

Analysis and Design of Large Scale Software II (SENG 401)

APPLICATION DESIGN REQUIREMENTS

1 Motivation

The purpose of this document is to present not only the design requirements for the concept mapping application and library, but also to give the rationale behind the design requirements.

2 Assumptions

There are two fundamental assumptions in the application: 1) it has a hypermedia database (hyperbase) consisting of nodes and links, and 2) it uses concept maps as a basic node type which models views of the underlying hyperbase to assist in navigation and database manipulation. Most of the rest of this chapter is dedicated to the details of these two assumptions. Although the application is a hypermedia system, it differs from typical hypermedia systems in that it does not force a strict author/reader role for its users, but rather treats every user as a potential author who may extend the hyperbase.

3 Specification Note

This specification is *leveled*, that is, many aspects are given multiple specifications in the form of "level 1", "level 2", etc. This is done in recognition of the spirit of the iterative prototyping approach. For each leveled aspect, "level 1" is the basic functionality that must be implemented to make a usable product. Subsequent levels reflect progressively greater refinement and functionality. The idea behind this approach is that it minimizes "wrong turns" by allowing one to get a working prototype quickly, then gradually refine the implementation to higher levels once the lower levels have shown themselves to be feasible. Another, more pragmatic, reason for taking this approach is the fact that the complete implementation (at all levels) may not be immediately possible due to various constraints, such as lack of time.

4 Class Library

In order to achieve the objective of creating an easily extendible and modifiable framework for iterative development, most of the application shall be written as a class library. An object-oriented language shall be used because of the ease of coding (using good encapsulation, modularity, and abstraction principles) and also because the object oriented approach is often invaluable in experimenting with quick modification to functionality or appearance of objects.

5 Placement in CSCW

This section deals with classifying the present CSCW (Computer Supported Cooperative Work) system with respect to the categories synchronicity/locality, domain, control of social protocols, media, and intelligent agents.

5.1 Synchronicity/Locality

To coincide with the normal business activities of the target user groups, it is perceived that the application must make virtually no distinctions between the four synchronicity/locality quadrants given in table below. That is, a group of users should be able to use the software together in an electronic meeting room in a face-to-face meeting. After the meeting, an individual user should be able to use the same software to access the same information generated by the group from his computer at his desk as though it was single-user software (asynchronous remote interaction). If a second user starts using the data he should be able to seamlessly (and harmlessly) enter into the environment in a synchronous remote interactive mode.

	Same Place	Different Place
Same Time	face-to-face meetings	synchronous remote interaction
Different Time	asynchronous interaction	asynchronous remote interaction

Interaction modes of CSCW systems.

Allowing for all four quadrants leads to the first requirement (below).

Easy Sharing Requirement	
1	The application should not force the user into any non-trivial registration-and-acceptance procedures.
2	Users should be able to determine when other users are in the same data space and who the other users are.

That is, allowing for normal security measures, users should never be forced to go to elaborate means to enter a data space regardless of whether or not other users are already in the data space. Nor should users using a data space be interrupted to approve other users entering a data space. This is not to say that users will not be *aware* of other users in the dataspace: Users should *be able* to observe others in the dataspace if they so desire.

Furthermore, while working synchronously in the same data space, several users should not be able to inadvertently overwrite each other's work:

Node-sharing Requirement	
1	The application should provide at least single writer, multiple reader access to hyperbase nodes.
2	Users should be able to obtain information about others' use of nodes, such as "Who holds the current write lock?" and "Who else is looking at this node?"
3	Sub-node locking should be granted to multiple writers of a node to the extent of dynamic locks on objects within the node, or even to the extent of dynamic locks on parts of objects within the node (e.g.: two different users may be allowed to manipulate the two endpoints of a line). Multiple writers should always generate animation of each user's state-changing activity if it affects the visible node display.

5.2 Domain

The application should be able to address a wide range of subject domains. The basic concept map facility should be generic enough to handle most domains, such as simple decision making and conceptual design. More extensive domain operations, such as voting mechanisms or CAD-like functions, should be left for later extensions. The current implementation relies on nodes and link typing to address various subject domains; however, there is still the problem of whether to allow user type extension or not (cf. sections 6.2.3. and 6.3.4. for more discussion on that issue).

The application is primarily intended for use by small groups of cooperating individuals. Its open style of interaction and lack of built-in security does not easily lend itself to large groups or confrontational situations.

