THE ROLE OF HIERARCHIES

IN DISCUSSION AND ANNOTATION

Thesis submitted in accordance with the
requirements of the University of Liverpool
for the degree of Doctor in Philosophy.

BY

MAHMOUD MOH'D MHASHI

B.S, *Yarmouk University*, 1984
M.S, *University of Colorado at Boulder*, 1988

Department of Computer Science
University of Liverpool

November 1991.

ABSTRACT

Networked, graphical workstations allow people to collaborate in ways heretofore not possible, and one of the common tasks which such computer-supported collaborative work addresses is document production. In many settings people work as teams in discussing, writing, and annotating documents. This thesis explores the requirements for such a system with an emphasis on the knowledge structures, particularly hierarchical knowledge structures, which people want. In developing these requirements, a rapid prototype and test methodology was used. Two software systems, called HERD and MUCH, were created to support multiple experiments and case studies.

HERD (Hypertext Environment for Reasoned Discourse) is a specialized, issue-based information system which represents a discussion as a hierarchical network. In a controlled experiment, HERD well supported an informal discussion about requirements documents. A case study was also performed in which HERD was used to annotate approximately 600 paragraphs of an on-line hypertext. Although HERD provides a useful environment for discussion, it is not especially suitable for annotation.

The MUCH (Multiple Users Creating Hypertext) system was developed to support collaborative authoring and includes features for discussing, authoring, and annotating. The Annotation subsystem was tested by performing an experiment on a few sections of the same on-line hypertext used in the HERD experiments. The users found the MUCH system to be significantly better than the HERD system in supporting annotation.

Additionally, word-based indexing strategies were applied to the discussion and

annotation texts, and indices were automatically produced. An algorithm developed by Mili organized these index terms into a hierarchical, semantic net. According to human subjects the semantic net correctly reflected meaningful relationships among the texts. Users were able, via the semantic net, to effectively browse and search the texts.

A theoretical model of hypertext, based on the above explorations and analyses, has been suggested. In combining discussion and annotation, this model facilitates the design of a single computer system for both discussion and annotation. Systems based on this model would support efficient discussion and annotation activities and provide users with functionalities that other hypertext models do not.

ACKNOWLEDGEMENTS

I would like to express deep gratitude to my supervisor Professor *Roy Rada* for his continued and invaluable advice, help and encouragement throughout the period of this research.

I would like to acknowledge the funding support given to me by *Mu'tah University, Jordan.*

I am grateful to my all colleagues and all the members of staff in the *Department of Computer Science* at *Liverpool University* for their assistance.

Finally, I would like to acknowledge the unending patience, influence and understanding shown by my parents, my wife *Asma Kloub*, and my children *Salsabil, Tasnim, Ishrag*, and *Noor.*

Table of Contents

Chapter 1 — Introduction **1**

1.1. Background 1
1.2. Literature Review 3
1.3. Aims and Objectives 7
1.4. Overview of the Methodology 9
1.4.1. Tool Development 9
1.4.2. Experimentation with Human Users 10
1.4.3. Studies in Cognitive Science 11
1.4.4. Theoretical Modeling 13
1.5. Preview of Topics 13

Chapter 2 — Hypertext Systems **14**

2.1. Introduction 14
2.2. Hypertext Functions 15
2.2.1. Primitive Functions 16
2.2.1.1. Creating Nodes 16
2.2.1.2. Creating Links 16
2.2.1.3. Structures 17
2.2.1.4. Annotating 17
2.2.1.5. Reading 18
2.2.2. High-Level Functions 19
2.2.2.1. Discussion Systems 19
2.2.2.2. Authoring Systems 21
2.2.2.3. Annotation Systems 22
2.2.2.4. Browsing and Searching Systems 22
2.3. Existing Hypertext Systems 25
2.3.1. Emacs INFO 25
2.3.2. EUCLID 27
2.3.3. gIBIS 28
2.3.4. Hyperties 30
2.3.5. IBIS 32
2.3.6. InterNote 33
2.3.7. KMS 36
2.3.8. Outline Processor 38
2.3.9. Quilt 39
2.3.10. SYNVIEW 40
2.3.11. WE 41
2.4. A Model of Hypertext 42

Chapter 3 — Discussion in Hypertext Systems **45**

3.1. Introduction 45
3.2. HERD Tool 48
3.2.1. Data Model 49
3.2.2. High-Level Functionalities 50
3.2.2.1. Discussion Function 50
3.2.2.2. Authoring Function 52
3.2.2.3. Browsing Function 54
3.3. Case studies 58
3.3.1. Case Study 1: Requirements Discussion 59
3.3.2. Case Study 2: Decomposition 63
3.3.3. Case Study 3: HERD as an Annotation System 64
3.4. HERD Experiment 67

Chapter 4 — Annotation in Hypertext Systems **70**

4.1. Introduction 70
4.2. MUCH System 72
4.2.1. Overview 72
4.2.2. Data Model 74
4.2.3. System Functions 77
4.2.3.1. Accessing 77
4.2.3.2. Annotation and Discussion 79
4.2.3.3. Writing 82
4.3. Case Study: MUCH as a Discussion System 84
4.4. Annotation Node Types 89
4.5. Investigations of Annotational Organization and Content 93
4.5.1. Experiment 1: Annotation Organization 94
4.5.2. Experiment 2: Annotation Content 103

Chapter 5 — Browsing and Searching **106**

5.1. Introduction 106
5.2. Algorithms 109
5.2.1. Indexing Algorithm 109
5.2.2. Mili's Algorithm 111
5.2.3. The RELATION Algorithm 113
5.3. Experiments 114
5.3.1. Accuracy and Usefulness of indexing and STATBUILDER ... 115
5.3.1.1. Experiment 1 116
5.3.1.2. Experiment 2 116
5.3.2. Usefulness of Index Terms and Relationships Experiments ... 119
5.3.2.1. Experiment 1 120
5.3.2.2. Experiment 2 121
5.3.3. Similarities of Discussion and Annotation 121
5.3.3.1. Experiment 1 122
5.3.3.2. Experiment 2 123

Chapter 6 — A Discussannotation Hypertext Model **129**

6.1. Introduction 129
6.2. General Requirements 132
6.2.1. Re-analysis of Previous Work 133
6.2.1.1. Discussion Handiling 134
6.2.1.2. Annotation 136
6.2.1.3. Similarities of Discussion and Annotation 138
6.2.2. User Environment for the Model 141
6.3. Practical Issues 146
6.3.1. The Storage Issue 150
6.3.2. The Presentation Issue 154
6.3.3. The Storage and Presentation Specification 158
6.3.4. The User-Model Interaction 163

Chapter 7 — Discussion, Conclusions, and Future Work **166**

7.1. General Discussion 166
7.2. Discussion of Results 168
7.2.1. Discussion Hypertext Systems 169
7.2.2. Annotation Hypertext Systems 178
7.2.3. Browsing Hypertext Systems 188
7.2.4. Discussannotation Hypertext Model 190
7.2.4.1. Hierarchy and Efficiency of the Model 191
7.2.4.2. The Power of the Model 194
7.3. General Conclusions 195
7.4. Future Work 197

References **201**

Appendix 1: List of Figures **209**

Appendix 2: Glossary **211**

Appendix 3: HERD Algorithms **213**

Appendix 4: MUCH Algorithms **235**

Appendix 5: Indexing Algorithms **240**

CHAPTER 1

Introduction

1.1. Background

The widespread use of computer technology and its applications is growing very rapidly in the modern workplace. Because of this it has become increasingly important that computer hypertext systems are designed to be easy to learn and use, and to satisfy user needs. Many hypertext models have been designed, many hypertext systems have been developed, and several universities have created laboratories for research on hypertext.

The complexity of software systems is one of the major problems of hypertext systems. The two key issues associated with this problem are:

1) transfer of user control: sometimes users need to access the function of a second tool while using one tool. This is considered to be an aggravation and a waste of time, regardless of the ease or difficulty associated with the switch and

2) transfer of data: sometimes users may need to transfer data from one tool into another tool to be used for additional tasks. Users can become annoyed by this transfer of data, irrespective of the ease or difficulty of the transfer. Bullen[Bullen1990] observed that a lack of integration is a barrier to the use of some systems that have been designed to support collaborative work.

In order to build a hypertext system that is easy to learn, easy to use, and satisfies the users' needs, the number of functions in a system needs to be made as large as possible. In other words, the number of functions of a system needs to be maximized. Furthermore, the knowledge needed to operate these functions needs to be

made as simple as possible by minimizing its cognitive complexity. This might be done by designing one model that supports these different functions.

From an examination of the existing hypertext systems it can be seen that both Discussion and Annotation are manipulated as different activities; both of them, however, are needed in document creation. Most of the problem solving systems do not explicitly support making Annotation and some of the Annotation systems (such as Quilt) do not support Discussion either. This raises some questions such as:

1) How similar are Discussion and Annotation?

2) Is combining them in a single system useful for collaborative work or not?

3) Is combining both of them in one single system possible or not?

4) If so, then what kind of constraints (such as node and link types) need to be imposed on the combined system?

This led to the exploration of the differences and similarities between Discussion and Annotation, with the goal of developing a hypertext model for supporting both.

In existing Discussion and Annotation systems, different strategies can be found, such as those for supporting structure and node and link types. Current hypertext systems and models either support only hierarchy, non-hierarchy, or both. In the same manner, current hypertext systems support a range of typing strategies. There may be no support for typing at all, or support for a fixed set of node and link types, or users may be given the option to use whatever they want. Thus, the question is suggested: what is the best strategy for supporting structuring and typing? In other words, is it possible to come up with a strategy that can satisfy all or most options in structuring and typing? This leads to the examination of issues such as the role of hierarchy in Discussion and Annotation hypertext systems and models.

1.2. Literature Review

Hypertext systems also help users organize information for the writing task. They can also help in tasks for organizing information into representations more formal than those used in writing. Such tasks include analyzing policy decision-making, capturing early design decisions, and computer-aided design.[Jordan1989] Three main subprocesses– planning, translating, and reviewing – are identified in the problem-solving aspect of writing.[Streitz1989] These processes are not subsequent stages but they show up during the whole course of writing.[Kellog1987]

In the writing task, initially, a Discussion occurs to set up an agenda and to determine the general goals and requirements. After the first draft of the document is prepared, annotations are collected to guide revision of the document.[Rada1991a] Many hypertext systems and models have been developed and implemented. Most of these hypertext systems support either Discussion or Annotation such as IBIS,[Kunz1970] gIBIS,[Conklin1988] SYNVIEW,[Lowe1985] Quilt,[Fish1988] and InterNote.[Catlin1989]

Some hypertext systems were designed for facilitating Discussion such as gIBIS, but they do not support making Annotation. On the other hand, some hypertext systems were designed for making Annotation such as Quilt, but they do not support Discussion. The hypertext systems that partly support making Annotation in addition to other activities are very few. One such system is KMS.[Aksc88] But there is no hypertext system nor any hypertext model that fully supports both Discussion and Annotation.

Examining the existing hypertext systems and models, different variations can be found. One of these variations is the node and link typing strategy. Some hypertext systems (such as IBIS, gIBIS, and InterNote) support using a fixed set of node

and link types. Some systems (SYNVIEW and Quilt) do not support using node and link types at all. Others left node and link typing as an option for the users. So, what is the best strategy for supporting node and link types in hypertext?

Another important variation is the kind of structure that is supported by the existing hypertext systems and models. Some of them support hierarchical structures only such as, SYNVIEW and Quilt. Some other systems support only non-hierarchical structures, such as Hyperties[Shneiderman1986], and others support both hierarchical and non-hierarchical structures such as gIBIS. So, what kind of structure will best represent hypertext?

The directionality is sometimes important for the links, which can distinguish between the hierarchical and the non-hierarchical structures. One of the latest definitions of hierarchy made by Mark Addison[Addison1991] was that the hierarchy is essentially a directed edge labeled graph which consists of a pair *(V,E)* and a root where

1) V is non-empty and
2) the element of E constitutes a directed edge, of not necessarily distinct elements of E, $E \subseteq V \times V$.

In addition each edge $e \in E$ has an associated label. The term root may define the upper-most level in a hierarchy or may be used to define all nodes in a forest of trees. Thus, the systems should support two types of root node: natural root (the first node created in time) and defined roots (user-defined roots).

The term hierarchy refers to different types such as hierarchical decompositions[Cooper1989], hierarchical abstractions.[Rasmussen1985], and hierarchical trees.[Acar1990] Hierarchy has also been used in different ways. In artificial intelligence, hierarchical structures have been used to reveal different details of reasoning

and explanation. In other fields of study, for instance in physics, hierarchies present efficient methods of storing information and data at relevant levels.

Therefore, it is important to be precise about the way in which the concept of hierarchy is being used. The type of hierarchy that has been adopted here is hierarchical trees. It can be defined with respect to a syntax and a semantics:

1) the syntax of hierarchical tree as defined by Knuth[Knuth1973] is a finite set T of one or more nodes such that:
   a) there is one specially designated node called the root of the tree and
   b) the remaining nodes, excluding the root, are partitioned into $m \geq 0$ disjoint sets $T_1, \cdots, T_m$ and each of these sets in turn is a tree. The trees $T_1, \cdots, T_m$ are called the sub trees of the root.
2) the semantics of the hierarchy is the set of full meaning relationships which connect the nodes with each other such as a *Position responds* to an *Issue* or an *Argument supports* a *Position*. The node's content and type should be consistent. If the node type is a *Position*, then the text should be written in the form of answering an *Issue*). The link which connects the two nodes should be created according to the defined set of relations which connect the nodes with each other. Furthermore, the two nodes and the link which connects them should be consistent (i.e if the two nodes are *Issue* and *Position*, then the content of the *Position* should *responds* to the content of that *Issue*).

However, both kinds of structures (hierarchy and non-hierarchy) are important in the hypertext systems. Discourse theorists have identified a host of stable patterns that writers employ on every level of text. These range from sentences and paragraphs, up to grand schemas that outline the structure of an entire text, such as a fairy tale, a resume or a policy argument.[Trigg1986a] They have described heuristics

which support navigational browsing between the hierarchical and nonhierarchical links. Empirical studies of reading comprehension confirm that readers understand and learn from texts more easily when the information is set out in well-defined structures. The text should also provide clear signals of transition from one part to the next.[Charney1987] To read such a document one not only has to make sense of the text at each node, the document must be navigated without confusion. When the text is unstructured, readers can lose track of where they are in the network (and where they have been). As a result, they often read a great deal of material that is not relevant to their purpose.[Yankelovich1985] A large body of research suggests that readers construct hierarchical representations of the text they read. Hierarchical representation is well-suited for writers who wish to argue a single point and provides the reader with a sense of the whole by including high-level overviews.[Krzywiec1983]

Frank Halasz, one of the developers of NoteCards, gathered 1577 nodes. Connecting these nodes were a total of 3460 links, 2521 of which syntax hierarchical links, 261 of which were syntax non-hierarchical, and the remainder were mail links. This example suggests that hierarchical structures are very important in organizing a hypertext network but non-hierarchy is needed also.[Conklin1987b] The disorientation problem, which is less likely in the hierarchical structures, is a major usability problem with the non-hierarchical structures. However, despite the advantages of the hierarchical structures, many hypertext systems support only non-hierarchical structures.

The non-hierarchical structures are important and sometimes are required, but are less common than the hierarchical structures. Some users would like to see multiple perspectives of the same arguments consisting of arbitrary granularities and layouts. The meaning of the arguments can be seen through the relationships between

them. Such relationships can not be represented by the hierarchy alone. With the important value of the non-hierarchical structures, some hypertext systems support only hierarchical structures. On the other hand, some of the hypertext systems support both hierarchical and non-hierarchical structures. However, most of these hypertext systems do not specify when the hierarchical or the non-hierarchical structures should be used and when not.

