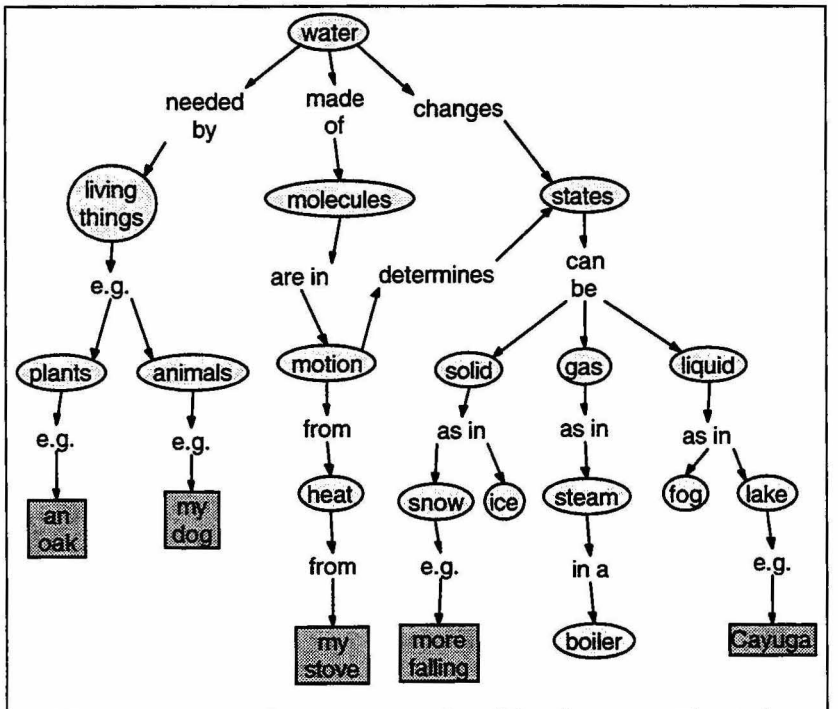


Figure 3: A student's concept map describing the concept "water".

Concept maps can represent knowledge on the very formal level, as in Gaines' KRS system [Gaines 91a], where concept map structures act as a complete interface to the Classic knowledge representation language (figure 2). But concept maps can also represent knowledge on a much less formal level if the system puts fewer constraints on the graphical "syntax". For example, concept maps are used in education [Novak 84] by students to demonstrate their knowledge of a subject to a teacher (figure 3).

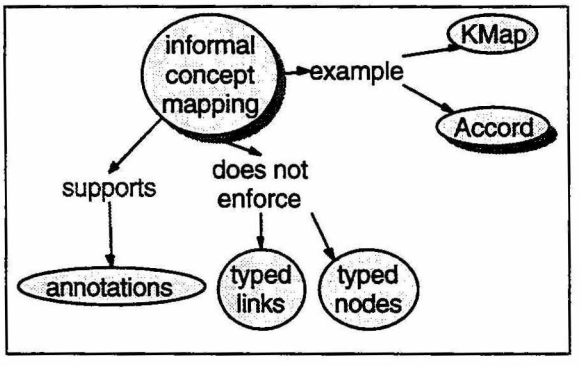
Concept maps tend to be much easier for human users to understand than other knowledge representations such as predicate logic [Nosek 90][3]. These two factors, intuitive understanding and the ability to represent knowledge at both the formal and informal levels, are of

[3] Actually, Nosek compares predicate logic to semantic nets, which generally include more formal semantics than concept maps. But for the current purposes there is not much difference between concept maps and semantic nets.

great importance: They lead to the possibility of novice users being able to express themselves informally and understandably to other users; and, as their proficiency increases, novices can gradually formalize their knowledge as represented in the concept map. As the represented knowledge is formalized, users will avail themselves to more and more assistance from the knowledge interpretation systems in the machine.

3.1. Informal Concept Mapping

Concept maps without formal, computational semantics are not without utility. Although they are not computational from a computer's point of view, for a human they appear to have greater "computational efficiency" than other forms of knowledge representation, such as text [Lambiotte 89]. Informal concept maps have long been used in education both to present concepts and as a vehicle for determining student understanding [Lambiotte 89, Novak 84]. Informal concept maps are usefully applied to knowledge engineering for brainstorming and free association techniques [Sowa 84] where "completeness is more important than neatness and rigor" (p. 299).



Accord [Kremer 93], a program which allows users to draw concept maps with a graphical direct-manipulation interface similar to that of KMap but without semantics, has been applied to brainstorming, planning, knowledge elicitation, documentation, presentation, decision support, and meeting facilitation in such diverse areas as research planning, computer networking, marketing, education, oil and gas strategy planning, computer program design, formal language design, academic paper planning, literature analysis, and personnel planning. The author's experience to date has been that most people enthusiastically adapt to reading and understanding informal concept maps (which concurs with [Lambiotte 84]) and readily learn to create and modify concept maps. The

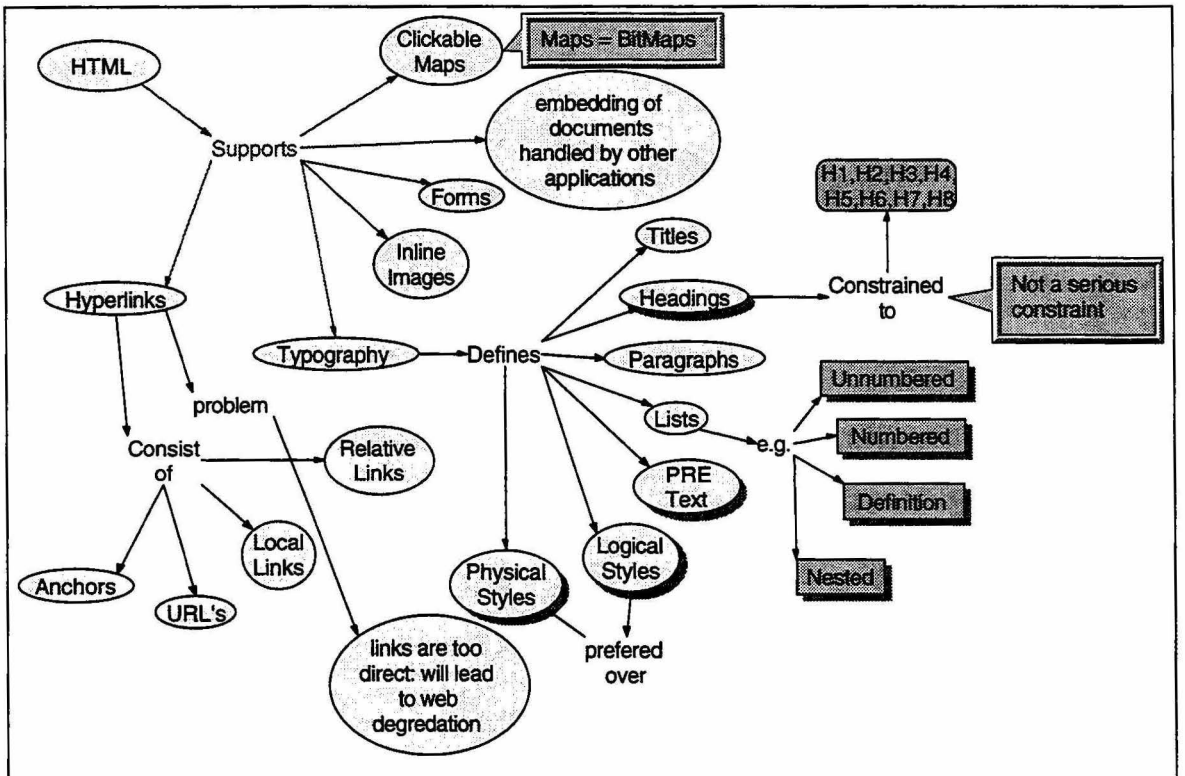


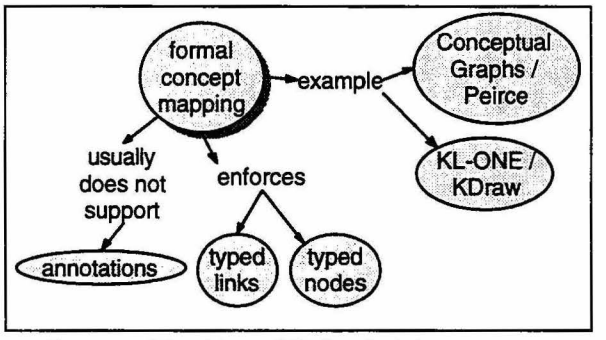
Figure 4: Part of an Accord concept map containing annotations. The annotations are the framed gray boxes. These annotations have an associations with other objects (a node and a link label) in the form of "links" which are visually distinct from other links in the graph.