5.3 Control of Social Protocols

The application does not rely on explicit computer controlled interaction protocols, but relies on normal human social protocols for mediating conflicts. This decision was taken on the basis that one can add explicit control to a system that lacks it, but it is difficult to remove explicit control from a system that is based on explicit control. It should be relatively easy to extend the application to handle control of social protocols. For example, one can imagine "addressing" nodes to individuals or groups where a protocol is enforced over the exchange of such nodes to achieve coordination.

5.4 Media

The application is intended as a "minimum configuration" application. It assumes at least an EGA monitor (640 by 350 resolution, color or monochrome) and a mouse. No additional equipment should be required.

Minimum Hardware/Maximum Function Requirement	
1	The application should not require more than a EGA display and a mouse.
2	The application should <i>be able</i> to take advantage of additional hardware (sound, video, etc.) if available.

The level two requirement is probably best implemented by allowing linking to other software, such as by use of OLE (Object Linking and Embedding) objects. Users with minimal hardware and software who read a node containing advanced features will fail to access the full data, but should be given some reasonable substitute (graceful degradation). For example, a system lacking video capabilities should be able to view the first frame of video clip as a still image.

6 Hypermedia

This section discusses the hypermedia database, links, and nodes. Security issues are not discussed here as this is something that should be added as the need arises as motivated by the user community. It will be interesting to see if the user community can get along with an almost purely open environment. Also not discussed is the issue of version control (except on a class basis). While version control is probably a valuable asset, it may not be worth the rather large storage and complexity overhead it entails.

6.1 Hypermedia Database

The storage mechanism for nodes (the hypermedia database) will be completely abstracted away from any particular model (such as relational database, object oriented database, or file system database). All storage and retrieval will be routed through a single class which will be responsible for interpreting all references, instantiating objects from references, and storing objects. This design allows for a simple storage model, such as one file per node, during the initial implementations and more complex (and efficient) models in future releases without disturbing the majority of the code.

Persistence References Requirement	
1	All persistent node object references, object instantiations from persistent store and object storage shall be routed through a single class interface, which will be the only place where knowledge about persistent references is stored.

Since the application is a multi-user, distributed system, the data is assumed to also be distributed: There is no single, centralized database, but small, distributed databases

throughout the network. The data is divided into topical databases and stored in whatever location is convenient.

Hyperbase Division Requirement	
1	The hyperbase is to be divided into databases (or topics) which represent an arbitrary unit of discourse analogous to a file directory.

In the distributed environment, where it is perceived that whole database sets may be passed around *between* networks (for example on floppy disks), there will not necessarily be any practical way of keeping name servers up to date or unambiguous. It will therefore be necessary to avoid a central name server and let individual databases take care of the task: all references in the database will have to be interpreted *by* the database. This may imply some extra work for system maintenance personnel (who may be the end users) in the event of the physical movement of referenced databases.

No Central Server Requirement	
1	Persistent references should not require a centralized name server, but rather every database should be able to act as its own name server.

Since the application is designed to be an evolving environment, yet one in which real users are expected to actually accomplish work, something must be done to preserve user data over software releases.

Readable Old Data Requirement	
1	Some mechanism should be in place to allow for reading data objects created with older versions of the system.

6.2 Links

This section addresses link anchoring, one way/two way linking, and link typing. More complex navigational issues, such as special indicators, paths, bookmarks, and search and query, are not addressed by the application at this time. These additional features should be relatively easy to add as the need arises as motivated by the user community.

6.2.1 Link Anchoring

In systems such as the application, where nodes may be of arbitrary size and complexity, there is good reason to justify links being anchored to objects within the node. However, sub-node anchors make navigation by agents considerably more complex. It is also unclear how users will use nodes -- nodes may be kept simple enough to render object-to-object links unjustified. For these reasons, the application's basic user links will be defined as node-to-node instead of object-to-object. Note, however, that object-to-object links can easily be implemented *in terms of* node-to-node links via class inheritance, so object-to-object links are not ruled out for future extensions.

Link Anchoring Requirement	
1	Links are anchored at the node level (not directly to objects within a node).

Note that composite links are intrinsically supported by links to hypermedia nodes containing concept maps (described in section 7).