Finally, two general functions are associated with hypertext systems:

1) the ability to retrieve information in a large information space and

2) the ability to create non-sequential text.[Knuth1990]

The most common method of retrieving information in the hypertext systems is browsing via link traversal. Hypertext systems permit users to easily traverse a large network of nodes containing text, graphics, and/or images. A link might guide the reader to an appropriate node. When a jump to a node is made, the reader can continue to follow further links. Since jumps can be disorienting, it is important that the readers have a clear sense of where they are jumping to and that they can easily return to where they have jumped from.[Faloutsos1990] So, is there any other method except for using the link that can guide readers to the desired information?

1.3. Aims and Objectives

The objectives for this research can be listed as follows:

1) To make a survey of existing hypertext systems and models and to make a comparative analysis in terms of their data models, functionalities, and user interface;

2) To test what kind of structure will best represent of Discussion and Annotation;

3) To test which strategy is the best for supporting node and link types. In other words, should the system support a fixed set of node and link types, or should the users be given an option to use whatever node and link type they want, or a combination of both these strategies;

4) To examine the role of hierarchies in both Discussion and Annotation;

5) To develop a new method for browsing;

6) Applying the new browsing method on the regular text, Discussion text, and Annotation text;

5) Exploring the similarities and the differences between Discussion and Annotation.

6) Defining the general requirements for combining both Discussion and Annotation in one single computer system.

7) Developing a hypertext model that supports both Discussion and Annotation.

There will be many advantages in achieving these aims and objectives, such as:

1) The complexity of the system, which is one of the major problems of hypertext systems, will be diminished by reducing the number of modes. Particularly, the two key issues that are mentioned in Section 1.1; transfer of user control and transfer of data; will be eliminated;

2) The quality of the produced document will be increased by using the three activities; document creation, Annotation, and Discussion; in one single hypertext system; and

3) The user satisfaction will be increased by increasing the number of system functionalities.

1.4. Overview of the Methodology

In order to achieve the aims and objectives that are mentioned earlier, different strategies were formulated. Two software systems were implemented and a set of experiments and case studies were performed to test a set of different issues regarding Discussion and Annotations. Three different algorithms were used to develop a new browsing method and to examine different issues regarding the similarities and the differences between Discussion and Annotation. Finally, a model of hypertext was developed with the consideration of the results from this and other work.

Different methods were used to measure the success in resolving the problems. Some of these methods are: quantitative assessment, statistical tests, and efficiency measurement. Some of were made automatically and others made manually. The key-strokes might be one way to measure the efficiency.

1.4.1. Tool Development

Most of the hypertext systems, not only focus on one activity such as Discussion or Annotation, but also support only one strategy, such as supporting one kind of structure, hierarchical or non-hierarchical. A Discussion system and an Annotation system were developed and implemented to be used as tools to test a set of different issues in Discussion and Annotation.

A system called HERD was developed by the author. HERD (Hypertext Environment for Reasoned Discourse) is an issue-based information system (IBIS) which supports informal Discussions about requirements documents. HERD is similar to most existing hypertext systems in that the text is attached to the nodes. HERD supports hierarchical structures and the semantics of non-hierarchical structures. A set of case studies and experiments were performed using HERD to

examine the role of hierarchy in a Discussion and to explore issues related to Discussion.

The author has participated in the development of a system called MUCH. The author was responsible for the sub-system which supports making Annotation. MUCH supports authoring and Annotation. Some experiments and case studies were performed to explore of issues regarding Annotation, to answer some questions about Annotation, and to examine the role of hierarchy in Annotation.

1.4.2. Experimentation with Human Users

In order to test different issues and to answer different questions regarding Discussion and Annotation, two sets of experiments and case studies were performed using HERD and MUCH. In the experiments, prior knowledge about the hypothesis means that it is possible to predict whether or not it will be supported. In the case studies in the other hand, the lack of prior knowledge about the hypothesis means that it is not possible to make such a prediction. This is the key difference between the experiments and the case studies.

The first set of experiments and case studies tests the different issues regarding Discussion, while the second set tests the different issues regarding annotation. The first set consists of three case studies and one experiment (Sections 3.3 and 3.4 respectively). The first case study (Section 3.3.1) tests:

1) using the semantics of hierarchy in discussing requirements,

2) using a fixed set of node and link types strategy, and

3) the role of hierarchy in Discussion within one group.

The second case study (Section 3.3.2) tests using the hierarchy in Discussion within more than one group. The third case study (Section 3.3.3) tests using the Discussion

system HERD as an Annotation system in a task involving creating annotations. The experiment (Section 3.4) tests:

1) using both the syntax and the semantics of hierarchy and

2) using the opened optional of node and link types strategy.

The second set consists of one case study and two experiments (Sections 4.3 and 4.5 respectively). The case study tests:

1) the usefulness of using the Annotation system MUCH in a task of creating annotations and making Discussion and

2) the strategy of allowing the user to choose the fixed and/or user-defined node and link types strategy.

The first experiment (Section 4.5.1) tests:

1) using a fixed set of node and link types,

2) the usefulness of using a specific set of Annotation node types, and

3) the role of hierarchy in Annotation.

Finally, the second experiment (Section 4.5.2) tests:

1) the strategy of allowing the user to define node and link types and

2) the importance of using three different approachs for making annotations.

1.4.3. Studies in Cognitive Science

Hypertext augments text by connecting it to a semantic net. In MUCH (Section 4.2), a collaborative authoring hypertext system, text points to *links* (instead of *nodes*) in the semantic net. Building a semantic net, or a knowledge base, is a time-consuming, conceptually challenging task.[Rada1987] Further, connecting text to a semantic net consists of no less than conceptual content analysis, an equally chal-

lenging task. Information retrieval research has shown the value of indexing strategies based on word frequencies in textual documents[Salton1983] Such methods have the advantage of practicality and ease of implementation. However, they describe documents by a flat set of index terms, instead of relationships between concepts as required by MUCH. An algorithm developed by Mili organizes a set of index terms in a hierarchy based on frequencies of co-occurrences in document indices.[Mili1987] In Chapter 5, the extent to which word-frequency indexing may be augmented with Mili's method to provide both a relational indexing for documents, and a semantic net for browsing was explored.

Two software algorithms– Salton's and Mili's algorithms– were used to build a semantic net. The first algorithm (Section 5.2.1) was used to index text blocks for a draft textbook entitled "HYPERTEXT from Text to Expertext", Discussion text, and Annotation text. The second algorithm (Section 5.2.2) was used to organize the set of index terms in a hierarchical semantic net. Another algorithm (Section 5.2.3) developed by the author was applied to the semantic net and the index vocabulary to draw the relationships among the text blocks.

Six experiments (Section 5.3) were performed to test the new method of browsing and its application on regular text, Discussion text, and Annotation text. The first two experiments (Section 5.3.1) were performed to test the accuracy of the first two algorithms. The next two experiments (Section 5.3.2) were performed on regular text to test the accuracy of the new method of browsing. The last two experiments (Section 5.3.3) were performed to test the application of this method of browsing on both Discussion and Annotation with the goal of finding some similarities and/or differences between the two.

1.4.4. Theoretical Modeling

The number of similarities between Discussion and Annotation was significant and this led to the development of a Discussannotation model (Section 6.3) which supports facilitating Discussion and making annotations. The requirements that have been placed on this Discussannotation model (Section 6.2) come from the results of this work and the general literature on hypertext systems. This model is more efficient in some respects and has greater functionality than other models. Such functionalities are necessary for document creation.

1.5. Preview of Topics

In this Chapter the topics, the objectives, and the methodology are previewed. Chapter 2 presents a comparative analysis for some of the existing hypertext systems and models, and an overview of a model of hypertext. Chapter 3 explores some issues and answers some general questions about Discussion in hypertext systems. In a similar manner, Chapter 4 explores some issues and answers some general questions about Annotation in hypertext systems. A new method of browsing and its application on both Discussion and Annotation has been described and tested in Chapter 5. Additionally, further explorations regard the similarities and the differences between discussion and Annotation have been made in this Chapter. In Chapter 6, a model of Discussannotation hypertext which supports both Discussion and Annotation, and its general requirement specifications were described. Discussion and conclusions are presented in Chapter 7, together with directions for future work arizing from this thesis.

CHAPTER 2

Hypertext Systems

2.1. Introduction

Hypertext is a valuable contribution to the information age, allowing readers to access related information through machine-supported links.[Zellweger1989] Yet, there is a lack of consensus as to a specific definition of hypertext.[Begoray1990] Conklin[Conklin1987] defined hypertext in terms of its features. These include: A network of textual or graphical nodes;

2) Windows correspond to nodes on one-to-one basis;

3) Window operations are supported (repositioned, resized, etc.);

4) Any number of link icons on each window;

5) The ability for the user to easily create new nodes and links; and

6) The ability to browse the database by following links, searching and navigating using browser.

Smith and Weiss[Smith1988] defined hypertext as an approach to information management in which data is stored in a network of nodes connected by links. Akscyn and Yoder[Akscyn1988] suggested a set of features for defining hypertext. These included information chunked into small units interconnected by links; users build information structures by creating units and they might access these units simultaneously; and users navigate by selecting links. Maurer[Maurer1990] defined hypertext as a graph of nodes and links, and software for its management and viewing. Knuth and Brussh[Knuth1990] defined hypertext in terms of its features. They have acknowledged that it is impossible to find a set of features other than browsing

and authoring which were embodied by all hypertexts.

Many hypertext systems have existed and new hypertext systems are appearing. The hypertext systems can be classified according to their applications area: macro literary systems; problem exploration tools; browsing systems; general hypertext technology; information presentation systems; and information collection and management tools.[Conklin1987, Carlson1990, Begoray1990] Another method of classifying hypertext is to consider the cognitive activities they support: reading; annotating; collaborating; and learning.[Carlson1990]

From an examination of the hypertext systems, it would appear that there is no hypertext system which fully supports the various activities needed in document creation: namely Discussion; Authoring; Annotation; Browsing and Searching. Furthermore, as already mentioned in Chapter 1, there are many varieties of hypertext systems and supported text models. To help understand the different functions supported by the hypertext systems, Section 2.1 provides a description of the low-level and the high-level functions while Section 2.3 provides details of how the hypertext systems currently being used to implement these functions and highlights their variations. One of the latest models of hypertext will be described in the Section 2.4.

2.2. Hypertext Functions

To help clarify the ambiguous nature of hypertext systems, Section 2.2.1 provides a general definition of the primitive functions and related terminology. Section 2.2.2 provides a classification of hypertext systems in terms of their high-level functionalities, and provides a general definition for these functions.

2.2.1. Primitive Functions

This Section provides some definitions that identify the basic hypertext objects and their interrelationship within hypertext systems.

2.2.1.1. Creating Nodes

Hypertext consists of a number of interconnected nodes. Each node may or may not contain one or more text blocks. In other words, the text might be attached to the node and it might be attached to the links. Each node might have a title, a type, or a name. Nodes appear on the screen in windows. A node might be larger than the window or it might be constrained to the size of the screen or window. In the first option, the hypertext systems must provide some mechanism for scrolling through the node's content.

Nodes are sometimes called something different such as *Frame* in KMS or *Card* in HyperCard. *Composite nodes* is a mechanism for aggregating related information in hypertext. By using this mechanism, several related nodes may be 'glued' together into composite nodes, and treated as a single node. The composite node has its own name, type, and identifier.

2.2.1.2. Creating Links

Linking is the essence of hypertext. Links are sometimes called relations. Each node may be linked to one or more nodes. The text might be attached to the link rather than be attached to the node. The links between nodes can be directed, bidirected, or undirected. The direction encodes whether the specified endpoint is to be considered a source of a link, a destination of a link, both a source and a desti-

nation, or neither a source nor a destination. The endpoint might be a single point or a text block. Links may be described by a type, a name, an identifier, or all three. The links between the nodes allow a user to move from node to node accessing the information the nodes contain. When a node is displayed, the user requires some mechanism for activating any links they select. Some mechanisms used by the different hypertext systems include a touch-screen, a mouse, menus, or key word searches.[Begoray1990]

2.2.1.3. Structures

The purpose of hypertext is to collect nodes about one or more topics and link them together. Such a collection of linked nodes may be considered as a graph which may be a hierarchical or non-hierarchical, or perhaps both. Structures other than a hierarchy can cause user disorientation regarding where they are and where to go next. To avoid this, some mechanism must be provided by the system for traversing the graph.

2.2.1.4. Annotating

A prerequisite for annotating text is that the text should be already created. The hypertext Annotation systems or those hypertext systems that support making annotations allow users to annotate others work. Computer annotation facilities may vary radically in the interface style. In one simple approach, the original document is in one file and all comments are in a separate file. In another approach each annotation is a separate file and may be directly linked to that part of the original document to which they most pertains. In viewing such annotations, windows are appropriate, and each comment appears in its own window with an explicit link

to another window that contains the relevant portion of the original document.

A relatively straightforward alternative to annotative windows involves allowing an annotator to make changes to the original copy. However, all changes are recorded so that either the original document or any of the changes can be recovered. The system records the activities of the annotator and separately maintains an unmolested version of the original document.[Rada1991a]

2.2.1.5. Reading

Reading implies the traditional sequential, line-by-line coverage of a document from page 1 to the end, in an unbroken stream. As with annotating, prerequisites for reading are:

1) text should be already created and
2) it should be possible to locate a point of interest, bring the text to the screen, and to start reading from that point.

However, on-line hypertext systems present texts non-linearly. Thus, the readers are required to specify what information that want to read and in what order. Browsing and searching help readers to find the required information. Browsing can be more or less focussed. It may range from open curiosity about any or all of the elements within a hypertext collection, to a constrained search for one particular element. Searching is understood in the relatively strict sense of starting with a specific question in mind and interacting with an information base to find the answer in a frailly straightforward manner[Duchastel1990]

2.2.2. High-Level Functions

Hypertext systems can be classified according to their functions. There are four broad activities supported by the developed hypertext systems: Discussion; Authoring; Annotation; and Browsing and searching. This Section provides a brief description for each one of these activities.

2.2.2.1. Discussion Systems

Discussion systems might include models for planning and problem-solving. The planning model would support the determination of document requirements and general goals,[Rada1991a] the setting up of an agenda, and the co-ordination of the whole authoring activities. Thus, the planning space contains the overall goal structure and plans for the writing process.[Streitz1989] The author has the option to stay with his/her original intentions or he/she is free to change and modify his/her initial decisions.[Haye1979]

The Problem-solving model might serve as the medium for exploring the capture of design history (the decisions, rejected options, and tradeoff analysis).[Conklin1987] Or it might serve as a conversation among the stakeholders (designers, authors, or discussants), in which they bring their respective expertise and viewpoints to the resolution of design problems.[Kunz1970] The generated arguments can be ordered and related to a specific problem based on their validity and relevance. The assessment is done by a type of quantitative voting.[Lowe1985] Thus, the discussion systems should be able to collect votes from users to support decision-making.

The decision-making is one of the most important issues in Discussion systems. If a conflict happened between users, the decision-making is the best solution for

such a conflict. Also, assigning a decision to a node will be useful in facilitating the reading process. If decisions are assigned to nodes, participants can go directly to those *issues* which are not yet resolved, and to continue the Discussion there.

Decision-making involves two or more persons elaborating on the nature of a problem, generating and evaluating potential solutions, and formulating strategies for their implementation. Each of these persons is characterized by his or her own perceptions, attitudes, motivations, and personality.[Desanctis1987]

Four types of group decision making can be observed:

1) a single decision maker within a collective decision environment,
2) non-cooperative decision making,
3) cooperative decision making,[Tung1984] and
4) consultation.