author has introduced computer-supported concept mapping to individuals who could then independently draw concept maps within five minutes, and to groups using separate workstations with a shared concept map workspace who where collaborating on their topic area within fifteen minutes.

Informal concept mapping systems may or may not support node and link types. If they do support typing on names and links the types either carry no semantic weight, or are not strictly enforced. For example, in the base-level Accord system, nodes and links both carry type information, but the types are used mainly to reflect node and link attributes such as surround shape, fill color, text color, line color. Some semantics may be applied to links in the form of constraints such as "A link of type SUPPORTS may only be draw from a node of type ARGUMENT to a node of type POSITION." However, these constraints are not rigorously enforced: ease of use of more important than rigor, so the user may easily override constraints. Accord also lets users easily add new types at any time.

Informal concept mapping also support annotation: the user should be able to break into plain text (or whatever media is desired) to facilitate the expression of an idea. For example KMap and Accord both let the user add free-floating text boxes as annotations to map. Accord allows annotations in almost any media (text, bitmaps, vector drawings, animation, and audio and video sequences) and allows "links" between annotations and other objects (such as nodes, links, and other annotations) which are visually distinct from the links which are part of the concept map proper (see figure 4).

3.2. Formal Concept Mapping



Concept maps are not computational (formal) unless they have an associated semantics. That is, the maps' node and link types and their interconnections must be constrained to allow for computer support. Indeed, computational concept maps form the primary representation system for many works including language analysis [Sowa 84], Ausubel's assimilation theory [Bradshaw 93] for human learning; Leishman's analogical tool [Lieshman 89] for reasoning about situations based on analogy with previous cases, influence diagrams [Bradshaw 93] for decision support and expert system development, and Gaines' KSSn/KDraw tool [Gaines 91b] for knowledge representation in expert systems. All of these constrain the maps to very specific, if not explicit, types on nodes and arcs.

For example, Gaines' KRS/KDraw tool allows the user to draw concept maps using a direct manipulation graphical interface similar to the MacDraw program (see figure 2). Extensions include arcs who's endpoints move with the node objects and context-specific popup menus. All nodes are labeled and visually typed based on graphic surround (see figure 5). For example, *concept* types have elliptical surrounds and *individual* types have rectangular surrounds. The node types correspond directly to the features in Classic: *primitive concept*, *concept*,

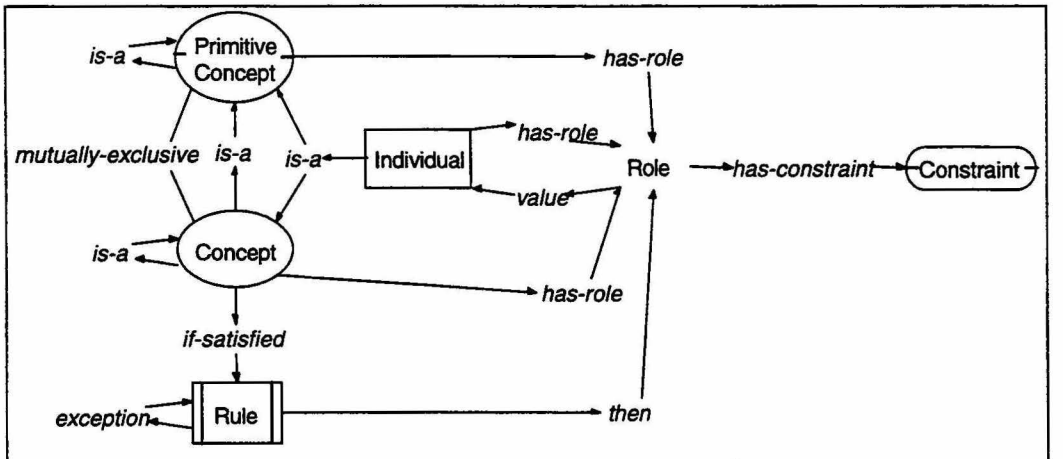


Figure 5: A subset of the KSSn/KDraw concept map types showing the legal link types between the node types. Link types (italicized) are implicit, but completely specified by the types of their terminal nodes, except for *mutually-exclusive* which has no arrow heads.

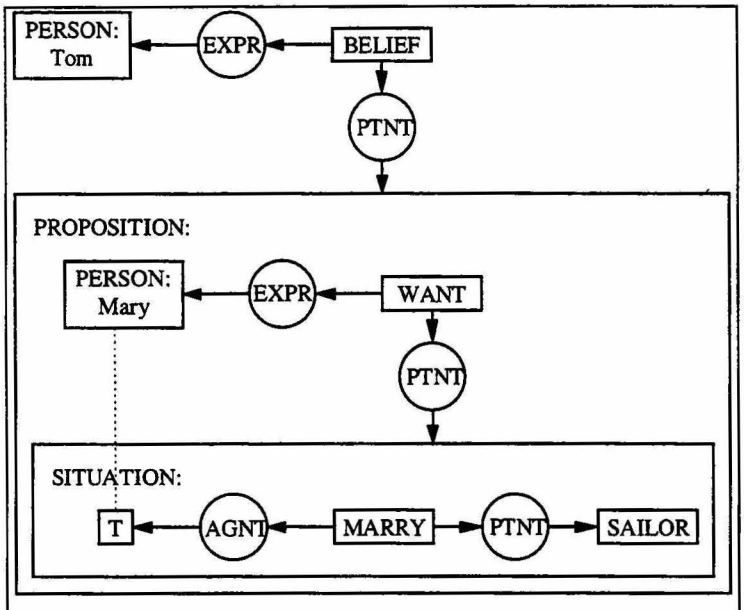


Figure 6: A concept map in the form of Sowa's conceptual graphs describing the sentence "Tom believes that Mary wants to marry a sailor."

individual, *constraint*, *role*, and *rule*. Arcs are unlabeled and only incompletely visually distinguished. For example, *is-a*, *constrained-by*, and *value* type arcs are all represented as uni-directional arrows, while the *mutually-exclusive* type is represented by a simple line. The incomplete visual distinction of arcs does not mean they are not strictly typed -- arc types are always unambiguously specified by the types of their terminal nodes.

KDraw's concept maps are computational in the same way as Classic [Borgida 89] is computational as a knowledge representation and inference language: The class (or type) of an individual can be inferred by the attributes of the individual and those of the concepts, and rules may be fired based on the inferred class of an individual.