6.2.2 Two-way/One-way

Two way links are seen as important to the application, as per much of the hypertext literature. It is important, both to users and to intelligent agents, to be able to "back navigate" links. However, two-way links must be implemented in terms of one-way links, not only because that is an easy way to do it, but also because it easily yields a degree of robustness in the (not unlikely) event that the other side of two-way link is destroyed or inaccessible. Therefore, one-way links are seen as the base level 1 requirement, and two way links are only added at level 2.

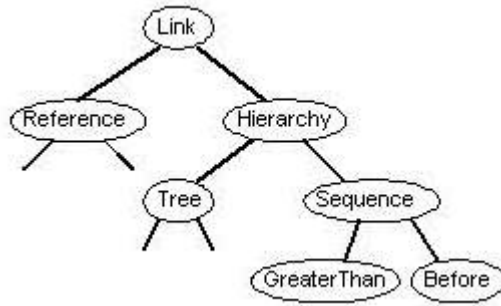
One-way/Two-way Links Requirement	
1	The application shall implement at least one-way hyperlinks.
2	The application shall implement two-way hyperlinks that are robust in the face of inaccessibility of either side of the link.

6.2.3 Typing

Link typing is seen as crucial both for human understanding, and for the use of intelligent agents. Note that the typing referred to here is quite superficial in that it does not imply differences in *behavior* of the links -- it is merely a *type name* that may be interpreted as appropriate by users or agents.

Link Typing Requirement	
1	Links shall have type strings. The interface should encourage users to fill in these type strings instead of letting them default.
2	Links types of two-way links may have a "reverse" name.
3	Link types shall form a type hierarchy.

An example "reverse" name for the link type *part-of* might be *has-part*. "Reverse" names are seen as important as it has been observed that many users seem to have strong (but apparently random) opinions on which way an arrow should point. It may be something to do with which node is perceived as the "focus".



An example link type hierarchy

Link types should be a type hierarchy primarily to assist intelligent agents. For example, in the hierarchy given in above figure, an intelligent agent responsible for printing collections of nodes may only be able to do so over *Hierarchy* links and not on unstructured *Reference* links. The agent only has knowledge of *Hierarchy* link and not of any type below it in the type hierarchy. The user can reasonably hope that the agent can generalize to handle a *Before* sequence of links (an order), and the agent can do this only with the information that a *Before* link "IsA" *Hierarchy* link. Link type hierarchies may also aid in human understanding and refinement of concept maps.

Since the application is not specific to any particular domain of knowledge, one cannot predict in advance all the useful link types. It is therefore appropriate to allow type extension.

Link Type Extension Requirement	
1	The application should allow users to create new link types as appropriate.

6.3 Nodes

This section describes the requirements for nodes in the hypermedia database with respect to node display, node sizing, and node type. Semi-structured nodes are described in section 8.

6.3.1 Node Display

It is perceived that, in the shared environment, users will want to compare and contrast different user's views on the same or related subjects, perhaps while building up their own views. To facilitate this, they should be able to view two or more nodes simultaneously. Fortunately, this is well supported by the basic windows environment. Indeed, it is the normal mode of operation.

Multiple Concurrent Visible Nodes Requirement	
1	Nodes are displayed as their own windows, and users may view many nodes at the same time.

6.3.2 Size

Contrary to many hypermedia systems, it is felt that restricting the size of a node display to a single "idea" is not always appropriate. Since the system cannot predict the natural location, size or shape of the node, it should be left to the user to select the location, size and shape of a node display.

Easy Node Arrangement Requirement	
1	Node windows shall be easily and freely arranged and resized.

Furthermore, the application is positioned on a platform in which many different physical display sizes and resolutions exist. Allowing user node window sizing in this environment automatically implies node window scrolling since a large node created on a large physical display must be scrolled to be viewed on another user's small physical display.

Node Scrolling Requirement	
1	Node windows shall be scrollable.

6.3.3 Typing

Nodes should not be typed except superficially, that is, they should have a type *string*, but there should be no differences in inherent *behavior* between node types. The type string is referred to as the *content type* and should refer to the *intended semantics* of the contents of the node, not necessarily reflecting the *syntactic* contents of the node at all. There is only one node type -- a "whiteboard" on which one can draw concept maps and make annotations of various kinds. The rationale behind the single node type comes from observations of users on a version of the application with several different node types: Users were constantly creating new nodes without having any idea about their eventual type. The single node type scheme solves the problem nicely.