In the first type, a particular decision maker ultimately makes the decision and assumes responsibility for his/her direction. Other decision makers can either support or object to the decision. In the second type of group decision making, the decision maker plays the role of antagonist or disputant. Objection and competition are common forms of non-cooperative decision making. In a cooperative environment, the decision makers attempt to reach a common decision in a friendly and trusting manner, and share the responsibility. Consensus, negotiation, voting schemes, and recourse to a third party to resolve differences are examples of this type of group decision making. Finally, the consultation is similar to the first type of group decision making in that it has only one decision maker. The decision maker must consult the other participants for their opinions, arguments, etc, in a friendly and trusting manner. Ultimately, he will make the final decision, after taking into account the other participants opinions and attitudes.

Sometimes voting constitutes the end of the decision process, but in most decision situations votes serve the intermediate role of helping to identify areas where consensus is lacking.[Kraemer1988] When confidence measurement is introduced to weigh votes of individuals, greater interpretive information is available for assessing the meaning of the votes. Voting is predicated on the assumption that pluralities signify substantial collective confidence in a given choice. There are more than five methods of voting such as: 1) yes/no; 2) agree/disagree; 3) percentage; 4) multiple choice; and 5) rank order.[Liou1989]

2.2.2.2. Authoring Systems

Authoring can be divided into three phases: preauthoring, preparing a first draft, and reworking subsequent drafts. Each phase can involve four categories of processes: collecting, planning, translating, and annotating.[Kellogg1987] Hypertext authoring systems are generally intended to help users organize information for the writing task,[Jordan1989] analyze policy decision making,[Marshall1987] capture early design decisions,[Conklin1987] and computer-aided design.[Delisle1986]

Authoring is one of the major functions of hypertext. Systems primarily designed for authoring focus on tools that allow users to create node and links, edit them, and organize them into network structures and revise these structures and their content. Some examples of authoring tools are: TEXTNET[Trigg1986] for idea processing, WE[Smith1987] for document preparation, and Neptune[Delisle1986] for supporting the design and documentation of large scale software systems.

2.2.2.3. Annotation Systems

Annotation is one of the most important activities in authoring. Annotating concerns reading the evolving text, evaluating the text and plans, and correcting errors.[Kellogg1987] Annotation hypertext systems focus on recording ideas dynamically generated while reading text, including critique, explaining difficult passages, sorting user-produced mnemonic aids, and communicating with the library manager and/or other users.[Carlson1990]

Some examples of Annotation systems are:

1) Quilt[Fish1988] for supporting social roles and providing annotations and messaging among the users;
2) InterNote[Catlin1989] helps users making annotations on each other's documents; and
3) PREP[Neuwirth1990] for supporting a variety of co-authoring and commenting relationships for scholarly communication.

2.2.2.4. Browsing and Searching Systems

Browsing via link traversal is the most common method of retrieving information in a hypertext environment and it is the primary characteristic that distinguishes hypertext from a traditional database.[Knuth1990] Browsing and Searching are two different styles of access to hypertext. Browsing involves jumping from place to place and only reading small segments in each of those places. Searching occurs when a user knows the label for some information and wants only that specific information.[Rada1991a]

The textual nodes and links that are stored in databases can be browsed in different ways by using either the graphical browser and windows, windows and link

icons, or a combination of both.[Conklin1987] In hypertext systems which include a graphical browser and windows, the browser often provides a graphical structure to the nodes and their interconnecting links on the screen (see Figure 1). 'Clicking' on a node in the browser causes the text attached to the selected node to be displayed. In the second method that using windows and link icons, the data attached to the selected node will be displayed on the screen in a separate window. If that node is linked to one or more nodes, then the link icons will appear on that window. 'Clicking' on that link icon causes a new window to be created and filled with the text that is attached to the node to which the link icon points. In the method using a graphical browser and link icons, the content of a node can be displayed on the screen either by 'clicking' on a node in the graph or by 'clicking' on the link icons in a window.

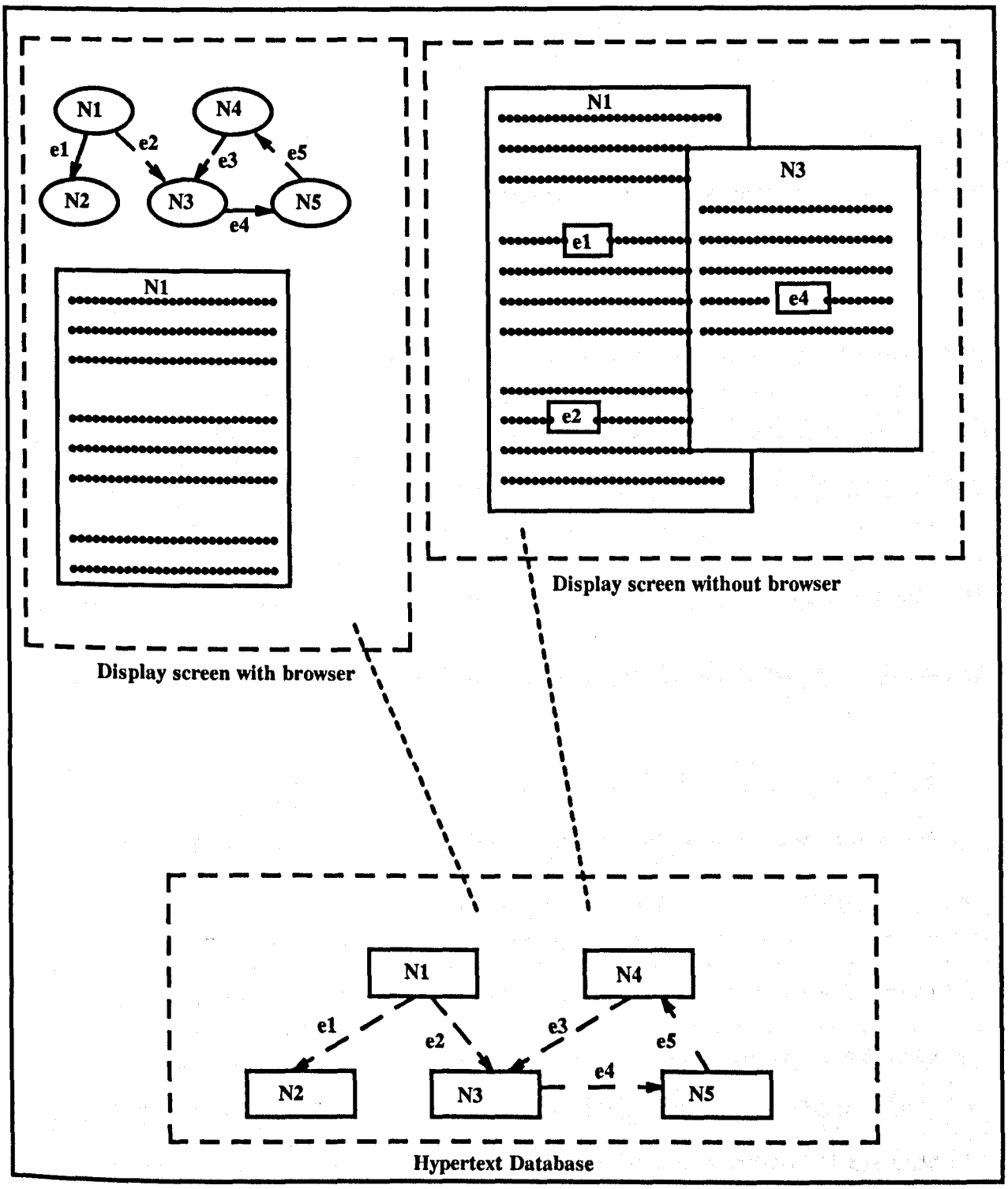


Figure 1 "Viewing nodes and links": The links and the nodes in the database, as in "Hypertext Database", can be viewed by: 1) using graphic views as in "Display screen with browser", 2) windows and link icons as in "Display screen without browser", or 3) a combination of both graphics and link icons.

2.3. Existing Hypertext Systems

Some hypertext systems have focused more on the development of the user interface issues, while others have focused on the storage issues. Table 1 identifies various functions of the different hypertext systems which have been implemented.

2.3.1. Emacs INFO

Emacs INFO[Stallman1981] is a powerful editing subsystem. It is heavily used in research facilities that run the UNIX operating system. It has a simpler set of standard commands, and its control input is done by single letters or short commands typed at the keyboard. It is primarily syntax hierarchical, but a user can jump to a different node in the hierarchy by typing the name of the destination node.

In Emacs-INFO a link to text that is an expansion of a particular topic may be presented on a menu, which consists of a list of node names. When a menu item is selected, the INFO program searches for the required node and presents its content on the screen. At the top of a node, call it node X, are labels that begin with either *"Prev:"*, *"Next:"*, or *"Up:"* (see Figure 2). Following the colon is the title of another node. *Prev:* points to a node meant to be sequentially previous to the node X. Conversely, *Next:* points to the node sequentially following X. The *Up:* label is followed by the name of the node that contains the menu entry for X. An option exists to jump to any node in the system by giving its node name. Additionally, the user may request via a "previous-in-time" command to see the node last visited. If the user had been at node Y before going to node X, then activating the "previous-in-time" option returns the user to node Y.

SYSTEMS

FUNCTIONS

	Emacs Info	EUCLID	gIBIS	Hyperties	IBIS	InterNote	KMS	Outline Processor	Quilt	SYNVIEW	WE	HERD	MUCH	Dexter
Annotation						X	X		X				X	
Discussion		X	X		X			X		X	X	X	X	
Authoring		X	X			X	X	X	X	X	X	X	X	
Syntax Hierarchy	X	X	X		X	X	X	X	X	X	X	X	X	X
Non-Hierarchy	X	X	X	X	X		X				X		X	
Discussion Types														
Annotation Types									X					
Link Types	X	X	X		X	X	X					X		
Node Types			X		X							X		
Graphical Browser			X						X		X			
String Search Browser	X							X				X	X	
Attributes			X						X			X	X	X
Composition													X	X
Anchoring														X

Table 1 "Hypertext Systems and their functions".

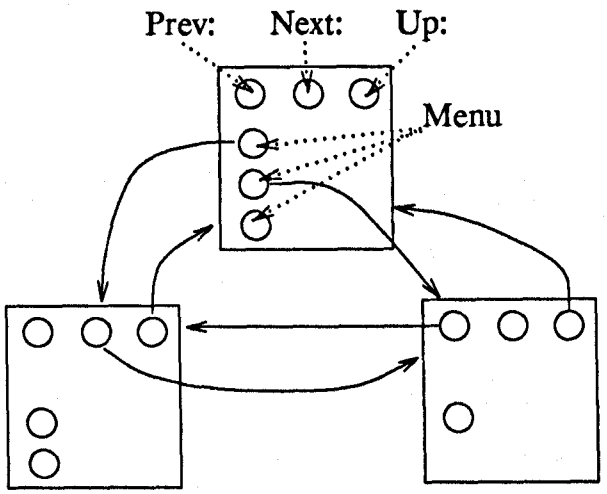


Figure 2 "Emacs INFO": The Emacs INFO structure is suggested by these 3 rectangles which represent nodes of text and links. The links are indicated by circles and solid arrows. The dotted arrows go from the name of a link type to the placement of that link in a node.

2.3.2. EUCLID

EUCLID[Smolensky1987] presents a hypertext system designed to support a specific task: development of reasoned discourse. EUCLID's environment provides a canvas on which the user may construct arguments and a palette of tools from which to choose building blocks for the arguments. The EUCLID computer screen is controlled by a constraint-based layout system and the palette is defined in the "Argument Representation Language" (ARL). EUCLID does not attempt automatic reasoning. While argumentation is the application of EUCLID, it could equally apply to developing papers or decision-making. The project aims to develop a powerful general framework in which hypertext systems may be defined. In constructing an argument, two kinds of knowledge are brought to bear: knowledge of the subject domain, and knowledge of the argumentation per se. A general purpose argumentation tool helps the user by virtue of its knowledge of argument structure, not argument content. "The DIVIDE" separates content information at the bottom

from structure information at the top. An example of assertions below the Divide is:

Zero interest rates lead to bull markets.

An example of statement above the Divide is:

Claim C_1 supports claim C_2.

The meaning of information below the divide is difficult abstractly to characterize, while the information above the divide is hypothesized to be similar across people and domains. The ARL statement corresponding to "Claim C_1 supports Claim C_2" uses the formal term *claims* and relates two such terms with the formal predicate *supports*. Other ARL terms include *argument* and *author*. Other ARL predicates include *asserts* and *main-point*. Since the content of each *claim* is not expressed formally but as natural language, ARL is a semi-formal language.

The nature of EUCLID applications requires many small nodes with as many as several hundred nodes displayed at once. This is too large for people to conveniently represent without the help of a data manager. EUCLID allows multiple perspectives of the same argument consisting of arbitrary granularities and layouts. For instance, the support for a given argument can be easily displayed.

2.3.3. gIBIS

Jeff Conklin and Michael Begman have developed the IBIS method to explore Issue-based methodologies for capture of design rationale, and to support computer mediated team work.[Conklin1988] One of the radical extensions to IBIS in gIBIS is the adding of some more node and link types to those which already exist such as:

1) *surrogate* type of node that contains non-IBIS material;

2) *Positions* can be *generalized* or *specialized* by other *Positions*, and likewise with *Arguments*; and

3) an *escape* mechanism for participants who cannot find a way to express a thought within the IBIS.

However, these node and link types, in both IBIS and gIBIS, are fixed and the user cannot change them by modifying them or by adding new node and link types.

The Discussion system gIBIS uses color to indicate the type of IBIS nodes and links, and relational database to facilitate building and browsing through typed IBIS network. gIBIS uses four windows interface:

1) a graphical browser to provide a visual presentation of the IBIS graph structure;

2) a node index window to provide an ordered, hierarchical view of the nodes;

3) a control panel which is composed of a set of buttons which extend the tool's functionality beyond simple node and link creation; and

4) an inspection window in which the attributes and contents of nodes and links can be viewed.

The gIBIS system uses special browsers to display global and local views of graphs, and the user can 'zoom' and 'pan' across a graph.

The resolution of an issue can be represented and displayed by combining with the aggregation in Issue-Position-Argument nodes, and then one can indicates that the Issue is resolved. This is done by changing the value of the resolved field to TRUE and indicating which of the Issue's Positions was the one selected as the resolution. In gIBIS, there is no specific node type for goals and requirements, nor is there particular support for making a decision (or reaching consensus) among the various positions of an issue. There is also no way of indicating that such a decision

has been made.

Finally, gIBIS supports both kind of syntax structures— hierarchical and non-hierarchical. Filtering the secondary links from a canonical IBIS subnet results in a syntax hierarchy, and this syntax hierarchy is the basis for the index window's structured linearization. The primary link is the first link which connects a node into the network. The secondary links are the subsequent links which connect a node to others in the network.

2.3.4. Hyperties

The development of Hyperties[Shneiderman1986] began in 1983 at the University of Maryland. It was developed for browsing instructional database and to serve as a platform for investigation hypertext interfaces.

In Hyperties the basic units are short articles (50-1000 words), which are interconnected by any number of links. The links are highlighted words or phrases in the article text. The user activates the links by touching them with a finger– on a touch-sensitive screen or using the *Arrow-jump keys* to jump to them. The *Arrow-jump keys* allow the cursor to jump to the closest highlighted string in the direction pressed.

Each article is divided into three fields: a title, a definition (which briefly describes the article), and the body of the article. When a user selects a Hyperties link, the destination article's title and short description are shown in a separate window. Confirming the selection causes the source article's display to be replaced by the destination article (see Figure 3). An article about a topic may be one or more screens long. As users traverse articles, Hyperties keeps the path and allows reversal. Users can also select articles from an index.