Another example of a formal concept mapping system is the *GrIT* [Ellis 92] system which provides a graphical interface to the PEIRCE system which, in turn, implements Sowa's conceptual graphs [Sowa 84] in much the same way as KDraw interfaces to the Classic compiler. These conceptual graphs are capable of cleanly and formally representing an extremely wide range of knowledge forms from predicate logic to natural language. For example, figure 6 is a conceptual graph representation of the surprisingly complex sentence "Tom believes that Mary wants to marry a sailor." The graph is bipartite, with the squared boxes representing concepts and the rounded boxes representing relations between the concepts. The larger boxes are also concepts but are a special kind of concept called contexts. The outer concepts may be read as "There is an instance of a belief who's experiencer in the person Tom, and who's patient is a certain proposition." The proposition is "There is an instance of a want who's experiencer is a person Mary and who's patient is a certain situation." The situation is "There is an instance of marriage who's agent is something (T) and who's patient is some sailor." There is also a coreference link (the dotted line) between the concept [Person: Mary] and the concept [T]. This indicates that [Person: Mary] and [T] are the same individual, which modifies the interpretation of the situation to be "There is an instance of marriage who's agent is a person Mary and who's patient is some sailor."

3.3. Formality is a Continuum

Concept maps may be considered a subtype of graph types which range from extremely informal or nonsensical to formal concept maps (see figure 7). For a graph to qualify as concept map it must have *meaning*. The meaning of the word *meaning* is not well defined, and therefore the distinction between concept maps and graphs is not well-defined. For example, a concept map whose node and type labels are written in Swahili has no meaning to a person who does not understand Swahili, but may be very meaningful to someone who does. Less obviously, a person with a significantly different cultural background (or merely a different ontological model) from that of the graph's author may not judge the a graph to be meaningful, and therefore the critic would not judge the graph to be a concept map. The question of whether a graph is concept map or not then is critically dependent on the observer, or rather on the background and models held by the graph's author and the observer.

Just as knowledge representation techniques constitute a continuum from informal to formal, so do concept maps cover a range of formality. On the informal end of spectrum lay KMap and the base-level Accord system. KMap is less formal than Accord because Accord constrains the maps to be bipartite, encouraging the user to place type labels (text with no surround) on the links. KMap enforces no such constraint; in fact, KMap does not formally distinguish between link labels and nodes.

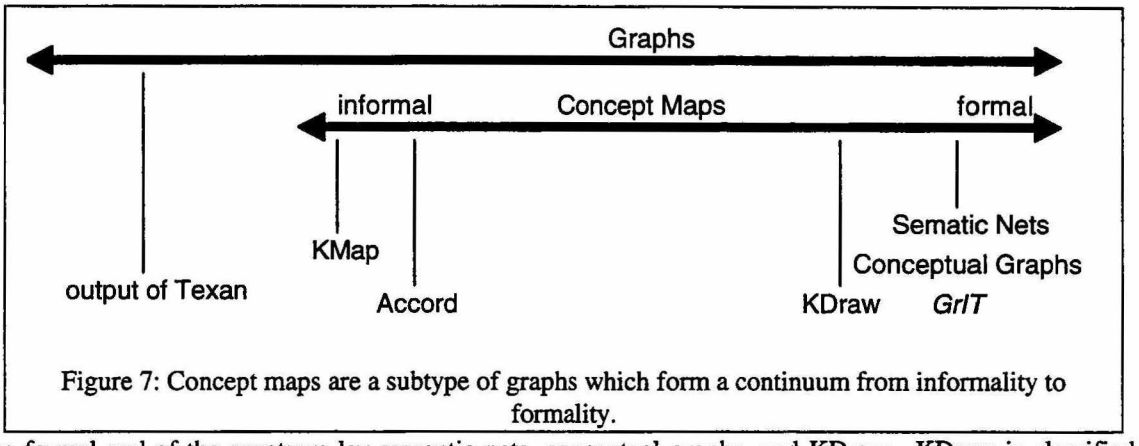
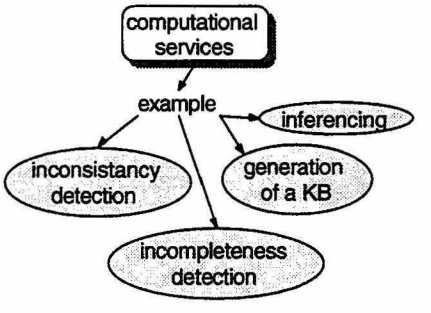


Figure 7: Concept maps are a subtype of graphs which form a continuum from informality to formality.

On the formal end of the spectrum lay semantic nets, conceptual graphs, and KDraw. KDraw is classified as a little less formal that the others because it allows annotation and also allows non-semantic graphs to be constructed, passing only the legal subset of the graph to the underlying Classic engine.

4. System Requirements



This section deals with the basic system requirements for a flexible knowledge representations tool, as outlined in the introduction. Figure 8 summarizes the requirements and the proposed implementation tools (concept maps and hypertext), together with the relationships between them.

One of the first considerations in establishing a set of requirements is the potential users. Since the objectives of a flexible knowledge representation tool range from traditional expert system development to domain expert knowledge refinement to third-party learning, the users community is bound to be quite varied. While the knowledge engineers developing expert systems can be assumed to computer-literate and willing to learn complex systems, this is not true for the either the domain expert or the third-party learner: these people cannot be assumed to be particularly computer-literate or willing to spend much time learning how to use a complex system. The first requirement is therefore *ease of use*.

Ease of use, however, is not one of the primary considerations for an expert user, characteristic of a professional knowledge engineer. This class of users requires certain computational services such as:

- inferencing: the ability to actually execute the knowledge to draw inferences;
- automatic generation of a knowledge base: the ability to compile the knowledge into some other notation that

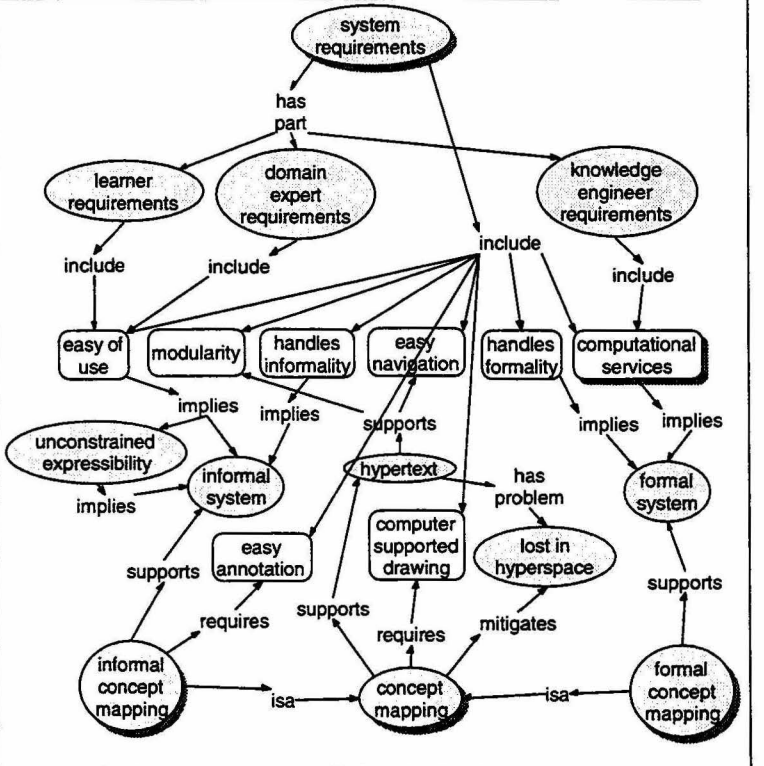


Figure 8: A concept map outlining the basic system requirements for a flexible knowledge representation tool.

can be executed by some other computational agent;

- inconsistency detection: the ability to have the system find errors in the entered data;
- incompleteness detection: the ability to have the system find omissions in the entered data.