At first glance, the reader may be taken aback by the one syntactic node type assertion since there is obviously a need for differing "types" such as concept map nodes and pure expository text nodes. But these are easily mimicked by the single whiteboard node type: the concept map is simply drawn on the whiteboard, while the pure expository text node is no more than a whiteboard containing a single textual annotation box covering the entire surface. Because annotations are not necessarily textual and can be any arbitrary medium, it is probably best for an implementation to allow annotation by linking to other applications that may be available on the system. While this approach greatly speeds up

development time and flexibility by not requiring the developer to "implement everything", it has the disadvantage of much reduced capacity for sharing among different hardware and software platforms. See section 8 (Semi-structured Nodes) for more on node syntactic typing.

6.3.4 Content Typing

Content typing of nodes is exactly analogous to link typing: The content type is primarily used as an aid to understanding and as a resource for future intelligent agents.

Node Content Typing Requirement	
1	Nodes shall have type strings. Users should be encouraged to fill in these type strings instead of letting them default.
2	Node content types shall form a type hierarchy.

Since the application is not specific to any particular domain of knowledge, one cannot predict in advance the useful node content types. It is therefore appropriate to allow type extension.

Node Content Type Extension Requirement	
1	The application shall allow users to create new node content types as appropriate.

7 Hypermedia Viewers

Overviews of the data in the hyperspace (often called browsers) are seen as critical to the usefulness of the software. The views should be much more than lists of node titles, but should clearly reflect relationships between nodes, both by explicitly labeled links and by spatial layout.

7.1 Concept Maps/Semantic Nets

The basic model of overviews should be in the form of concept maps, or semantic nets as they are often referred to in the literature. Concept maps are not only views of the data, they actually *are* the data in many cases since concept maps are an effective way of presenting and comprehending information.

7.2 Display and Behaviour

It is important for concept maps in the application to have a recognizable and distinctive style to clearly distinguish them from other graphical annotations.

7.2.1 Nodes

Concept map nodes are labeled shapes that each represent nodes in the hyperspace.

Nodes in Concept Maps Requirement	
1	Nodes in concept maps shall represent single nodes in the hyperspace and shall appear as text within a surround. The shape, text color, border color and fill color are dependent on the content type of the hypernode.

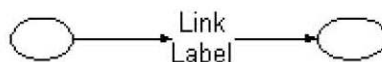
Since nodes represent hyperspace nodes, it is natural that concept nodes are hypermedia link anchors; that is, there should be some simple command to move from a concept map node to a the hypermedia node it represents.

Concept maps in this context are not static displays but are dynamic entities in which users may easily rearrange all elements to enhance understandability through spatial layout. The computer should provide assistance to this end by providing at least minimal layout assistance. Eventually layout assistance may come in the form of a selection of artificial intelligence agents that the user can choose to arrange a particular map.

Node Sensitivity in Concept Maps Requirement	
1	Nodes in concept maps shall be easy to reposition.
2	Nodes in concept maps shall avoid overlapping one another except by explicit user placement.

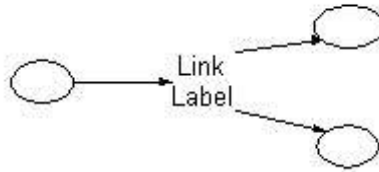
7.2.2 Links

Concept map link display is perhaps the most important element of the concept map design because it conveys the essence of the hyperspace model. The decision has been taken that all links should be explicitly typed and that the type name string should be explicitly displayed on all concept maps. Links are directed arcs between nodes, but there is still considerable design latitude within this constraint. For example, how is the label placement to be handled?



A relatively simple solution is to make links a composite object of the label (which functions as a first-order, moveable graphics object) and an incoming and an outgoing arrow "pinned" to the link label and referent or reference node at either end. This allows the basic interface software classes for both nodes and arcs to take advantage of a high degree of sharing, and gives the user interface elements a high degree of consistency. The label acts as a convenient manipulation "handle" on the link. The label also acts as a "hinge" on the link arc, allowing the user to "bend" the arc around objects intervening

between the reference and referent nodes. Arc bending could be accomplished by other means, but these methods are much more complex from both the developer's and users' points of view.