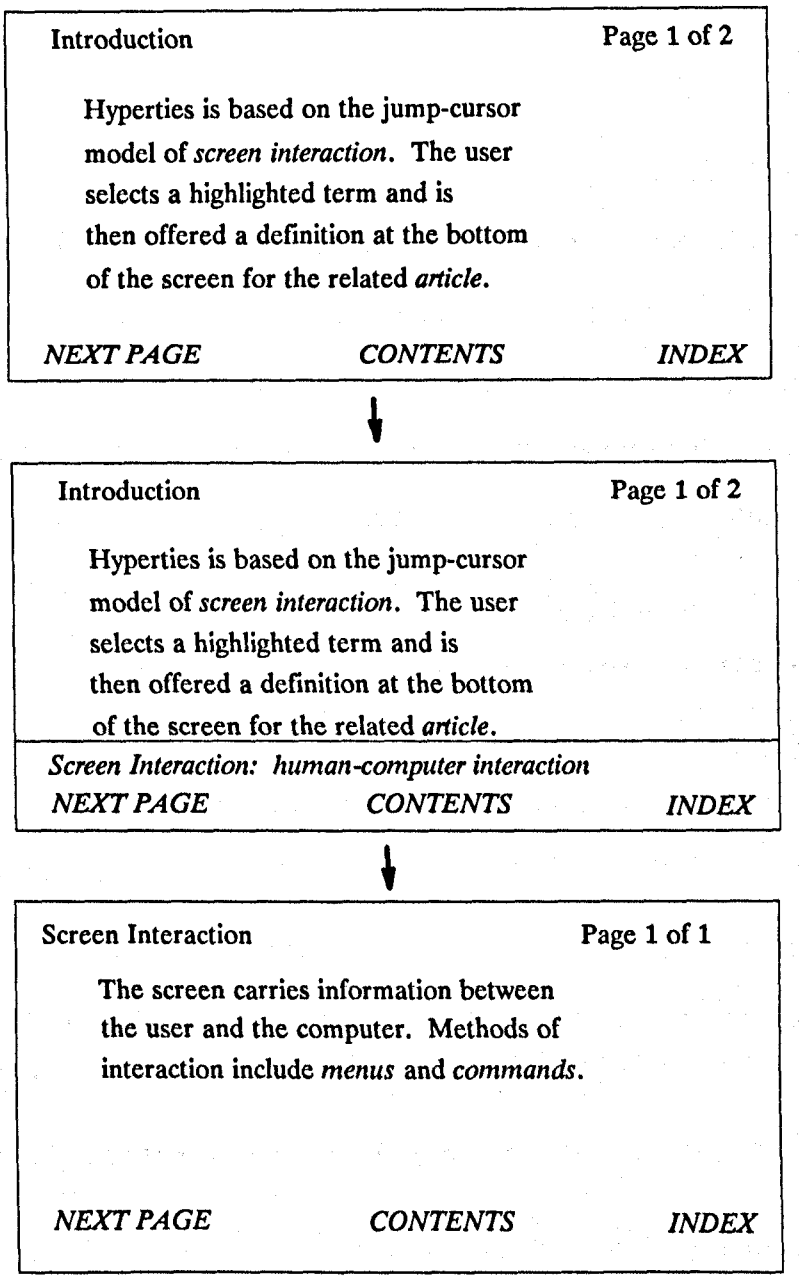


Figure 3 "Hyperties Example": In the top screen the the italicized (highlighted) terms are 'screen interaction', 'article', 'NEXT PAGE', 'CONTENTS', and 'INDEX'. If the user selects 'screen interaction', then the next screen differs from the first only by the addition of a brief note near the bottom of the screen about 'screen interaction'. If the user now activates this brief note, then the next screen is the article with that term as its title.

2.3.5. IBIS

Werner Kunz developed Issue-Based Information Systems (IBIS) to handle systems analysis in the face of 'wicked problems'.[Kunzl1970] Kunz describes wicked problems as problems which cannot be solved by the traditional systems analysis approach – define the problem, collect data, analyze the data, and construct the solution. IBIS guides the identification, structuring and settling of issues raised by problem solving groups, and provides information pertinent to the Discussion. Elements of the IBIS model are topics, *Issues*, questions of fact, Positions, Arguments, and model problems.

The IBIS model uses a fixed set of node and link types. It uses three types of nodes; *Issues, Positions,* and *Arguments*; and nine types of relationships to link the nodes – *responds-to, questions, suggests, supports, objects-to, specializes, generalizes, refers-to,* and *replaces*.

IBIS supports both hierarchical and non-hierarchical structures in conjunction with each other. It does not provide methods to convert non-hierarchical structures into hierarchical structures. Furthermore, it does not distinguish between the structure on the display and the structure in the storage.

An *issue* is resolved by selecting one of the *positions* that responds to it as being the best answer. There is, however, no specific way of registering that an *issue* has been resolved by agreement upon some *position*, nor is there a stopping rule other than for considerations that are external to the problem. The goals of the Discussion are for each of the participants to try to understand the whole problem better, to exchange information and to argue his or her viewpoint, ideas, concerns, in order to persuade others of one's own point of view.

2.3.6. InterNote

The InterNote is one of the hypertext systems which is designed mainly to support Annotation. InterNote helps groups of annotators comment on each other's annotations and satisfies a set of requirements.[Catlin1989] Some of these requirements are:

1) annotations can be created with any available editing tools, including those for creating text, graphics, and animation;
2) annotations may be added to annotations;
3) any number of annotators may simultaneously add annotations to a given document;
4) the author may easily incorporate an annotation into the document. The author may also merge or sort annotations; and
5) multiple interfaces are available for viewing annotations.

For instance, the user may see simultaneously the source document and the annotation. The user may also see other users' annotations.

For creating an annotation, the user makes a selection in a document and selects the 'Create Annotation' command. An annotation consists of two frames: the *Incorporation Frame* which fills the top portion of the Note window and the *Commentary Frame* which fills the bottom portion of the Note window (see Figure 4). When a Note is first created, the *Incorporation Frame* initially contains an exact copy of the material from the article which has been elected to annotate. The annotator may edit the contents of the *Incorporation Frame* and, in the *Commentary Frame* may include general suggestions for revising the document. Authors can incorporate the contents of the *Incorporation Frame* automatically and they must use 'copy' and 'paste' to incorporate the contents of the *Commentary Frame*.

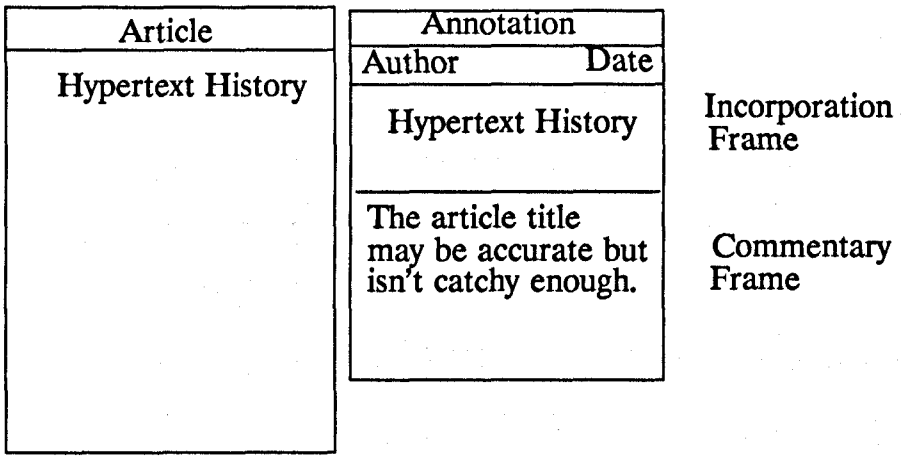


Figure 4 "InterNote Screen": Depiction of InterNote Annotation structure. Original text is on the left. The annotation is on the right.

In InterNote, links, called the local anchor and the remote anchor, are made between document objects. To traverse a link, a user selects a link and picks the "Follow" command. To transfer data across this link, a user selects the link and issues the *Push* or *Pull* command (see Figure 5). The *Push* command copies the contents of the local anchor and pastes it at the other end of the link, replacing the contents of the remote link anchor. *Pull* has the opposite effect and copies the content of the remote anchor over the local anchor.

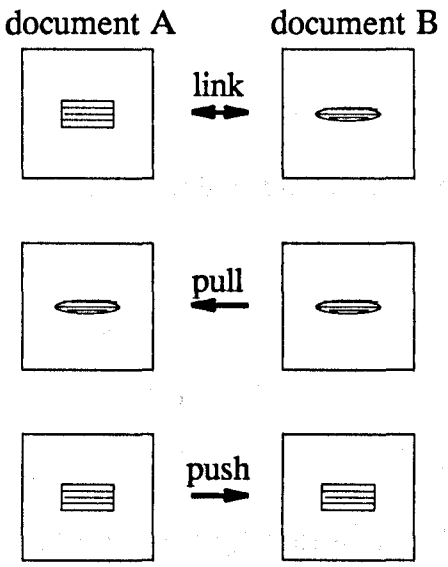


Figure 5 "Push or Pull": In the top diagram, the rectangle in document A is linked to the ellipse in document B. Document A contains the 'local anchor' and a pull will move the ellipse from document B to document A. Conversely, a push will move the rectangle from document A to document B.

In viewing such annotations, windows are appropriate, and each comment appears in its own window with an explicit link to another window that contains the relevant portion of the original document. Internote has different strategies in linking one annotation to another annotation or to the base document. In addition to using the links for navigation, they can be used to transfer data. On the other hand, InterNote does not allow users to modify the existing Annotation node and link types, nor to add new node and link types.

InterNote supports two major types of annotations: suggested changes in the same data type as the original document, and textual commentary for notes comments, non-specific suggestions, and justification. Finally, both syntax hierarchy and non-hierarchy are supported by this system. The annotator can either make a navigational link from or to an existing annotation or annotate an existing annotation.

2.3.7. KMS

KMS (Knowledge Management System)[Akscyn1988] is a commercial version of ZOG system developed at Carnegie Mellon University from 1972 to 1985. KMS is designed to help organizations manage their knowledge. At Knowledge Systems KMS used for different aspects including document production, product design, and software engineering.

A KMS database consists of a set of interlinked, screen sized workspaces called *frames*; what is called in other hypertext systems *nodes*. There is only one type of frame in KMS. Frames is a unit of combination of text, graphics, and image items, each of which may be linked to another frame. While any kind of link can be used, KMS particularly supports syntax hierarchical links. Users interact with the database by navigating from frame to frame, manipulating the contents of frames, and creating new frames. Tools also exist for inheriting characteristics from one frame to another and for importing material from other sources, such as text files, into frames.

Each frame may have six different functional parts (see Figure 6) includes:

1) The *frame title* describes the frame topic;
2) The *frame name* is a unique identifier for the frame, as a page number is a unique identifier for a page in a book;
3) The *frame body* expands on the topic of the frame;
4) The *Tree buttons* link to frames at the next lower level of the hierarchy;
5) The *Command buttons* initiate actions, such as exiting KMS; and
6) The *Annotation buttons* begin with an '@' and provide notes or cross-references.

The workstation screen is normally split into two windows, each of which shows a frame.

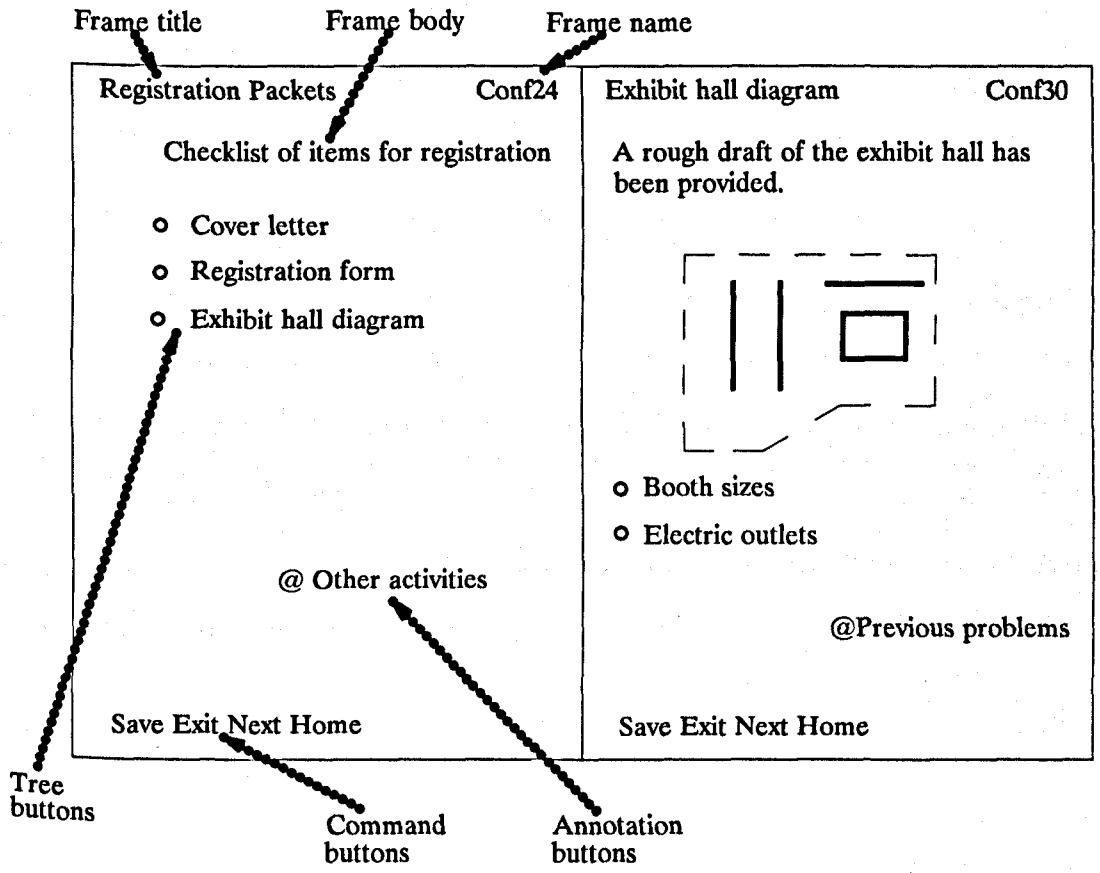


Figure 6 "KMS Screen": An illustration of a prototypical screen in KMS.[Akscyn1988] Tree buttons are preceded by a hollow circle, while annotation buttons are preceded by an '@' sign.

In addition to specifying within a frame the children of that frame, KMS provides only one other view of the contents–a listing of the entire hierarchy of frame titles. There is no graphical browser. The outline of the database is not presented on the screen as a two-dimensional graph. In early versions of KMS graphical views were available, but studies showed that the graphical views were rarely used.

The source for a KMS link can be any region of text in a frame. The destination of a link can be another entire frame. The destination of a link can also be computer program in that when the link is activated, the program is executed. A frame is considered to be a small enough unit that the whole frame, rather than any

part of it, can sensibly serve as the link destination. In KMS there are two types of links: *tree* and *annotation*. Tree links point to lower-level frames in a hierarchy, such as a Chapter of a book. Annotations point to peripheral material, such as comments. These two link types distinguish between structural relationships and purely associative relationships. Links may be more than one line of text and allow the author to provide substantial semantic information about the link.

The KMS user interface is based on the *direct manipulation* paradigm. Users navigate from frame to frame by pointing the mouse cursor at an item linked to another frame and clicking the mouse button. Editing and browsing are done in the same mode. By exploiting contextual constraints, over 90 per cent of the user's interaction requires a single point-and-click. The average time per meaningful operation is less than it would be for pull-down menus.

2.3.8. Outline Processor

Outline Processor[Hershey1985] is a word processing tool which is specialized for organizing blocks of text and processing outlines, in that its main commands deal with movement among, creation of, and modification of outline entities. Because text is a critical part of the final product, outline processor incorporates some text editors and do some text formatting, so that the user can use the same tool to go from outline to finished document. One of the most powerful features of the outline processing is the ability to collapse or expand an outline at any headline level. The users can view the top *n* levels, or they can open up just those entities that are useful to the idea that they are working on. Each entity in the outline can have a textual body of any length associated with it, and the users can make this text appear or disappear with a single keystroke.