Another requirement is therefore the ability to offer *computational services.*

Typically, knowledge bases tend to get fairly large -- too large for most people to grasp all at once. This implies that *modularity* is a requirement: the ability to break up the knowledge base into a collection of simpler views that can be easily understood. If there are to be multiple views, or subsets, of the knowledge base, some system is required to allow the users *easy navigation* throughout the knowledge base.

As already mentioned in the introduction, the ability to handle *informality* as well a *formality* are also required to support the range of formality of knowledge representation.

At this point, it is judicious to consider the implementation tools that could support the identified requirements. As should be obvious from the introductory sections, concept mapping and hypertext are both candidate tools.

The ease of use requirement as well the as the ability to handle informality imply the need for an informal system. Furthermore, ease of use implies that the system should not overly constrain a user's ability to express an idea in a way appropriate to that user and the domain. This is further support for an informal system. On the other hand, the expert user's (knowledge engineer's) need for computational services as well as the need to handle formality both imply the need for a formal system. Fortunately, various flavors of concept mapping techniques support both informal and formal systems.

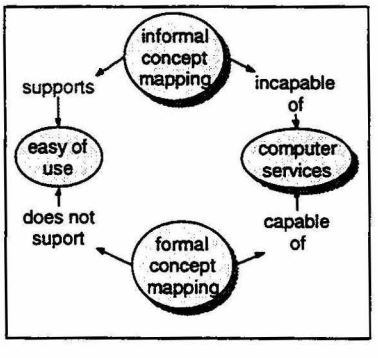
The modularity and easy navigation requirements can both be addressed by hypertext. Modules can be stored in hypertext nodes and the hyperlinks can be used to navigate between them. Hypertext has the problem that people tend to get "lost in hyperspace", but this can often be mitigated by providing a map of the hyperspace. Fortunately, concept mapping techniques provide a very good infrastructure for such a mapping of the hyperspace.

The decision to use concept mapping techniques brings forth a few more requirements. Concept maps are easy to draw on paper, but they are very difficult to modify on paper, usually requiring complete redrawing. The drawing and editing of concept maps must be supported by the computer[4]. Thus, *computer supported drawing* is also a requirement.

While, theoretically, concept maps should be able to express almost anything, this is often not completely practical. Particularly in informal representations, the system should allow the expression of information in whatever form is convenient. So as not to completely break the over-all flavor of the systems, these other representations should be considered as annotations to the concept maps. The annotations should be as flexible as possible ranging from plain text to bitmaps, vector drawings, animation, audio and video. The annotations are, in general, very hard for a formal system to interpret, so they are considered part of the informal system, not part of the formal part of the system. None-the-less, users should be able to add annotation to the concept maps at will, so *easy annotation* is also a requirement.

The requirements are summarized in the rounded rectangles in figure 8.

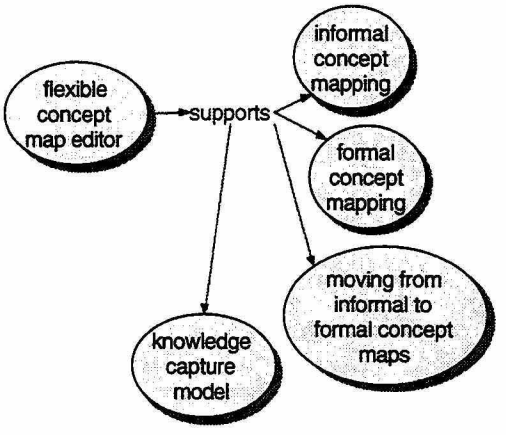
The requirements specify that a flexible knowledge must be an informal, easy-to-use system as well as a formal system offering computational services. On the surface, this would seem an obvious conflict. It is difficult to see how a system can be both formal and informal at the same time. Furthermore, formal systems do not support ease of use, and informal systems do not support computational services. How are these conflicting requirements to be addressed? The next section explains how.



[4] Of course, these concept map drawing programs must have simple and intuitive interfaces.

5. A Flexible Concept Map Editor Addresses the Requirements Conflict

A flexible concept map editor can be informal in that it will allow users to enter any desired constructs, including annotations, without regard to semantics. It can, at the same time be formal in that it may "know" about certain types and type constraints, such that it can prevent non-legal constructs from being passed on to the formal end of the system.



To facilitate such flexibility, the Accord concept mapping system has been designed to be very modular. Figure 9 outlines the architecture. The base-level concept mapping system is almost semantics-free, functioning primarily as a drawing and editing tool and as a user interface to the rest of the system. The actual concept maps can be stored in a hypertext database as nodes in the hyperspace. Each node on a concept map corresponds to a node in the hyperspace, so the concept maps themselves act as maps, or views, of the hyperspace (which is a repository for concept maps). Thus, concept maps are used to help the user visualize the hyperspace, and the hyperspace is used as a repository for the concept maps.

Each hypertext database has an associated type system, or *type theory* (described in detail later). Type theories are stored in a special library, and are imported to a databases at database creation time or at a later time, much like most word processors import "templates". Type theories can be extended and modified by users. The type theory representation language is understood by the base-level concept mapping system. This enables the base-level system to *encourage*, not *enforce*, novice uses to adhere to constraints of the type system. Conformance is encouraged using such techniques as offering a menu of only legal type names when the user creates a link, but allowing the users to type an arbitrary type name if they wish.

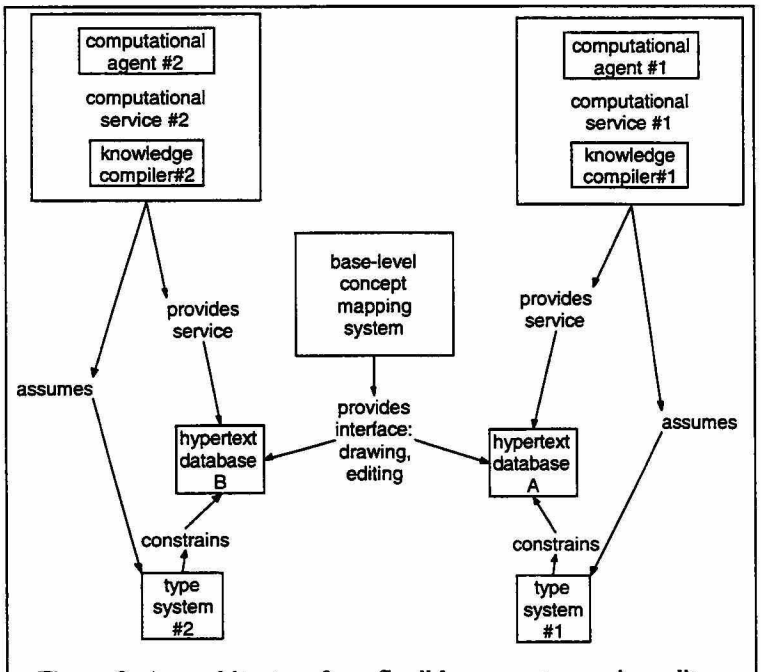


Figure 9: An architecture for a flexible concept mapping editor, taking advantage of a hypertext database system.

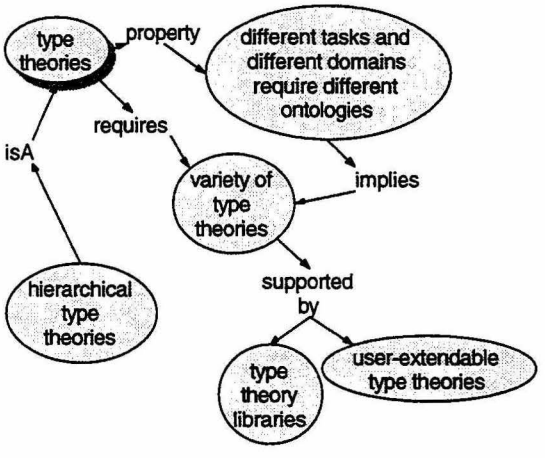
The part of the system that offers computational services is distinct set of modules. These modules interface with the rest of the system via an API (Application Program Interface). A computational service is applied to a hypertext database to provide some specific service. The computational service generally has an associated type theory (which is assumed to have been imported into the target database), such that it can expect to understand at least some of the information in the hypertext database. The computational service cannot assume it will understand all of the information in the database because the user may have chosen not to completely conform to the type theory, or may have extended the type theory with new types. If the user has extended the type theory by inheriting from, or composing from, types that the computational service does understand, then the computation service is expected to interpret these types as the ancestor or compositional types.