Furthermore, several hypermedia links may partially "share" a concept map display if they are the same type and share the same reference node. That is, a concept map link may consist of an arrow from a reference node to a link label, and *several* arrows from the link label to different referent nodes. This composite concept map link object represents n hypermedia link objects, where n is the number of arrows radiating *from* the link label. Although this scheme sounds complex in text, it serves to reduce clutter because there are fewer total labels displayed; to increase readability because all like types are gathered together; and to partially conform to the prevalent database record model because link labels may be viewed as multi-valued fields with referent nodes as their values.

Links in Concept Maps Requirement	
1	<p>Links as represented in concept maps may be composite objects representing more than one link in the hyperspace if the links have both the same source node and the same link type. Composite links shall appear in concept maps as three components:</p> <ul style="list-style-type: none"> • A link type label whose text is the link type • A directed arc from the link source node to the link type label • Zero or more directed arcs from the link type label to the destination(s) <p>The type label color and arc colors are dependent on the link type.</p>

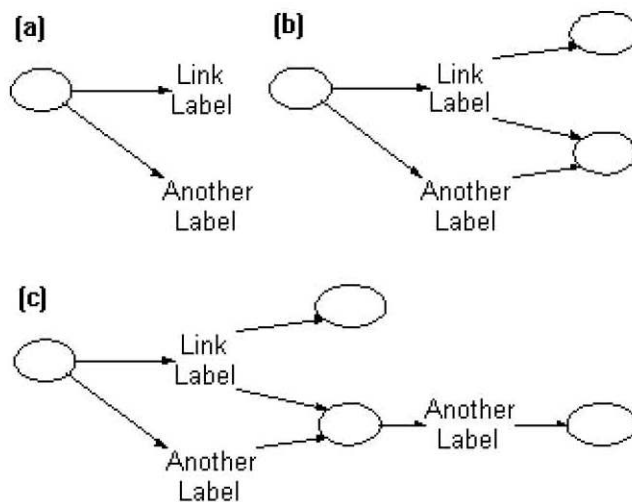
Initial observations with use of the multi-referent link displays indicates that users *expect* to also have multi-*reference* links; that is, link labels with several incoming arrows from reference nodes. However this is not practical because hypermedia links sharing the same display element would have to have *both* a common type and all common referents, which would be rare and would defeat the purpose of reducing clutter, gathering together common types, and modeling database records. Furthermore, obvious editing operations, like elastic band dragging from a link label to a referent node to create a link, would be severely compromised by multi-reference links.

As already mentioned, links should respond to direct manipulation commands in as similar a way as possible to nodes.

Link Sensitivity in Concept Maps Requirement	
1	Link type labels in concept maps shall be easy to reposition. Link anchor points shall automatically move with the reference and referent node.
2	Link type labels in concept maps shall avoid overlapping one another except by explicit user placement.

Since links are the basic mode of navigation within a hyperspace, their navigation properties should also be supported within concept maps. In other words, it should be possible to "grow" a concept map based on existing links just as one can navigate a hyperspace based on existing links. Simple commands should allow a user to visualize, from a specific concept map node, both the links and their referent nodes: The user should be able to *expand* links from a node to produce a "larger" concept map.

Link Expansion in Concept Maps Requirement	
1	<p>It shall be possible to expand links from a particular source node in a concept map in three different ways:</p> <ol style="list-style-type: none"> 1. Expand all links, showing only the link label and the arc from the source node to the link label. 2. Expand all links as in (1), but also show each of the destination nodes and the arcs to them. If a destination node is already on the screen, it shall not be redisplayed, but the arc should be drawn to its pre-existing position. 3. Expand all links as in (2), but recursively up to an arbitrary, user-selectable length.



Examples of the three different expansion types from a single node: (a) A types-only expansion; (b) a single-level expansion; (c) a two-level expansion.

The three expansion types are given in the above figure.

7.3 View Types

A *view* is a concept map where each element of the concept map (its nodes and links) represents a corresponding element (hypernode or hyperlink) in the hyperspace. As noted above, all concept maps are views. There have been two view types identified so far. The first is simply a generic, all-purpose view, and the second is a complete view of a particular database.

7.3.1 Generic Views

Generic views are simply concept maps representing arbitrary nodes and links in one or more databases. There are restrictions on what may or may not be displayed in a generic view.

Contents of a Generic View Requirement	
1	Concept maps display only nodes and links representing nodes and links in the hyperspace, but are unrestricted in that they display some subset of the nodes and links in the hyperspace, not necessarily restricted to a single hypermedia database.