Most outline processors are personal computer tools, and they are growing in popularity. Their range of capabilities varies widely. ThinkTank and Executive Writer/Executive Filer offer slightly different solutions to the problem of managing textual information. ThinkTank presents a method of organizing ideas into a structure for writing. Executive Filer has a flexible retrieval system for large volumes of information, and Executive Writer is a powerful word processor. Most outline processors do not support inter-entity references, and only a few others provide windows for nodes. None of them provide explicit "mousable" link icons.

2.3.9. Quilt

Quilt[Fish1988] is a computer-based tool for collaborative writing , which provides annotation facilities to support information sharing among the collaborators on a document. In addition, sets of social roles are used to provide views of a document tailored to individual collaborators of the document based upon their position in a permission syntax hierarchy.

In Quilt, three social roles are defined– author, commenter, and reader– as well as six objects– base document, suggested revision, directed messages, private comment, public comment, and history. The set of editing operations include creation, modification, deletion, reading, browsing, searching, copying, moving and attaching comments and messages. Quilt uses information social roles to determine the types of activities that are allowed on the types of objects by various types of collaborators. Quilt defines three types of collaboration types:

1) exclusive, in which only the author of a Section in the base document can modify it;

2) shared, in which any author can modify any Section; and

3) editor, in which a designated editor can modify any Section and other authors may only make suggested revisions.

Quilt supports syntax hierarchical structures, in which the base document can be annotated and annotations can be annotated recursively. As Quilt supports different types of annotation, but it does not support node and link types. Finally, Quilt supports browsing. When browsing through a Quilt document, the user is shown the structure of annotations associated with an object, either using an outline or a graphical overview of the links between the nodes.

2.3.10. SYNVIEW

The SYNVIEW system was introduced by David Lowe.[Lowe1985] It allows users to maintain a Discussion and to indicate their degree of support for each item in the Discussion. The method is similar in concept to IBIS, but the direction is different. The participants have to assess previous postings as to their validity, in addition to posting items. For example, if a participant posting an *issue* and one of the others responds with an *argument*, he or she should assign a grade to his response. If the argument makes a good point but is not really a direct response to that *issue*, a "10,-10" grade (where 10 is a high validity rating and -10 is a low relevance rating) might be assigned for that argument.

Users can request certain types of changes to the structure or content of the Discussion and submit these changes for assessment by the other participants. Depending upon the significance and the relevance of each item of information to each topic, the users can choose the depth to which they wish to examine these structures for the purposes at hand. Where SYNVIEW gives its users some guidance as to which depth they can choose, IBIS and gIBIS leave that as an open

option to its users to choose whichever they prefer.

The goal of the Discussion in SYNVIEW is to collect the best available evidence on each *issue* which is at hand. Another more radical goal is to organize the content of many evidences on any given *issue* into a single syntax hierarchical structure. However, there is no way of indicating that such evidence has been reached.

2.3.11. WE

WE[Smith1987] is a hypertext writing environment that can be used to create both electronic and printed documents based on a specific cognitive model of the writing activity. The reading activity is described as the process of taking the linear stream of text. The text that is comprehended is a hierarchy integrated into a network of long-term memory. The writing activity involves the iterative and recursive application of the cognitive modes (prewriting, organizing, and writing and editing) to a series of intermediate products.

WE contains two major view windows, one graphical and one hierarchical, and many commands for moving and structuring material. Using the prewriting mode, the writer can create nodes and title them. The writer can also position a representation of the node anywhere in the network window. The tree mode helps the user build a single, integrated hierarchical structure for the document. Nodes created in either the non-hierarchy or the tree are just titles representing ideas. The editor mode provides access to a standard text editor by selecting a node in either the non-hierarchy or tree modes and selecting the edit option. In the text mode, the document is presented in linear form.

2.4. A Model of Hypertext

Dexter[Halasz1990] has provided one of the more recent hypertext models. The goal of the model was to provide a principled basis for comparing systems as well as for developing interchange standards. This model has three layers: a runtime layer, a storage layer, and a within-component layer, as illustrated in Figure 7. The storage layer concerned with the network of nodes and links. The run time layer concerned with the mechanisms that support the user's interaction with hypertext. The within component layer covers the content and structures within hypertext nodes.

The storage layer is composed of a finite set of a hierarchy components and two functions, a resolver function and an accesser function which are jointly responsible for retrieving components. The components are interconnected by relational links. A component is either an atom (node), link, or a composite component. A node contains generic data such as chunk of text, graphics, images, or animations. This model is different from some other models in that the text is attached to the node not to the link. Composite components are constructed out of other components. Each component may have arbitrarily many attribute-value pairs. A link is divided into parts, each of which points to a node (see Figure 7).

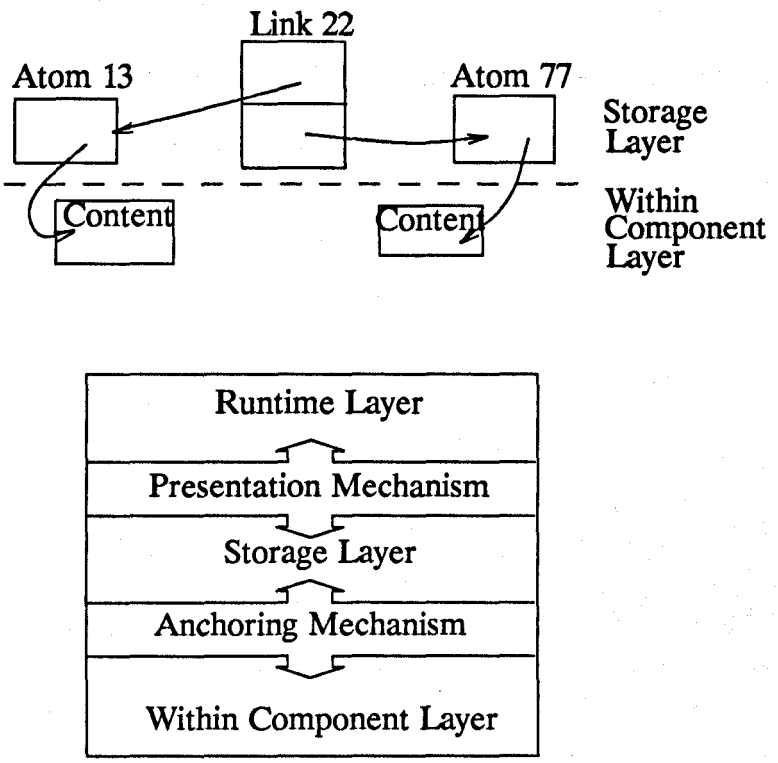


Figure 7 "Dexter Model": Depiction of link connecting two nodes in upper part of Figure. The components in the storage layer contain various attribute-value pairs. The lower part of the Figure shows the layers and their interfacing mechanisms for the Dexter model.

Between the storage and runtime layers is a "presentation mechanism" and between the "storage" and "within-component" layers is an "anchoring mechanism". Anchors are a mechanism which provide the function of linking between two documents or between two spans of characters in different documents while maintaining a clean separation between the storage and within component layers. The presentation mechanism presents the hypertext to the user. The way in which a component is presented to the user can be a function of the specific hypertext tool that is doing the presentation. It can also be a property of the component itself and/or of the access path taken to that component.

The network structure can be accessed, viewed, and manipulated through the runtime layer. This is done by using the "instantiation" of a component which is the fundamental concept in the runtime layer. A copy of the component is cached in

the instantiation, the user views and/or edits this instantiation, and the altered cache is then written back to the storage layer. Important attribute-value pairs for group-text applications include authors and time/date of creation.

CHAPTER 3

Discussion in Hypertext Systems

3.1. Introduction

Discussion as a function of hypertext systems is supported in a number of systems, such as SYNVIEW, gIBIS, IBIS, WE, and Outline Processors (see Chapter 2). SYNVIEW, gIBIS, and IBIS implement issue-based information, and can be used to discuss requirements specification documents. Research on requirements specification is important from both the artificial intelligence and software engineering perspective.[Charney1987] Many of the problems of software system development can be traced to poor understanding or specification of what the system is supposed to do.[Yeh1980] The earliest phase of requirements acquisition involves a "skull session" whose goal is to achieve consensus among end users about what they want and need from the software system.[Rada1991a]

The major phases of software requirements specification are 1) Discussion, 2) requirements, 3) design, 4) production, 5) testing, and 6) maintenance [Rada1990a] (see Figure 8). This sequence emphasizes the need to meet the overall defined requirement; the need for each step to meet the specifications of the previous step; and the importance of reworking an earlier stage in the light of a later stage. The Discussion phase was typically neglected in software engineering literature but is now appreciated as a crucial phase. The key personnel in a company commissioning the development of a large system must agree on the purpose of the system and what the features of the system are to be.

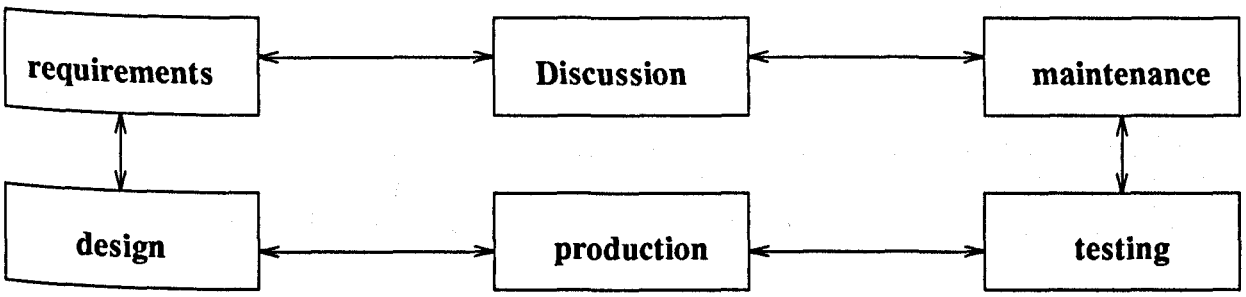


Figure 8 "Software life cycle": The software life cycle often goes from Discussion to requirements to design to production to testing to maintenance and then again to Discussion as a new system is envisaged.

From an examination of the Discussion hypertext systems, it would appear that producing a requirements' specification documents report at the end of the Discussion is not one of the goals of some systems. Additionally, there is no particular method of representing and displaying the resolution, nor is there particular support for making a decision between the various nodes in the network which represent the Discussion. Furthermore, there are some variations in what these systems support, such as:

1) the kind of structure;

2) the node and link types; and

3) the goal of the Discussion.

Such variations led to the development of a system called HERD. This system addresses a number of design issues, particularly issues concerning:

1) the kind of network that will best represent the Discussion (hierarchical, non-hierarchical, or both);

2) node and link types issues such as to support or not to support typing, or which node and link types will best represent Discussion;

3) how the resolution can be represented and displayed;

4) how the discussant can be assisted in linking new information to the existing structure; and

5) how the system can assist the reader who wants to browse text on a specific topic, or wants to find one particular piece of information.

A Discussion might be described as a search through hierarchy of knowledge states, with the ultimate goal being to provide a solution to the problem. A knowledge state is the information researchers know or postulate when they are at a particular stage in their search for a solution. Each state reached in the hierarchy contains a little more knowledge about the issue than that reached previously and thus Discussion knowledge states are incremental.

Another way of describing Discussion might be in terms of planning in order to reach a goal. The goal can be decomposed into sub-goals, which in turn are defined in terms of their sub-goals, until some basic level is reached at which further decomposition is not needed. The final goal is often reached by establishing and achieving a series of sub-goals that represent partial solutions.

Additionally, the Discussion might be described as issue-based information. A Discussion may begin with someone posting an *Issue* and another person may post a *Position* in response to this *Issue*. An *issue* may represent a problem, concern, or question needing Discussion. A *position* is a statement or assertion which resolves an *issue*. The goal of the Discussion might be to resolve an issue. The resolution of the issue should enable each of the participants to understand the whole problem better, exchange viewpoints, and persuade others to adopt alternative points of view.

The focus in this thesis will be on this last aspect (i.e issue-based information Discussion). As described earlier, the Hierarchy of a Discussion consists of syntax and semantics. In the hierarchy, a node is a text block which has a label. Both the

text block and the label might be created separately. Each text block should have a label but the converse is not necessarily true (i.e a label might be created without creating a text block). A label might be an *Issue, Position, Argument* or *Comment.* The semantics of the hierarchy is the set of full meaning relationships which connect the nodes with each other such as a *Position responds* to an *Issue* or an *Argument supports* a *Position*

This Chapter provides an introductory description of the HERD tool which is given in Section 3.2. A description of three case studies to evaluate the system and to explore some issues about Discussion are described in Section 3.3. In order to explore more issues about Discussion, one experiment was performed, as described in Section 3.4.

3.2. HERD Tool

The Hypertext Environment for Reasoned Discourse (HERD) is an issue-based information system, whose goal is to provide an environment that effectively supports informal Discussion. HERD users may include designers, researchers, authors, or any intellectual laborers who are analyzing information or processing ideas. A basic subset of the features of issue-based information systems was incorporated in the HERD system.

In this Section, the basic features for the HERD system will be described and illustrated with simple examples. The HERD system is implemented under Unix on Hewlett Packard 9000-300 computers. The system makes heavy use of Xwindows, Emacs, Unix Documenters Workbench, and Xdvi. Xwindows provides a quick means of developing a mouse-driven user interface. Xdvi is a screen driver which allows typeset text and graphics to be viewed on the users' computer screen. In

describing the system, a description of the data model will be discussed first (Section 3.2.1), then the HERD functions: Discussion; Authoring; and Browsing will be described (Section 3.2.2).

3.2.1. Data Model

Since the syntax hierarchical structures facilitates reading process and diminishes the disorientation (see Section 1.2), HERD supports the hierarchical structures and the semantics of non-hierarchical structures. The storage is composed of a syntax hierarchy of nodes. Nodes contain the chunks of text, graphics, or images. Nodes are treated as generic containers of data, and there is no differentiation between text components and graphics components. There is no restriction on the number of text blocks that a node might contain. Nodes are described by types and unique identifiers (i.e each node has a unique identifier and a type). Links are described by types only.

HERD supports a fixed set of node and link types, and the user has no option to add new types nor to modify the current types. The current prototype supports three types of links: *responds-to, generated-by*, and *other* (see Figure 9), and supports three types of nodes: *Issue, Position*, and *Argument*. *Positions* can be linked to an *issue* by the *responds-to* link. An *issue* is generated by a *position* and may represent a problem, concern, or a question that needs Discussion. Each *issue* is a root for a subtree and can have one or more positions linked to it. A *position* is a statement or an assertion that resolves an *issue*, and can have one or more arguments linked to it. An *issue* can be linked to a *position* by the *generated-by* link. *Arguments* are linked to any kind of nodes with the *other* link.

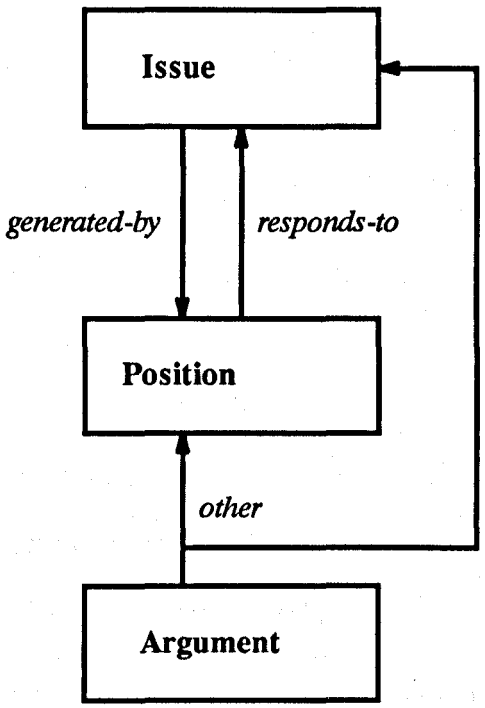


Figure 9 "HERD's nodes and links": Each of the three node types can be linked with a subset of the three available link types.