Typical computational agents are expected to contain several internal modules. As shown in figure 9, the modules might contain a knowledge compilation module which is responsible for reading the information out of the

hypertext database and reformatting it in some convenient way for the actual computational agent. The computational agent, for example, a knowledge-base engine, does the actual processing -- producing some output, such as feeding back information in the form of additional nodes, links, and annotations to the hypertext database and its concept maps.

All this modularity implies that users may initially represent their knowledge in the form of networks of free-form, informal concept maps using only the informal base system. The concept maps can be manipulated and edited until its inherent structure becomes apparent. If, at this time, the effort of formalization seems worth while, the user may choose (and extend) an appropriate type theory from the library, manipulate the knowledge to fully or partially conform with the type theory, and apply an appropriate computational service. The next three subsections describe type theories, the process of moving form an informal set of concept maps to formal ones, and a more detailed knowledge capture model, respectively.

5.1. Type Theories



The typing applied to nodes and links in a concept maps is meant to serve as an aid both to human understanding and as a basis for semantic interpretation by computational agents. Ontologies designed for human use can be far less well-defined and flexible than can those for computational agents. This alone points to the need for not just one type theory, but a variety of them. Further evidence for a variety of type theories comes from the huge variety of ontologies, both formal and informal, that currently exist. For example gIBIS [Conklin 87] and EUCLID [Smolensky 87], and KRS [Gaines 91b] constitute a range of formality, and all offer dramatically different, though all very useful, ontologies for their various domains. All this implies that a flexible concept map editor must be able to access and use a library of ontologies, or type theories, depending upon the domain of discourse. Thus, Accord is designed to allow users to import type theories from a library into their hypertext databases, and to modify and extend them at will.

A type theory, in this sense, is a database of type information. There are two distinct sections to a type theory: node types and link types. Node type entries encode the following information:

- type name
- parent types
- visual attributes that appear in concept maps: shape, fill color, outline color, and text color
- a template: the nominal data structures (which are themselves annotated concept maps) that always appear in the display of the node of this type
- flags and other information that may be needed by an associated computational service.

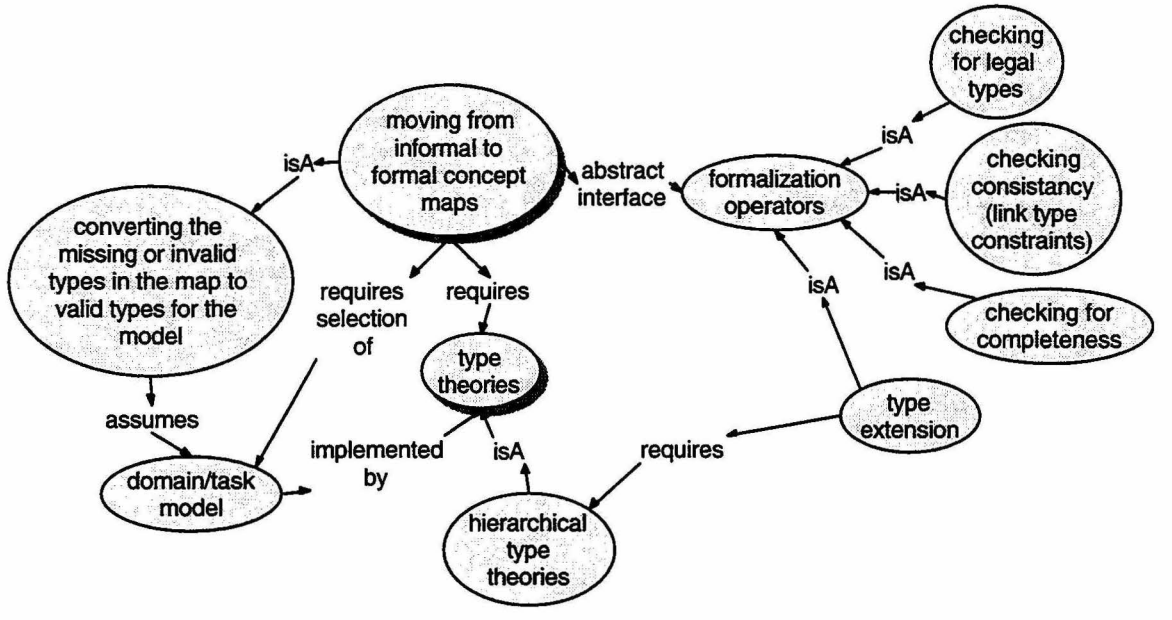
Link type entries encode the following information:

- type name
- parent types
- arity
- visual attributes: arc style, arc color, and label text color
- constraints on type types of nodes that this link may legal join
- a list of attribute/value pairs (analogous to a node's template)

- flags and other information that may be needed by an associated computational service.

Most computational services will be associated with a particular type theory. For example, an inference engine based on Classic (such as that used in KRS) would be associated with a type theory based on figure 5, containing node types *primitive concept, concept, rule, individual,* and *constraint* and link types *is-a, mutually-exclusive, has-role, value, has-constraint, if-satisfied, then,* and *exception.* The nodes in figure 5 are actually node types, and the links (with non-surround labels) are the link types, and specify the restriction on legal link terminators. It is interesting that in this domain, the user has very few link types to actually specify as the system can almost completely infer the link types from the types of there terminal nodes (the only ambiguity is the between link types *mutually-exclusive* and *is-a*).

5.2. Moving from Informal to Formal Concept Maps



A flexible concept map editor should support user modification of an informal network of concept maps into a formal one. The formal system implies the existence of a type theory for which the semantics are defined. So the process of formalization is really the process of converting from a user-defined ad-hoc typing on the concept map to a system-defined typing that can be semantically interpreted by some computational agent. Since the "semantics" of the informal system, by definition, can't be interpreted by the system, this is obviously a task to be performed by the users, who are the sole source of interpretation of the informal concept map. The system can facilitate the users' task by supplying a very good interface for concept map editing, but the task is further facilitated by supplying a convenient set of operators which the users may apply to their maps. These operators include:

- Type extension: Users may extend the type theory by adding types either that inherit from existing types or are altogether brand new. The computational agent's interpretation of the information is not affected by new inherited types as it can merely follow up the inheritance chain until it finds a known type which defines the semantics of the user-defined type[5]. On the other hand, types that are altogether brand new generally cannot be interpreted by the computational agent, so it must either ignore objects of these types altogether or treat them as having type **T**, the universal type.

[5] Note that this behavior is dependent on the type editor's ability to restrict type extensions such that link type constraints are always tightened and never relaxed in child types.

- Illegal types detection: The user should be able to quickly and easily home in on the objects in the concept maps which have types without semantics. One simple method is to have the system highlight all such objects in a map upon user request.
- Constraint checking: Similar to illegal types detection is constraint checking. There may be structures in the concept maps which are legal types but violate integrity constraints; for example, a concept map utilizing the gIBIS type theory (see figure 10) may have a *supports* link between a *position* and an *issue* which is clearly illegal. Again, the system could highlight all such objects in a map upon user request.