7.3.2 Database Views

The purpose of a database view is to provide a concrete interface to a database. One may think of a database view as an enhanced alternative to the standard windows list box containing a long list of node names. The advantage of database views over list boxes is that the database view can show much more information. Information about nodes is conveyed by the names, shapes and colors; while information about relationships between nodes is conveyed by both links and spatial layout.

Contents of a Database View Requirement	
2	A database view is similar to a generic view, except that it always contains all nodes in the database it displays and never contains any nodes <i>not</i> in the database it displays.

Whether or not the inter- and intra-database links are displayed should probably be a user preference. Database views are not contained in normal nodes but are special purpose nodes for examining databases. So far, no experimentation has been done with database views. Thus, it is unclear whether there should be a single database view for each database, or multiple database views with different layouts. Different spatial layouts may suit different purposes and preferences, but the overhead and possible confusion of supporting a choice of views may not support the benefit.

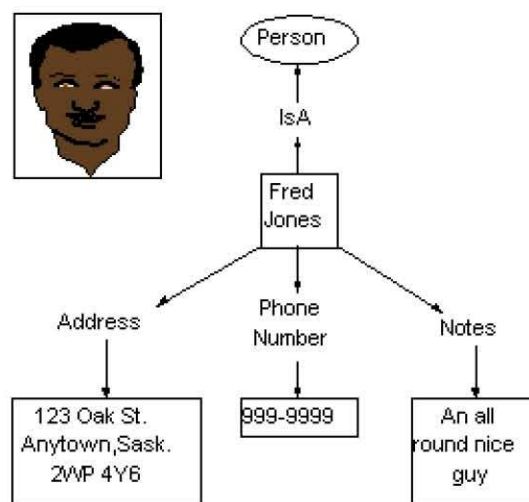
7.4 Annotation

There are occasions where concept maps are not sufficient to express an idea. Frequently, some explanatory text, diagrams, pictures, audio or video are necessary as an accompaniment. Annotations should be seen as being "pasted" on a node's surface; however, often an annotation is semantically associated with some other object on the display and this association should be visually (and perhaps behaviorally) supported in some way.

Annotations Requirement	
1	Users shall be able to annotate the contents of a node display. Annotations are within the scope of the node, and shall not be otherwise visible from the hyperspace.
2	Annotations shall not be limited to text but shall also include graphics, audio, video and whatever other facilities the local hardware and software support.
2	Annotations shall have the ability to be visually associated with other objects on the display.

Because annotations are not necessarily textual, and can be any arbitrary medium, it is probably best for an implementation to allow annotation by linking to *other* applications that may be available on the system. As noted earlier, this approach greatly speeds up development time and flexibility by not requiring the developer to "implement everything", but has the disadvantage of much reduced capacity for sharing among different hardware and software platforms.

8 Semi-Structured Nodes



A "person" concept map

	Name	Fred Jones
	Address	123 Oak St. Anytown, Sask. 2W6 4Y6
	Phone	999-999
Notes	All round nice guy	

A "person" form

It is recognized that some nodes naturally do not contain concept maps. These include information or data nodes that contain text, pictures, video clips, etc. An example of such a node may be a "person" node in a database of people containing name, pictures, addresses, phone numbers and notes. Although such a concept of a "person" is easily modeled by concept maps (see above figures), it is probably much closer to the users' mental models to model "person" as something akin to the relational database model with fields (see above figures). When objects (or fields) are typed, the nodes themselves form a content-based type hierarchy.

Due to limited resources, the application will not directly support semi-structured nodes in the current version, although it should weakly support information nodes through view annotations. Full semi-structured node support implies a type system imposed on the display objects and object combinations (referred to as "node semantic type" in previous sections).

9 Summary

This chapter has outlined the basic design requirements for the application with respect to its placement in the CSCW framework, its hypermedia database requirements, and its viewers. See below for a summary of requirements. These requirements do not actually specify an implementation but serve to restrict the implementation to one within a range of acceptable limits.

Easy Sharing Requirement	
1	The application shall not force the user into any non-trivial registration-and-acceptance procedures.
2	Users shall be able to determine when other users are in the same data space and who the other users are.
Node-sharing Requirement	
1	The application shall provide at least single writer, multiple reader access to hyperbase nodes.