3.2.2. High-Level Functionalities

This Section provides a brief description for the different functions supported by this system: Discussion; Authoring; and Browsing.

3.2.2.1. Discussion Function

The Discussion using HERD might serve as a planning and problem-solving function. Discussion starts at an initial root node being posted, and proceeds in the following fashion: users who wish to respond to the text that is already in the tree construct a new node by filling in a form to write their response and linking this to the existing node to which it relates most closely.

Discussion using HERD forms a tree, whose nodes are pieces of text (com-

ments). The structural relationship between the nodes signifies the semantic relationship between the comments. In a typical session with HERD, a user might post an *Issue*, such as "What kind of computers our department should by?". A *Position* can be created by that user or another user, such as "We should buy Unix workstations". An *Argument* can be created, such as "A lot of good software are being developed for Unix workstations". Another discussant browsing the information might post another *Position* in responds to the first *Issue*, such as "Our department should by personal computers". The last *Position* might be supported by an *Argument* posted by a discussant, such as "Buying low-cost personal computers would allow everyone to have their own computers".

Initially the scope of the problem may not be well understood and various issues and opinions may be vague, confused and changeable, and so users of HERD may create 'proto-nodes'. A proto-node is an arbitrary text block which is not explicitly linked to any other text block. Each proto-node is tagged with the name of the person who wrote it and the time it was written. Thus users can search through the database of proto-nodes through the author and time tags.

Once an issue (problem or question) has been isolated as the topic for a Discussion, that Discussion is constrained in the HERD system to be part of a hierarchical structure. This hierarchical organization may help focus attention and facilitate decision-making.

3.2.2.2. Authoring Function

Authoring is another major function of hypertext systems. It focuses on the information creating and linking aspects. The authoring function allows a user or a group of users to create and edit the storage of nodes. This function provides the Emacs editor and the Unix Documenters Workbench in which the text can be written. Emacs is an extensive text editor that includes a LISP-like programming language for text and document manipulation. The Unix Documenters Workbench is a document formatting language in which proportional text, figures, tables and graphics can be described.

The users enter the HERD system using menus. All the basic features for HERD are selected by pressing and holding the right button, using the mouse to highlight the desired choice, and releasing the button. To create a new node, users select from the menu the "Create-Node". This causes a file (the name of the file is "fill_form" concatenated to the user-id of the user) to be displayed in a separate window. As shown in Figure 10, this file contains four fields:

1) Node-Parent: the content of this field is the word "Root" if this is the first node (users will be asked for the project name before this node can be linked). Otherwise it is a number appearing on a node on the graph;
2) Edge-type: the content of this field should be one of the edge types mentioned in Section 3.2.1;
3) Title: an abstraction for what the users want to write in the "Text" field; and
4) Text: users use this field to write their text in response to the contents of the parent node.

Would you please fill this form; you can exit after you

have finished by typing: CTRL x CTRL s then CTRL x CTRL c

Remember the following

(1) Node-parent must be a number existing in the graph, or the word 'Root' if it is the first node.

(2) Edge-type must be one of the following words or letters: 'R Responds-to G Generated-by other'.

(3) If the node you want to create is an Issue then, Edge-type should be 'G' or 'Generated-by'.

(4) If the node you want to create is a Position then, Edge-type should be 'R" or 'Responds-to'.

(5) If the node you want to create is an Argument then, Edge-type should be 'other'.

(6) Leave a space after each occurrence of ':' in the fields.

OTHERWISE YOU WILL GET AN ERROR MESSAGE

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

Node-parent: 1.1.1
Edge-type: Responds-to
Title: Abstract
Text:
IBIS, gIBIS, and SYNVIEW are hypertext tools implementing issue-based information systems, and they might be used to discuss the requirements specification documents. Some of these tools support only hierarchical structures (for example, SYNVIEW), others (such as

Figure 10 "HERD's Fill-form screen": This screen gives the user some information regarding the rules of connecting nodes and links. At the bottom of the screen, four fields need to be filled according to the information at the top.

To link a node, users select from a menu the "Link-Node" option. If there is no error in the contents of the fields the node will be automatically linked into the existing network of nodes. An error might be that the content of the field "Node-parent" does not exist, or the content of the field "Edge-type" does not follow the set of legal rhetorical moves in HERD (see Figure 9). In the case of such an error occurring, an error message will be displayed on a separate window on the user's

screen which explains:

1) what the error was and

2) what the user has to do.

Linking a node to the existing network of nodes causes a unique identifier to be assigned to that node. Each identifier will be used as a reference to a specific node, so it can be referred to whenever the user wants to perform an operation on that node.

3.2.2.3. Browsing Function

The browser allows users to traverse the storage of nodes created by the authoring function. It provides different facilities to help the users browse the storage. There are currently three facilities provided: Search; Index; and Author and Date. They can be selected from a menu. The search facility allows a string search of the storage. When a user selects the option "Display-Title-Inf", this causes the titles of all nodes to be displayed on a separate window (see Figure 11). The node title window provides an ordered, hierarchical view of the nodes in the current HERD structure. The *Issues, Positions*, and *Arguments* are given unique sequence numbers. The nodes can be selected by searching on these unique identifiers.

******* Title-Information-File *******

NODE NAME	TITLE
I [1]	What do we want to discuss?
P[1.1]	HERD Requirements Guidelines
I [1.1.1]	What main Sections?
P[1.1.1.1]	Requirements for Scacchi paper
A[1.1.1.1.1]	Comments on Scacchi's form
P[1.1.1.2]	Layout from KBRA
A[1.1.1.2.1]	Comment on KBRA's req. form
P[1.1.1.3]	65CS Req. Layout
A[1.1.1.3.1]	Comment on 65CS req. form
I [1.1.1.3.1.1]	What extensions required?
A[1.1.1.1.3.1.2]	Extension of 65CS req form
A[1.1.1.3.1.2.1]	Comment on comment on 65cs
A[1.1.1.3.1.2.2]	Difference with 1.1.1.5
A[1.1.1.3.1.2.3]	Appealing for finalizing
A[1.1.1.3.1.2.3.1]	Reminder for finalization
A[1.1.1.3.2]	Nomination of 65cs approach
A[1.1.1.3.2.1]	It is no conclusion
A[1.1.1.3.2.2]	No nomination yet

Figure 11 "HERD's Title-Information": This screen shows the title for each node. The nodes are ordered according to what each node most pertains.

The HERD users also have the option of checking the names of the authors of the nodes, and the nodes' times of creation by selecting the option 'Display-Author-Date-Inf'. They can ascertain which issue a participant was addressing and they can quickly learn whether another participant has responded to their *arguments, positions*, and *issues* as well as learn who made responses and when the responses were made (see Figure 12).

********* Author-Date-Information-File *********

Note:

1st = First name or initial, Last = Last name, or surname.
D = Day, M = Month, d = day as a number, and
hh:mm:ss = hours:minutes:seconds.

NODE							
NAME	Author		DATE				
	1st	Last	D	M	d	hh:mm:ss	Year
I [1]	Mahmoud	Mhashi	Sat	May	20	14:19:40	1989
P[1.1]	Akmal	Zeb	Sat	May	20	14:36:07	1989
I [1.1.1]	Akmal	Zeb	Sat	May	20	14:58:40	1989
P[1.1.1.1]	Kenny		Sat	May	20	18:04:40	1989
A[1.1.1.1.1]	Kenny		Sun	May	21	17:21:53	1989
P[1.1.1.2]	Akmal	Zeb	Sat	May	20	18:46:54	1989
A[1.1.1.2.1]	Kenny		Mon	May	22	11:21:34	1989
P[1.1.1.3]	Akmal	Zeb	Sat	May	20	19:12:32	1989
A[1.1.1.3.1]	Kenny		Mon	May	22	11:28:12	1989
I [1.1.1.3.1.1]	Akmal	Zeb	Tue	May	23	13:25:27	1989
A[1.1.1.3.1.2]	Judith	Barlow	Sun	May	21	18:05:20	1989

Figure 12 "HERD's Author-Date-Information": This screen shows when each node had been created and who created which node.

Users can easily find out who has added related *arguments* and *positions*. They may choose to view a quick summary of added nodes (see Figure 13) or view the full text of the nodes by selecting the "Display-Text" option (see Figure 14). Selecting the "Print" option instead of the "Display" option causes the contents of the files "Author-Date-Inf", "Text", and "Title-Inf" to be sent to a printer, depending on the file that the user select.

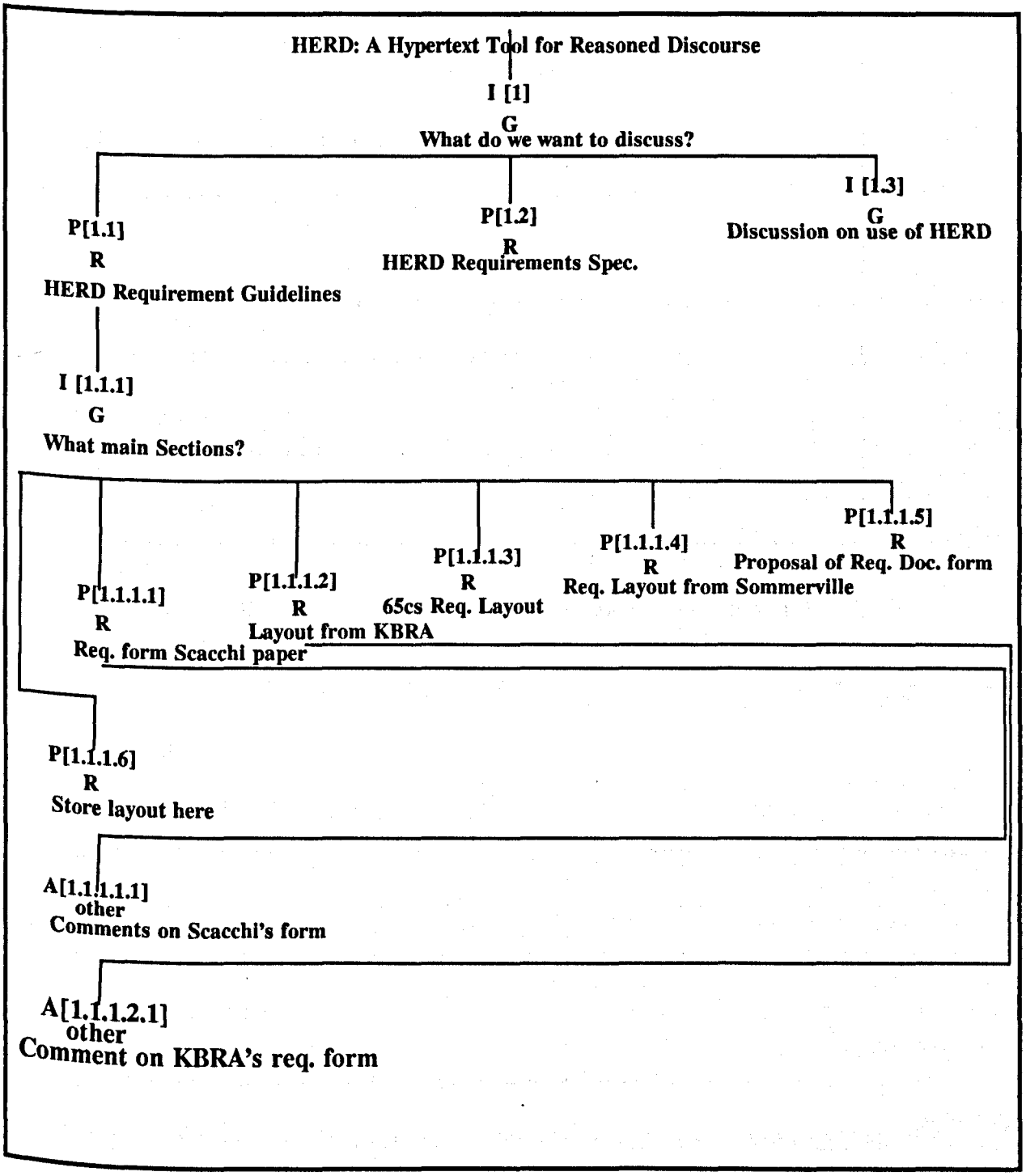


Figure 13 "HERD's Discourse representation structure": This screen shows how the Discussion can be presented as a graph.

```
********** TEXT FILE **********
**********  Beginning of node I[1]  **********
Node-name: I[1]
Text: What should be the Discussion on HERD?
**********  End of node I[1]  **********
**********  Beginning of node I[1.1]  **********
Node-name: P[1.1]
Text: This part of the Discussion will be used to
discuss the requirement guidelines for HERD.
The format for the final document will be discussed.
**********  End of node P[1.1]  **********
**********  Beginning of node I[1.1.1]  **********
Node-name: I[1.1.1]
Text: What should be the layout for the main Sections
for the Discussion on the requirement guidelines for HERD?
**********  End of node I[1.1.1]  **********
```

Figure 14 "HERD's text screen": This screen shows how the text of the nodes is presented on the screen. The nodes are ordered in the same way as the title information ordered.

3.3. Case studies

Three case studies were performed. In the first case study (Section 3.3.1), HERD was used. The second case study (Section 3.3.2) was performed manually. In addition to discussing different software requirement guidelines, a number of issues were tested, such as:

1) what kind of structure will best represent a Discussion;

2) how is it useful to restrict the users with a set of node and link types; and

3) what is the role of decomposition in the Discussion of software requirements.

In the third case study (Section 3.3.3), HERD was tested as an Annotation system. The goal of this case study was to explore some similarities between Discussion and Annotation. Furthermore, the goal was to evaluate the use of HERD by comparing the results from this case study with the results from the first case study (Section

3.3.1) which was performed to discuss the software requirements.

3.3.1. Case Study 1: Requirements Discussion

The requirements document establishes the boundaries on the solution space for the development of useful software systems. It is used as the basis for:

1) the communication among users, experts, analysts, and designers;

2) supporting design validation; and

3) controlling the operations and evolution of the system it specifies.

A good requirements document should have the following characteristics:

1) *clear, unambiguous*, and *understandable*: since the requirements document must serve such a variety of people, every requirement should have only one possible interpretation;

2) *complete*: the requirements document must be complete in that all constraints and assumptions are explicitly stated;

3) *traceable*: one should be able to begin with a requirement and trace it forward through the design and into the implementation to see that the product will satisfy that requirement;

4) *modifiable*: sometimes, clients can change their minds, or some requirements cannot possibly be met. It is often possible to renegotiate a requirement, yet still be satisfactory to the client. These changes must be documented; and

5) *consistent*: it should not be contradictory.[Steward1987]

Five subjects were asked to discuss guidelines for a requirements' document using the HERD system. The five subjects included three computer science graduate students and two computer science academic staff, all aged between 28 and 38

years. All five had workstations in their offices, and were familiar with their use. The participants had experience with the Emacs editor and worked daily on the machines used in the exercise. The subjects were given background to the problem before doing the exercise. They had participated in seven face-to-face meetings to discuss the strengths and the weaknesses of different models of guidelines for a requirements' document.

At the seventh face-to-face meeting the decision was taken to move the Discussion to the HERD system. The face-to-face meetings had not led to documented conclusions, and the HERD system was seen as a tool which might facilitate the achievement of written conclusions. All the subjects were asked to participate in this Discussion by creating some nodes in their own time. After two months, participants were no longer adding comments to the Discussion. A meeting was then held, and the group agreed that the Discussion had reached a natural end, and that a satisfactory conclusion about a good requirements document outline was evident. After this, the participants were asked to write their evaluation of HERD.