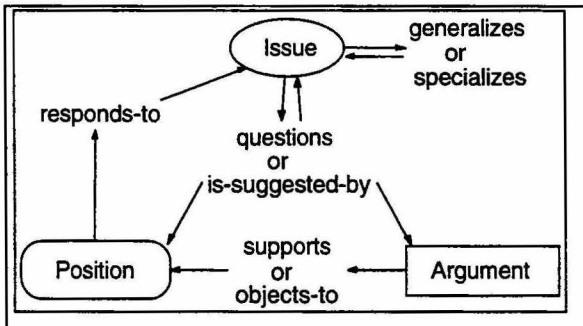


Figure 10: The legal nodes and link types in the gIBIS decision-support model. Links are constrained to connect only the nodes that they connect in the diagram.

- Completeness checking: Certain domains have completeness constraints that, though not strict semantic violations, must be enforced to ensure reasonable processing by a computational agent. The gIBIS model provides a simple example: an *issue* node is of little value as a decision focus if there are no *positions* which have *responds-to* links to it. Such a structure is likely to be incomplete. The system could prompt the user to complete it, such as by drawing in an uninstantiated *position* linked to the offending *issue*.
- Global type substitution: An informal concept map may contain nodes or links of a type x, that has the same intended semantics as that of a different type y, that exists in a formal type theory. This problem can be handled by changing the original type x to a sub-type of type y. However, it may be more convenient to provide an operator which could search the database for type x and change all occurrences to type y.

5.3. A Knowledge Capture Model

A knowledge capture model that takes advantage of a flexible concept map editor is sketched in figure 11. The domain expert or experts freely build up a database of concept maps without the constraints of a formal type system: they leave node and link object types undefined, or they can make up domain-specific type names. No assumptions are made about the structure of the eventual knowledge base.

Once the concept maps are complete enough, they can to analyzed. At this point, a decision can be made as to whether continuing with the project is justified: the informal concept maps may be most valuable as a learning aid to novices, or the knowledge may not be amenable to any practical form a knowledge processing. If it is decided to proceed, some type theory must be selected for the database. Ideally, the type theory may be an existing one,

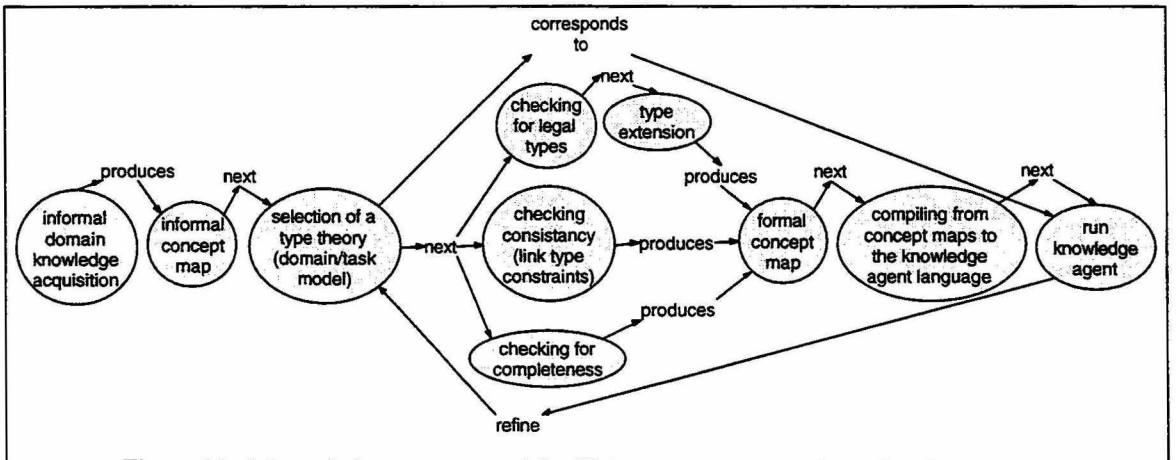


Figure 11: A knowledge capture model utilizing a concept map formalization process.

selected from a library, with an associated computational service. However, there may be no computational service associated with some library type theory, or a brand new type theory may have to be constructed.

Once the type theory has been selected, the process of converting the informal concept maps into a formal one may begin. This process is undertaken by the user with the assistance of the "formalization operators" listed in section 5.2. The process may both modify the concept maps and extend the type theory.

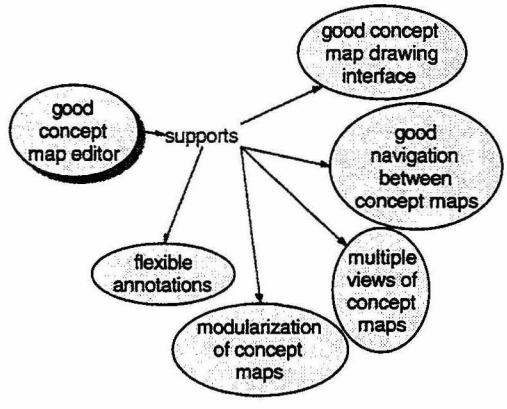
When the concept maps are formalized, a computational service may be applied to the knowledge. Since the concept maps may not be completely formalized, the computational service must be robust in the face of unknown object types, missing data, and constraint violations. Some unknown object types may have known types in their ancestry (these types will take on the semantics of the known types), but others may not, and will likely have to be ignored. Due to the complexity of this data checking, a computational service will usually be divided into two major components: a "compiler" which cleans up the information for the computational agent, and a computational agent which sees only the cleaned-up data. The "cleaning up" is the process of filtering out (or modifying) information and possibly converting the knowledge to a different format.

After the computational service has been run on the knowledge, the process may be iterated back to any point to correct or extend the system.

Early identification of the type theory is useful, particularly in large domains, since it allows for a the development of a small prototype, which can be used to "prove" the viability of the final system before a lot of effort is invested in filling out the entire knowledge base. Knowing the type theory early also means less effort must be extended to convert from the informal to the formal type system.

6. A Good (as opposed to flexible) Concept Map Editor

A "good" concept map editor is one that eases the users' burden of drawing, editing, and keeping track of concept maps. Basic drawing and editing capabilities usually imply direct manipulation [Shneiderman 87] of objects (nodes, links, and annotations). Examples of operators include:



- Object selection for subsequently specified operations
- Group selection
- Mouse dragging for placement of objects or groups of objects
- Fast creation of nodes and links (nodes simply typed, and links "elastic line" drawn between nodes).
- Clipboard copy-and-paste operations

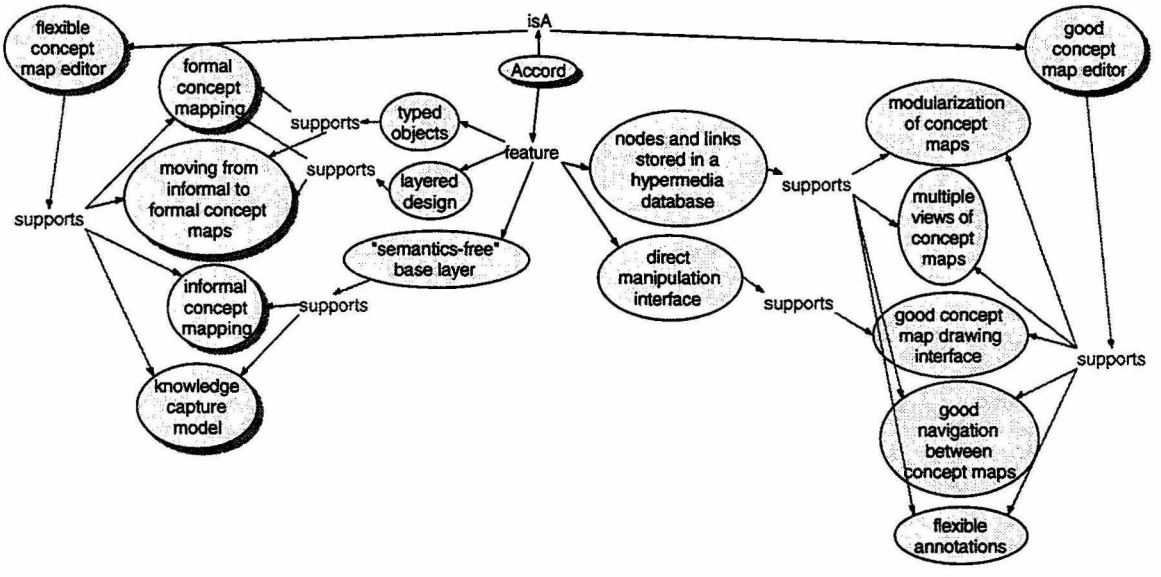
But a good concept editor should also support navigation among concept maps, multiple views of the same concepts, and modularization of concept maps. This is because few real-life domains are simple enough that a single, reasonably-sized concept map can contain all the relevant concepts. Even a work as small as this paper translates into intimidatingly large single concept map. Instead, this paper is better represented as a network of smaller concept maps, some of which are included here as illustrations. A good concept map editor should have some means of navigating this network, beyond mere directories of named concept maps. A far more reasonable approach is to allow hypertext links between concept map nodes and actual concept maps.