2	Users shall be able to obtain information about others' use of nodes, such as "Who holds the current write lock?" and "Who else is looking at this node?"
3	Sub-node locking shall be granted to multiple writers of a node to the extent of dynamic locks on objects within the node, or even to the extent of dynamic locks on parts of objects within the node (e.g.: two different users may be allowed to manipulate the two endpoints of a line). Multiple writers shall always generate animation of each user's state-changing activity if it affects the visible node display.
Minimum Hardware/Maximum Function Requirement	
1	The application shall not require more than a graphical display and a mouse.
2	The application shall <i>be able</i> to take advantage of additional hardware (sound, video, etc.) if available.
Persistence References Requirement	
1	All persistent node object references, object instantiations from persistent store and object storage shall be routed through a single class interface, which will be the only place where knowledge about persistent references is stored.
No Central Server Requirement	
1	Persistent references shall not require a centralized name server, but rather every database shall be able to act as its own name server.
Readable Old Data Requirement	
1	Some mechanism shall be in place to allow for reading data objects created with older versions of the system.
Link Anchoring Requirement	
1	Links are anchored at the node level (not directly to objects within a node).
One-way/Two-way Links Requirement	
1	The application shall implement at least one-way hyperlinks.
2	The application shall implement two-way hyperlinks that are robust in the face of inaccessibility of either side of the link.
Link Typing Requirement	
1	Links shall have type strings. The interface shall encourage users to fill in these type strings instead of letting them default.
2	Links types of two-way links may have a "reverse" name.
3	Link types shall form a type hierarchy.
Link Type Extension Requirement	
1	The application shall allow users to create new link types as appropriate.
Multiple Concurrent Visible Nodes Requirement	
1	Nodes are displayed as their own windows, and users may view many nodes at the same time.
Easy Node Arrangement Requirement	

1	Node windows shall be easily and freely arranged and resized.
Node Content Typing Requirement	
1	Nodes shall have type strings. Users shall be encouraged to fill in these type strings instead of letting them default.
2	Node content types shall form a type hierarchy.
Node Content Type Extension Requirement	
1	The application shall allow users to create new node content types as appropriate.
Nodes in Concept Maps Requirement	
1	Nodes in concept maps shall represent single nodes in the hyperspace and should appear as text within a surround. The shape, text color, border color and fill color are dependent on the content type of the hypernode.
Node Sensitivity in Concept Maps Requirement	
1	Nodes in concept maps shall be easy to reposition.
2	Nodes in concept maps shall avoid overlapping one another except by explicit user placement.
Links in Concept Maps Requirement	
1	<p>Links as represented in concept maps may be composite objects representing more than one link in the hyperspace if the links have both the same source node and the same link type. Composite links shall appear in concept maps as three components:</p> <ul style="list-style-type: none"> • A link type label whose text is the link type • A directed arc from the link source node to the link type label • Zero or more directed arcs from the link type label to the destination(s) <p>The type label color and arc colors are dependent on the link type.</p>
Link Sensitivity in Concept Maps Requirement	
1	Link type labels in concept maps shall be easy to reposition. Link anchor points should automatically move with the reference and referent node.
2	Link type labels in concept maps shall avoid overlapping one another except by explicit user placement.
Link Expansion in Concept Maps Requirement	
1	<p>It shall be possible to expand links from a particular source node in a concept map in three different ways:</p> <ol style="list-style-type: none"> 1. Expand all links, showing only the link label and the arc from the source node to the link label. 2. Expand all links as in (1), but also show each of the destination nodes and the arcs to them. If a destination node is already on the screen, it should not be redisplayed, but the arc should be drawn to its pre-existing position. 3. Expand all links as in (2), but recursively up to an arbitrary, user-selectable

	length.
Contents of a Generic View Requirement	
1	Concept maps display only nodes and links representing nodes and links in the hyperspace, but are unrestricted in that they display some subset of the nodes and links in the hyperspace, not necessarily restricted to a single hypermedia database.
Contents of a Database View Requirement	
2	A database view is similar to a generic view, except that it always contains all nodes in the database it displays and never contains any nodes <i>not</i> in the database it displays.
Annotations Requirement	
1	Users shall be able to annotate the contents of a node display. Annotations are within the scope of the node, and shall not be otherwise visible from the hyperspace.
2	Annotations shall not be limited to text but should also include graphics, audio, video and whatever other facilities the local hardware and software support.
2	Annotations shall have the ability to be visually associated with other objects on the display.