The structure of the Discussion in terms of node types and their connection to other node types is shown in Figure 15. It can be clearly seen in that figure that the majority of nodes are of the type "Argument". A tally of the link types shows that of the 57 links, 40 are of the type "Other" and go from "Argument" largely to itself (see Figure 16). During the two months of Discussion on HERD 59 nodes were created that corresponded to 20 pages of text. The contents of these nodes ranged from one sentence to four paragraphs. Also, four subjects selected the option "Create-Proto-Node" and there was no facility to register how many times this option was selected by each subject.

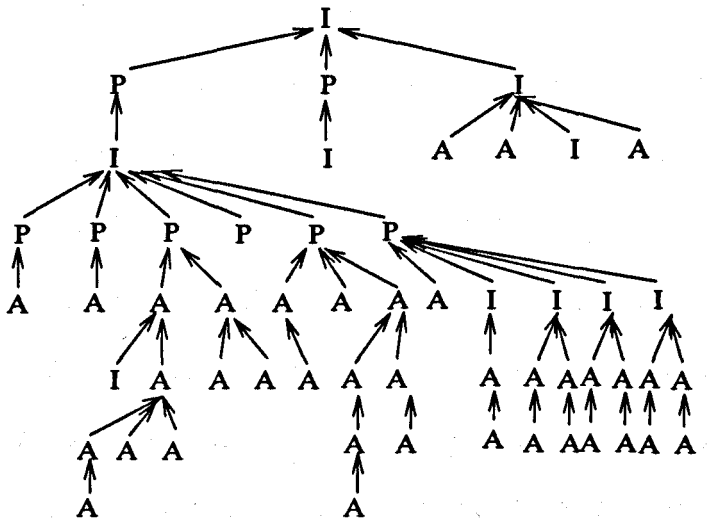


Figure 15 "Diagram of Discussion".

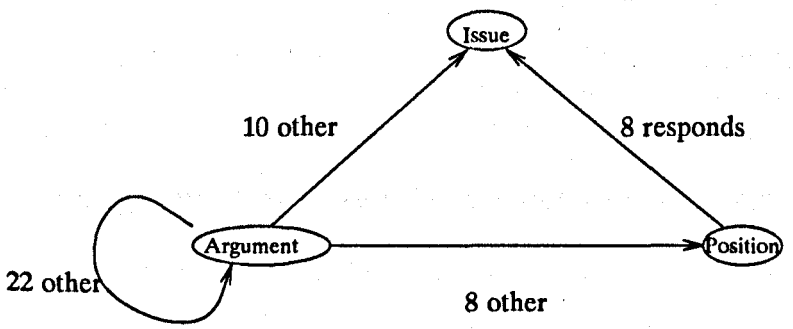


Figure 16 "Predominant Links": The four most frequently occurring node-link-node triples are indicated in this figure. The number followed by a link type and a link specifies the frequency of occurrence.

A counter was maintained by the program to show how often each of the different views of the Discussion was selected and revealed:

1) the text view was selected 74 times;

2) the title view was selected 82 times; and

3) and the author-date view was selected 116 times.

At the end of the exercise, the subjects were asked by the facilitator, in a face-to-face meeting, to write their evaluation of HERD by creating one or more nodes using HERD. The four subjects who completed the evaluation of HERD all said

that they found the tool helpful. Comments included:

> "HERD's hierarchical structure was found to be effective for supporting the Discussion in that it forced the users to structure their ideas and organize their thoughts"

Two subjects noted that they sometimes got confused and that a restructuring of the information was needed. Since there was only one link type between *Argument* and *Position* available to the users, it was not clear whether a user was responding in support of an issue or was offering some criticism. Also, decision-making was not explicitly supported.

The data of the Discussion was analyzed and the semantics of hierarchy was tested and assessed by two experts who know IBIS and the requirements. Generally, in the first 32 nodes that were created, the subjects were using the non-decomposition strategy, and they were using the decomposition strategy in the next 27 nodes. The total number of links was 58 (46 hierarchical links and 12 non-hierarchical links). Eleven of the non-hierarchical links were in the non-decomposition part. In contrast there was just one non-hierarchical link in the decomposition part. Six nodes were used to conclude an intermediate summary and to restructure the information in order to carry on the Discussion. All of the 6 nodes were in the non-decomposition part. In the non-decomposition part also, one node was completely reused. One issue was posted and never responded to. A node was created which was not necessary to the Discussion. Finally, some text in some nodes was not related. Thus, it can be seen that using decomposition diminishes the non-hierarchical links and the disorientation problem. Furthermore, it leads to a faster conclusion in that all the nodes which are discussed in the decomposition part are needed. In contrast, in the non-decomposition part, some nodes were discussed when they were not needed.

Ten nodes were created by one of the most important subjects (a Doctor) and eight of these nodes were responded to by some of the less eminent subjects (post-graduate). Both the date and the time for all these nodes (i.e. the nodes created by the Doctor and the nodes created by the postgraduates in response to the Doctor's nodes) were analyzed. All the nodes created by the Doctor responded to first by the postgraduates. The perceived importance of the Doctor is demonstrated by the high priority given by other subjects to the nodes he/she created.

3.3.2. Case Study 2: Decomposition

In the last exercise, the advantages for using decomposition were tested within one group. In this case study, an attempted was made to test whether or not the decomposition would be preferable than the non-decomposition to a set of groups of users.

In a software engineering course for undergraduates at the University of Liverpool, the students were taught about the software life cycle, the requirement guidelines, and the requirement documents, by a professor of computer science. As part of the course exam, the 68 students were divided into 23 groups of 2 to 5 students each. Each group was asked to develop a requirements' specification for a computer game and to begin by discussing the requirements. Each group had to document both the Discussion and the requirements. Based on the results from the HERD exercise, one might hypothesize that decomposing the problem into its sub-problems first and then resolving these sub-problems will be regarded by the groups as preferable to resolving the sub-problems as they arise.

The data collected from the students at the end of the class was analyzed. Four groups discussed the problem of what to put into the requirements by resolving

sub-problems as they arose. Five groups documented the Discussion as a conversation, and did not explicitly identify problems or resolutions. Finally, in 14 groups (or 61% of the groups) the students first decomposed the problem into its sub-problems and then resolved these sub-problems.

3.3.3. Case Study 3: HERD as an Annotation System

To evaluate the use of a Discussion system for Annotation, results from two case studies were compared. One case study involved using the HERD system for Annotation. The other case study was performed to test what kind of network would best represent the Discussion (Section 3.3.1). This latter case study was conducted in collaboration by Eevi E. Beck and others;[Mhashi1991] it involved using HERD as an Annotation system to annotate approximately 600 paragraphs of an on-line hypertext book. The hypothesis was that the process of Annotation and Discussion are sufficiently similar for a support Discussion system for some aspects such as the node and link types to serve equally well in the support of Annotation.

Seven volunteers from a research group in computing agreed to participate in the case study. The participants were all familiar with the operating system and electronic mail; however, their experience with the HERD system ranged from extensive to none at all. The volunteers met regularly over a period of 2-3 months to revise preliminary drafts for a book.

The facilitator created an outline on HERD for a Discussion about the book. One Node was created for each Chapter, and participants were expected to make comments about the book linked to the appropriate node outline. After that stage, they started posting nodes onto the structure which was set by the facilitator.

The group progressed with HERD slowly. Participants had expected that using

a system which allowed asynchronous Discussion would eliminate the need for face-to-face meetings. Contrary to this original expectation, however, meetings were still felt to be necessary. A lot of time was spent on these meetings. A reorganization of the work then emerged from the group itself. The participants agreed to work in a more synchronized way (such as reading the same Chapter at the same time), even if they didn't have time to complete their task. By this point, dissatisfaction with the HERD system, relating to speed, and in particular the time taken to link a node had been expressed.

The output from the group in terms of feedback on the book was low – 55 nodes were created. In addition to the top-level nodes corresponding to each Chapter, a separate node on administration had been created. Eleven of the 55 nodes were on administrative topics. The fact that annotations were about the book but the book was not connected to HERD was a fundamental source of difficulty. It was only partially alleviated by the breakdown of the top-level issue into sub-issues based on the Chapter headings of the book.

Four of the participants had read parts of the book and had commented beyond those comments expressed in the node tree, without entering these into HERD. The long time taken to create and link a node, and the general reluctance to express thoughts in writing, were given as reasons. The participants in the case study also had some difficulties making the necessary distinction between node types in HERD, i.e., classifying their comment as an *argument, position* or *issue.*

The subjects were interviewed separately at the end of the case study and they were asked about the use of HERD as an Annotation system. The four subjects who participate in both HERD as a Discussion and as an Annotation system made comments that included,

"HERD was not as suitable for Annotation as it was for Discussion."

However, creating the Annotation in a form of hierarchy facilitated the revising process (see Figure 17).

In the node tree arising from the Discussion, the maximum horizontal spread (i.e., the maximum number of children of a node) was 6 steps. The maximum vertical distance (i.e., distance between a parent and its furthermost child) was 7 levels. On the other hand, for Annotation, the maximum horizontal spread was 9 steps, while the maximum vertical distance was 4 levels. In the Discussion, the number of nodes with at least one child at least four levels further down the tree was 42 nodes, while for Annotation it was only 4 nodes. Thus the tree structure for the Discussion was deep and narrow, while for Annotation it was broad and shallow.

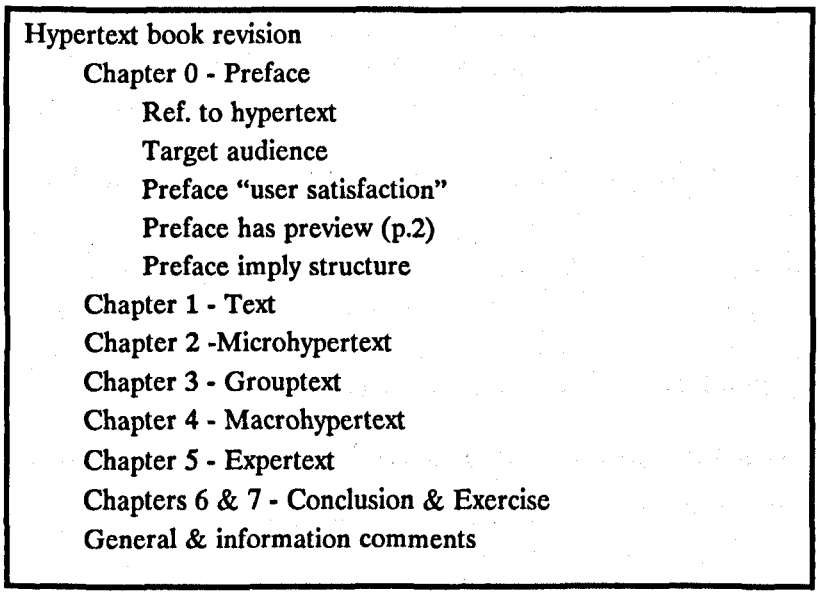


Figure 17 Annotation outline using HERD": Part of the Annotation outline from a case study which was performed using the Discussion system HERD.

3.4. HERD Experiment

In the first case study (Section 3.3.1), the semantics of hierarchy was tested and it was found to be preferable to the users than the non-hierarchy. The four subjects who wrote their evaluation about the hierarchy (both syntax and semantics) were also in favor of using the hierarchy. However, there was not an option to use the syntax of non-hierarchy. In this experiment, the syntax of both hierarchy and non-hierarchy was tested.

The subjects who participated in this experiment were four of the subjects who had participated the HERD exercise. They were given some mechanisms to allow them to 'chain' the current tree structure according to their needs (i.e to use hierarchical and non-hierarchical structures). The current tree structure is the one which was produced by doing the HERD's exercise.

The tree structure was drawn on paper. Each node was named and titled, and each link was labeled. Each subject was given a paper copy of the tree structure and a letter explaining what they should do. The letter explained that the subject should feel free to restructure the Discussion manually, so as to make it easy to read, browse, and to facilitate producing the conclusion that they are looking for. The restructuring could be done by adding/deleting one or more links and modifying any one of the link types (labels) using one of the current set link types or a new link type. All the subjects had access to the text of the Discussion which was stored in the computer. Each subject was asked to restructure the tree independently from the others.

The data was comparatively evaluated as follows: There were 11 non-hierarchical links added to the tree structure which had 58 hierarchical links. One link was deleted and one link type had not been used (*other*), and 12 new link types

had been added. A link is considered as a non-hierarchical link if it does not satisfy both the syntax and the semantics of the hierarchy definition. The link types that were added to the current tree structure were: *summarizes, comments, decision-made, supports, suggests, criticizes, clarifies, objects, replaces, contains, extends*, and *answers*. The relation between the node types and the link types after users' modifications can be seen in Figure 18.

The contents of the nodes which were connected by the nonhierarchical links were analyzed. Three nodes which had 6 non-hierarchical links were created as a summary for the other few nodes, and they were those most likely to be related to more than the parent node. Two nodes were created, each of them responded to two nodes. Such nodes should be connected to the two nodes to which they responded. One non-hierarchical link represented the difference between the contents of two nodes. Two nodes were created, each one of them contained more than two paragraphs, and they are most likely to be related to more than the parent node. Therefore, all the reasons which cause the non-hierarchical links can not be eliminated.

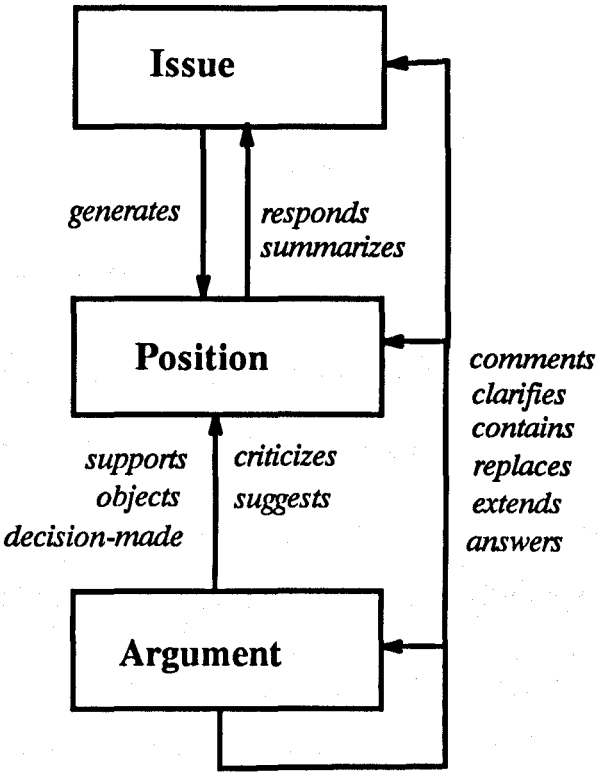


Figure 18 "Suggested links": The relation between the node and the link types after users' modifications.

CHAPTER 4

Annotation in Hypertext Systems

4.1. Introduction

The Annotation function is supported by many hypertext systems (see Chapter 2). Many issues are manipulated in different ways by these hypertext systems, such as:

1) How can collaborators share information and supervise each other?

2) How can collaborators review (commenting, questioning, and criticizing) documents?

3) Should the annotations be placed in the documents or in a separate record?

4) What Annotation node and link types should there be?

5) What kind of structures and outlines should there be?

Many of these issues are tackled by the literature[Leland1988, Catlin1989] but other are not. Such issues led to the development of a system called MUCH.

Before starting to build the system, a set of questions about Annotation was encountered, such as:

1) Should the Annotation node and link types be completely supported by the system, by the user, or by both?

2) What are the precedence factors in presenting the annotations to the readers, who are the authors and annotators, or in performing some operations on the annotations?