If concept maps are to be split into networks of concept maps, there needs to be some means of identifying an individual concept in more than one map. This is most easily done by allowing that textually identical concepts are the same individual concept, however the editing environment must take great care to maintain identity: a user correcting a spelling mistake in a concept name on one map must not disassociate that concept from its other instances in other maps.

A good concept map editor should also support modularization of concept maps; that is, the user should be able to easily split a large concept map into multiple smaller concept maps, and also merge several concept maps into one.

Finally, a good concept map editor should allow users to enter non-concept map objects into a concept map as annotations to support ease of human understanding: Not everything is best represented as a concept map; occasionally a picture or a video clip is more appropriate. Annotations should be visually distinct from other concept map objects to prevent confusion.

7. Being a Good And a Flexible Editor



The flexible knowledge acquisition tool sought at the beginning of this paper can be achieved by the marriage of a flexible concept map editor and a good concept map editor. The Accord project attempts to fulfill this goal. It combines concept maps and hypertext to provide an almost semantics-free concept map editor as the base level with a hypertext database providing storage and navigational capabilities.

Accord's basic organizational model is shown in figure 12. The complete information space available to a user is divided into *topics* which are actually individual databases. However, a database of databases is maintained, which allows for objects in one database to refer to (link to) objects in other databases. Each topic contains a collection of nodes which may be thought of as views or windows. These hypernodes contain graphical objects such as concept maps and multimedia annotations.

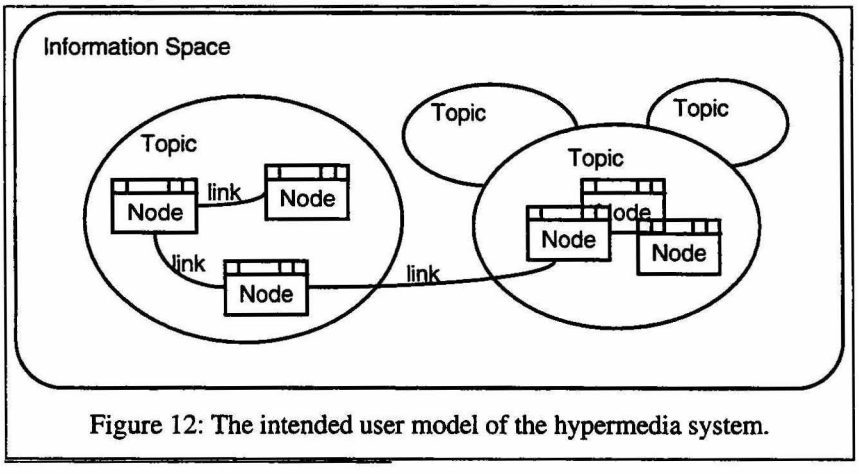


Figure 12: The intended user model of the hypermedia system.

The graphical nodes in concept maps (as opposed to the hypernodes) always refer to a particular hypernode in the information space[6]. Thus, concept map nodes are called node references, and can be mouse-clicked to jump to the view

[6] Actually, concept map nodes do not *always* refer hypernodes: node references may sometimes be "ghosts" in such situations as the referenced hypernode has been deleted or the node reference is a template place-holder for a hypernode that has not yet been defined.

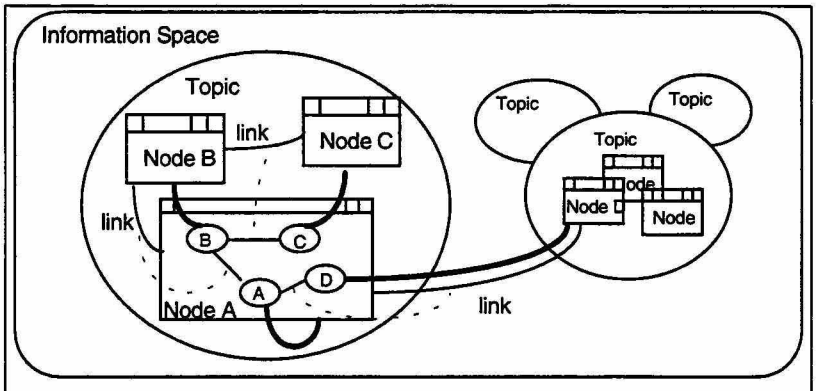


Figure 13: A more detailed view of Accord model shown in figure 12. The reference links between node references in a concept map and the hypernodes are shown as thick curved lines; the actual hyperlinks in the hyperspace are shown as thin curved lines; the references between the concept map links and the hyperlinks are show as dashed curved lines. Note that the concept map is a subview of the information space.

contained in the referenced hypernode.

(The reader will note that there is terminological problem here. There are two distinct classes of objects referred to as *nodes*. The primary objects in the hyperspace are called nodes [the term used in the hypertext literature]; and the objects which are linked together in concept maps are also called nodes [the term used in the concept map and graph theory literature]. Where the distinction is not obvious from context, the objects in the hyperspace will be called *hypernodes*, and the objects in the concept maps will be called *node references*.)

The relationship between hypernodes and node references is shown in figure 13, which is a more detailed version of figure 12. To facilitate the maintenance of the relationship, users creating concept maps also automatically create entries in the hyperspace for each concept map node they create. A similar relationship exists between arcs on concept maps and links in the hyperspace. Thus, although the information in a database may be spread among multiple concept maps, the union of the all these maps may simply be read directly from the database. Individual concept maps are subviews of the knowledge in the database.

Maps may be constructed from information in the hyperspace. A user may place a reference to a hypernode in a blank view (window) and then query the system for all neighbors (nodes that are linked to this node). The system will respond by drawing the neighboring nodes and the labeled links between the original node and these neighboring nodes. In this way, a complete concept map may be quickly built up from the data in the hyperspace. This concept map may be edited to reflect a particular view of the knowledge.

This system also encourages moduralization of concept maps since it is simple for users to clip sections of concept maps and paste them into other concept maps. Once users get used to the fact that the information persists in the hyperspace, they tend to loose inhibitions about deleting information from individual concept maps because they know the information can be recovered from the database.

Concept maps can be annotated by adding annotation objects to them. The annotations range from text to audio and video clips, and are visually distinct from concept map nodes and links. Annotations may also be visually linked to concept map objects view *tethers* which are visually distinct from concept map links (see figure 4).

The features described above relate to Accord as a good concept map editor. But Accord is also designed to be a flexible concept map editor as described in the preceding sections. Accord's base level, as described, is virtually "semantics-free", that it, it does not enforce any constraint on the user other than drawing nodes with labeled links connecting them. But Accord's base level also supports a type system as described in section 5.1.

Each topic contains a type theory which can be imported from a library of type theories, edited, extended, and merged with other type theories. Based on the topic's type theory, Accord implements most of the concept map formalization operators listed in section 5.2. Accord never strictly enforces constraints, but it can inform the user when constraints are violated. Thus Accord can directly support the transition from informal concept mapping to formal concept mapping.

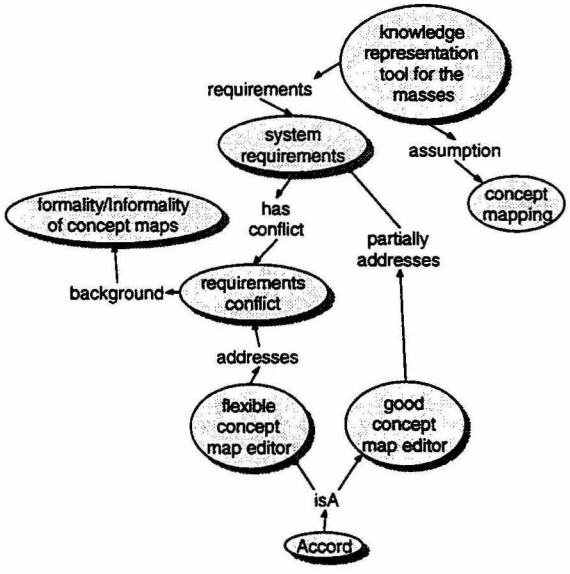
Although it has not yet been implemented, Accord is designed to have an API (application program interface) which will allow other programs (computational agents) to link into it. This will allow the agents to access the data in Accords hyperspace, and individual maps, to perform their tasks. The API will allow agents to both read the data and update it, including both drawing new information into concept maps and updating the hyperspace. The agents may register themselves with Accord, such that agents may be associated with specific type theories and

tasks. This will allow users to access the services of the agents transparently, without leaving the Accord environment.

The features supported by Accord include most of the features required by the knowledge capture model given in section 5.3. Since Accord still lacks the agent interface required by the model, the model (and Accord's implementation of it) cannot yet be verified. Accord has been informally tested on a wide variety of user, and appears to be a good concept map editor, which is required to implement the model to the "selection of type theory" step. Although all of Accord's type theory features are not yet completed, preliminary observations indicate that Accord seems to be very reasonable at supporting the formalization steps, but confirming this part will require a fairly intense study of actual use. Complete validation of the model cannot come until the computational services (agents) are also implemented.

8. Summary

This paper has argued for the a flexible knowledge acquisition tool which handles both formal and informal knowledge representation. The tool should also facilitate the transition from an easy-to use informal representation to a formal representation that can offer computational services such as inferencing.



Concept mapping with hypermedia navigation and multimedia annotation capabilities has been selected for the implementation of such a tool, which is currently in development. A single tool which handle both informal and formal concept mapping is referred to as a *flexible concept map editor*. But, to handle the objectives of knowledge acquisition, a concept map editor must also be a *good concept map editor* which is easily to use and allows for modularization of the concept maps.

The tool being developed, Accord, allows users to easily construct a web of concept maps without the constraints of a pre-defined type theory. This produces a collection of informal concept maps. Once these are analyzed, an appropriate type theory can be chosen from a library of type theories. Accord facilitates the modification of the original concept maps to conform to the chosen type theory through a set of "formalization operators". After the concept maps have been formalized, a computational agent (which handles the type theory), can be applied to provide computational services, such as inferencing, constraint checking, and completeness checking.

References

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

[Asymetrix 89] Asymetrix Corporation. Using ToolBook: A Guide to Building and Working with Books. Asymetrix Corporation, Bellevue, Washington, 1989.

[Borgida 89] A. Borgida, R. J. Brachman, D. L. McGuiness, L. A. Resnick. "CLASSIC: A Structural Data Model for Objects." *Proceeding of 1989 SIGMOD Conference on the Management of Data,* New York, ACM Press, pp. 58-67.

[Bradshaw 93] J. M. Bradshaw, K. M. Ford, J. R. Adams-Webber, J. H. Boose. "Beyond the Repertory Grid: New Approaches to Constructivist Knowledge Acquisition Tool Development." *International Journal of Intelligent Systems*, **8**, February, 1993.

[Conklin 87] J. Conklin, M.L. Begeman. gIBIS: A Hypertext tool for Team Design Deliberation." *Hypertext'87,* November, 198, pp. 247-2517.

[Ellis 92] G. Ellis, R. Levinson, eds. *Proceedings of the First International Workshop on PEIRCE: A Conceptual Graphs Workbench,* Las Cruccs, New Mexico, 1992.

[Gaines 91a] B.R.Gaines. "Empirical Investigation of Knowledge Representation Servers: Design Issues and Applications Experience with KRS." *AAAI Spring Symposium on Implementing Knowledge Representation and Reasoning Systems*, Stanford, March 1991.

[Gaines 91b] B. R. Gaines. "An Interactive Visual Language for Term Subsumption Languages." *Proceedings of IJCAI-91*, Sydney, Australia, August 24-30, 1991.

[Gaines 91c] B.R. Gaines. "Organizational Modeling and Problem Solving using an Object-Oriented Knowledge Representation Server and Visual Language." *ACM Conference on Organizational Computing Systems.* Nov. 5-8, 1991.

[Gaines 93] B.R. Gaines, M.L.G. Shaw. "Supporting the Creativity Cycle Through Visual Languages." *AAAI Spring Symposium: AI and Creativity*, Menlo Park: AAAI. pp. 155-162.

[Goodman 87] D. Goodman. The Complete HyperCard Handbook. Bantam Books, Toronto, 1987.

[Hayes 87] I. Hayes (Ed.). Specification Case Studies. Prentice-Hall, Englewood Cliffs, N.J., 1987.

[Kremer 91] R. Kremer. "Experience in Applying KRS to an Actual Business Problem." 6th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Canada. October 6-11, 1991.

[Kremer 93] R. Kremer. "A Concept Map Based Approach to the Shared Workspace." MSc. Thesis, University of Calgary, Canada, June, 1993.

[Lambiotte 89] J. G. Lambiotte, D. F. Dansereau, D. R. Cross, S. B. Reynolds. "Multirelational Semantic Maps." *Educational Psychology Review,* **1**(4), 1984, pp. 331-367.

[Leishman 89] D. Leishman. A Principled Analogical Tool Based on Evaluations of Partial Correspondences over Concept Graphs. MSc Thesis, University of Calgary, 1989.

[McKnight 91] C. McKnight, A. Dillon, J. Richardson. Hypertext in Context. Cambridge University Press, Cambridge, 1991.

[Nosek 90] J.T. Nosek, I. Roth. "A Comparison of Formal Knowledge Representation Schemes as Communication Tools: Predicate Logic vs. Semantic Network." International Journal of Man-Machine Studies, **33,** 1990. pp. 227-239.

[Novak 84] J.D. Novak and D.B. Gowin, Learning How To Learn, New York: Cambridge University Press. 1984.

[Shneiderman 87] B. Shneiderman. Designing the User Interface: Strategies for Effective Human-Computer Interaction. Addison-Wesley, Reading Mass. 1987.

[Smolensky 87] P. Smolensky, B. Bell, B. Fox, R. King, C. Lewis. "Constraint-based hypertext for Argumentation." *Proceedings of Hypertext'87*, New York, ACM, 1987. pp. 215-245.

[Sowa 84] J.F. Sowa. Conceptual Structures: Information Processing in Man and Machine. Addison-Wesley, Reading, Mass. 1984.