An author is a person whose name will appear on the document and who has the

responsibility regarding the content (text) of the document. An annotator is a person who reads a document and intends to contribute to that document or to the system; but his name will not appear on the document as a co-author. However, his/her name might appear at the end or beginning of the document (i.e Acknowledgement Section).

People working together to create documents ranging from small information networks to entire on-line books is a common occurrence in a wide range of disciplines including scientific research. A document might be a software engineering requirement, software design, information for use in the classroom, instruction material, a scientific paper, or a book. In some scientific fields sixty five percent of articles are explicitly co-authored, even when only one author's name appears on the final version of the document.Fish1988

The annotative collaboration process is started by first producing a draft of a document. Every portion of the document may be open to Annotation by any member of the group, or there may be restrictions on which parts each member can annotate. The role of Annotation is to bring forward the views of reviewers. These reviewers may include the author(s). Through their annotations, they assess the appropriateness of the current version of the document with respect to its intended message and audience.Hahn1989 Annotation can be defined as a text block which has a label or is itself a label pointing to the Triple (document, Discussion, or Annotation) that is to be interpreted by authors as a guide to modifying the document, by the system designers as a guide to modifying the system, or is to be left as histories of comments for readers (all authors can be readers but not vice versa). Some of the annotation labels might be considered as a name, a unique identifier, and a node type such as "check-spelling'. The annotations text blocks can be connected to the Triple via links. These links might have labels and types.

The goal of this Chapter is to explore and analyze some issues concerning the role of hierarchies in Annotation. The next Section provides a description of MUCH system. A case study in which MUCH used as a Discussion system is described in Section 4.3. Suggested Annotation node types are described in Section 4.4. Two experiments were then performed (Section 4.5) to test the usefulness of these annotation node types and to explore further issues regarding Annotation. The first experiment concerned document organization and the second experiment was concerned with the document content.

4.2. MUCH System

The *MUCH* system supports document "authoring", "accessing", and "annotating". An overview will be given in Section 4.2.1. The data model will be described in Section 4.2.2, followed by a description for the above functions and the implementation issues in Section 4.2.3.

4.2.1. Overview

The MUCH system has been developed to support the various phases of writing a document: *exploring, organizing,* and *encoding.* In the exploration phase knowledge is acquired, brainstorming occurs, and unstructured notes are made. Next, the unstructured notes are organized into an outline. In the encoding phase the prose for the final document is written.[Rada1990a]

MUCH is being developed on a network of graphical workstations using public domain software tools including Unix, Xwindows, Emacs, Unix Document Workbench, and Xdvi. MUCH has four main functions: 1) exploring, 2) accessing, 3) writing, and 4) Annotation and Discussion. A five window screen is used as a

common user interface. All the functions have some variations on the usage of windows for their own purposes. The system also provides a key menu within the whole screen to indicate the meaning of function keys under different functions. A skeleton of the entry screen is shown in Figure 19.

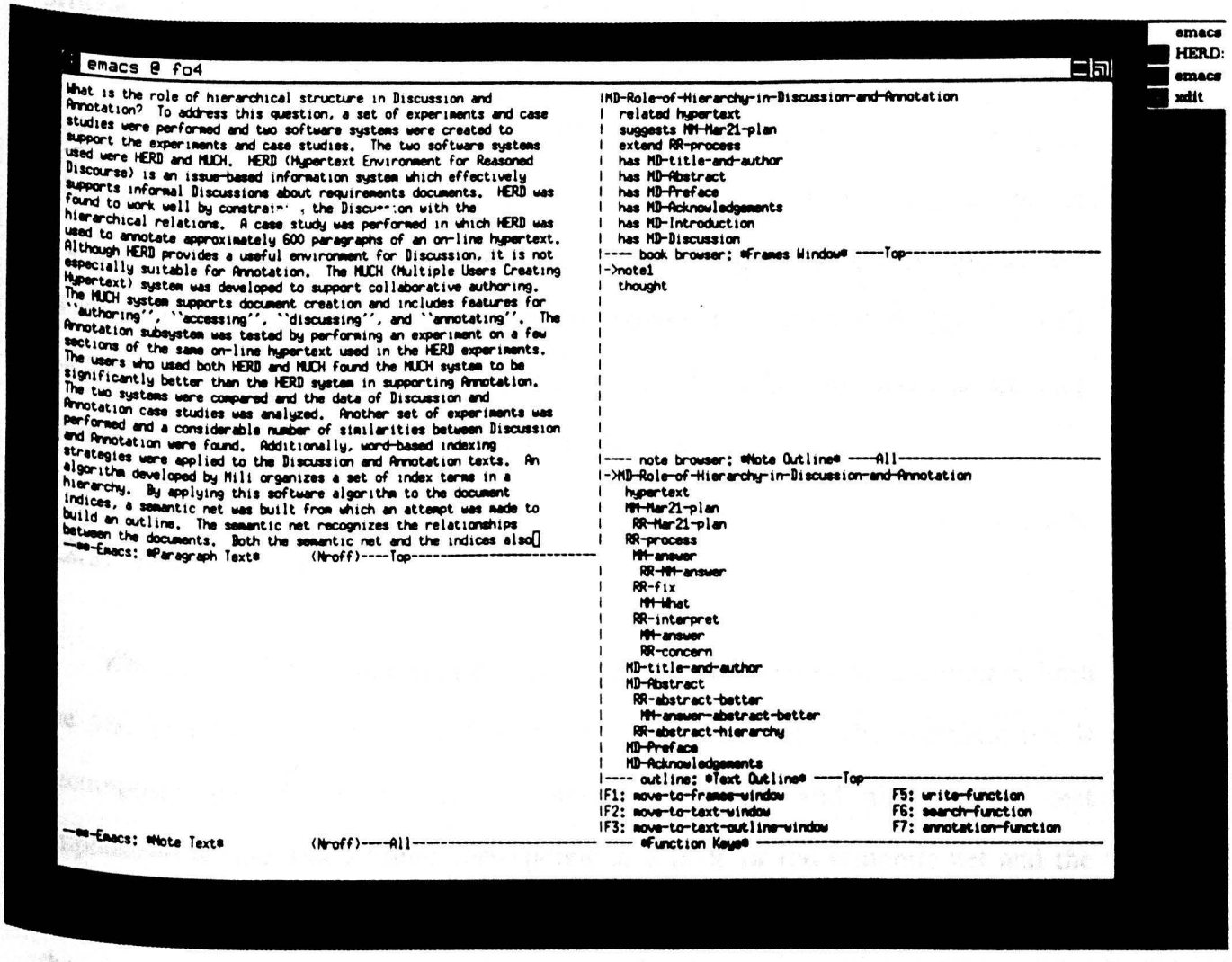


Figure 19 "First MUCH screen": The first MUCH screen contains six labeled subwindows. From top to bottom on the right side: the 'Frames Window' shows one frame of the document; the 'Note Outline' shows a list of links and nodes for notes; the 'Text Outline' shows the outline of the document; and the 'Function Keys' gives a menu of function keys and their actions. On the left side are two Emacs subwindows: 'Paragraph Text' displays a text block of the document, and 'Note Text' shows a text block of a note. When the user points to a term on the right half of the Emacs window and hits carriage return, then the associated text appears on the left half.

The MUCH system is a homogeneous distributed collaborative authoring system, that has been developed on HP-9000/300 workstations. Emacs and Xwindows constitute the front end while Ingres serves as the back end of the system. The user interface is controlled by an emacs-lisp program. This program makes calls to a C program which runs on the file server along with Ingres. The C program contains embedded SQL commands that are used to access, store, and update data kept by Ingres. The Lisp program sends requests to the C program. The C program translates these requests into SQL queries which are sent to Ingres. Ingres executes the queries and returns any required data to the C program which, in turn, sends it to the Lisp program. The Lisp program manipulates the data and displays it on the screen. In the next Section a description of the MUCH's data model is provided, followed by a description for the three main functions.

4.2.2. Data Model

With MUCH, multiple authors can collaboratively create text, annotate both the text and the annotations, and generate a semantic net. The semantic net is decomposed into frames which are internally coherent and made up of text emphasized terms. Each frame corresponds to a node of the semantic net and the links emanating from it and each frame is represented as a list whose first element is the name of the frame. The remainder of the frame consists of sublists with two basic components: *link* and *target-frame*. The *link* labels an edge of the semantic net. The *target-frame* is the node of the semantic net (or frame) to which the edge points. The directionality of the links is defined by the author by specifying a source and destination frame for each link. The MUCH system has blocks of text associated with the edges of the semantic net. A node in the semantic net can be linked to more than one node. Two nodes connected with a link form a link object. The link

object in the semantic net should be unique.

There is no restriction on the number of lines that a text block might contain, and the collaborators can create notes as well as annotations. Notes are neither connected to the semantic net nor to each other. The author creates annotations by creating nodes and linking them to frames in the semantic net. These annotations have predefined names in order to distinguish them from the frames in the semantic net. The text of annotations is attached to the node, not to the link. Annotations on annotations are allowed by creating an annotation node and linking it to the existing annotation nodes. The links which connect annotations to frames (within the semantic net) or to other annotations have a predefined directionality. That is, the newly created annotation is always the source of the link. The destination of the link can be either a frame or an existing annotation. The structure of the Annotation is different from the structure of the document. This occurred because the Annotation function was not clear, at this early stage, for anything more than making comments on a text block.

The MUCH system is based on a relational database model. The database holds the frames and links of the semantic net, the text blocks, the annotations attached to frames, notes, and information about the authors and creation dates of text blocks. Text blocks are stored in tables, one tuple per line. The database model of the system consists of 5 entities: *text block*, *note*, *link*, *frame*, and *Annotation*. An entity/relationship model of the system can be seen in Figure 20.

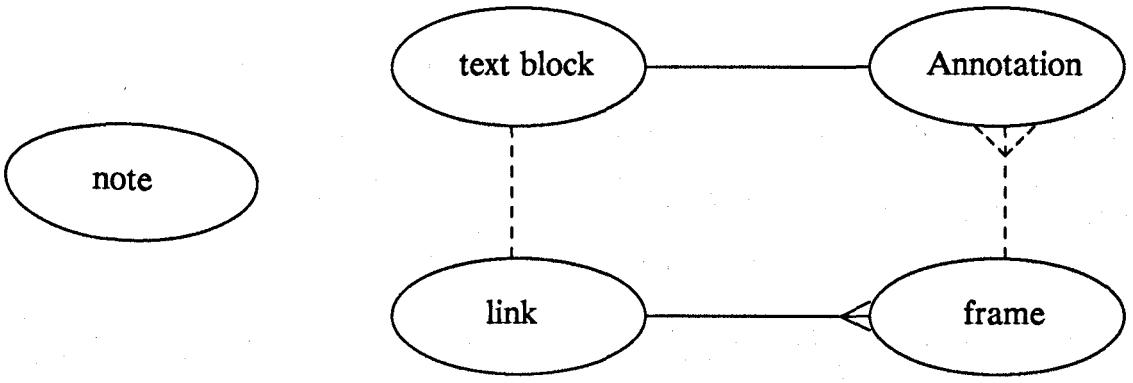


Figure 20 "MUCH entity/relationship": The MUCH entity/relationship model. The solid straight lines indicate a 1:1 (one-to-one) relationship. The delta notation indicates a 1:n (one-to-many) relationship. The dashed straight lines imply that occurrences of adjacent entities may exist without taking part in the relationship.

The above entities have been further refined through normalization and their internal representation consists of tables in 3NF (Third Normal Form). These tables along with their attributes are listed in Table 2.

Table names	Attributes
Text_Inf	link_name, author, date, index_terms
Text_Block	link_name, text_line, line_order
Link	link_name, author, source_frame
Link_Date	link_name, date, destination_frame
Frame_Author	frame_name, author
Frame_Date	frame_name, date
Note	note_name, author, date
Note_Text	note_name, text_line, line_order
Annot_Text	frame_name, annot_name, text_line, line_order
Annot_Date	frame_name, annot_name, date
Annot_Author	frame_name, annot_name, author
Annot_Frame_Link	frame_name, annot_name, annot_link_name
Annot_Link	frame_name, source_annot, destination_annot, annot_link_name

Table 2: MUCH entities and their attributes.

The above tables have common attribute names. Some of these attributes have different interpretations, depending on the entity. The link_name attribute has the format "source_frame - link - destination_frame", it is unique and indicates the link

where the text block is located. In the Annotation tables annot_link_name is merely the name of the link. The author attribute is the author's login name and it has the same interpretation within all the tables. The date attribute indicates the date that an author created a text block, a link, a frame, a note, or an Annotation. The index_terms attribute consists of a sequence of index terms associated with a text block. The text_line attribute contains a line of a text block, note, or Annotation. The line_order indicates the order of appearance of each line of text within the text block, note, or Annotation. Source_frame, destination_frame, and frame_name represent the name of a frame in the semantic net. The note_name is a unique identifier for a specific note. Finally, annot_name indicates a predefined name for an annotation.

4.2.3. System Functions

In this Section a brief description will be given of the functions which are related to the Annotation and Discussion. In particular, a description will be given to the accessing and Annotation functions. The writing function which is created by the PhD student Geeng-Neng You, is described here because it will be reused for making Annotation and Discussion. The description of the other functions of this system, such as exploration, can be found in the literature.[Mhashi1990]

4.2.3.1. Accessing

Accessing a document in MUCH consists of three functions: 1) reading, 2) browsing, and 3) searching.[Zeb1991] The following operations can be performed in the accessing part of the MUCH system:

1) Browsing annotations. This operation can be performed by selecting an annotation from the Annotation outline.

2) Reading an annotation for updates.

3) Browsing a text block. A text block can be either a paragraph of the original document or a note. Browsing a paragraph can be performed by selecting a frame either from the *Text Outline* window or from the *Frames Window*. A note can be browsed by selecting a note name from the list provided in the *Note Outline* window.

4) Reading a text block for update. The text block, in this case, can also be a text paragraph or a note.

5) Searching a specific text block. An outline of the index terms is provided in a window *Index Terms* (which replaces the *Note Outline* window). If the user selects any index term from that outline, all the text paragraphs are searched for that index term and a list of frames is displayed which contains that index term. The user can now look at any or all of the text blocks which contain that index term.

MUCH provides an outline (a hierarchical table of contents) of a document to help readers and writers visualize the structure of that document.[Rada1990a] One can define a document as a set of link objects which form the final version (e.g., a scientific paper, a book, or a report) of what the users intend to produce at the end of the document creation process. The link objects of the document are organized according to levels of abstraction, so that the contents of a text block which is attached to a link object lead naturally to the contents of the following text block.

The traditional outline consists of terms or headings in a hierarchy. In collaborative authoring, generating an outline in further detail enables the authors to dis-

cover the missing parts of the document, the Sections of the document which require more work, and the inconsistent Sections. In the current version of MUCH, the text outline (displayed in the *Text Outline* window) and the index terms are both generated manually. The intention is to develop a dynamic outline generator. Similarly, a word-frequency based indexing system will be introduced to create index terms automatically. These index terms will also be used as *cross-references* in order to make the browsing of related text blocks easier and faster.

4.2.3.2. Annotation and Discussion

Annotations are meant to be interpreted by others as a guide to changing a paragraph, link, or a frame. The Annotation facility enables users to write comments in the form of criticisms or suggestions about specific parts of an evolving document. The annotations are argumentative. Accordingly, there is a need for supporting the creation of "Link" instances. Annotations can be connected to the "annotated" blocks of text using simple labeled links. Both the destination and the link type should be determined by the user. Each annotation can be linked to more than one annotation.

In this system, there is no restriction on the link types and the node types, which are different from the node names which are assigned automatically by the system. In this system, each annotation is separated from the others. A relatively straightforward alternative to annotative windows allows an annotator to copy some material from the original paragraph and make changes to it.

The MUCH system can support Discussion, since MUCH supports annotations on annotations. A Discussion is similar to annotations on annotations except that there does not have to be a document which is at the root of the annotations.

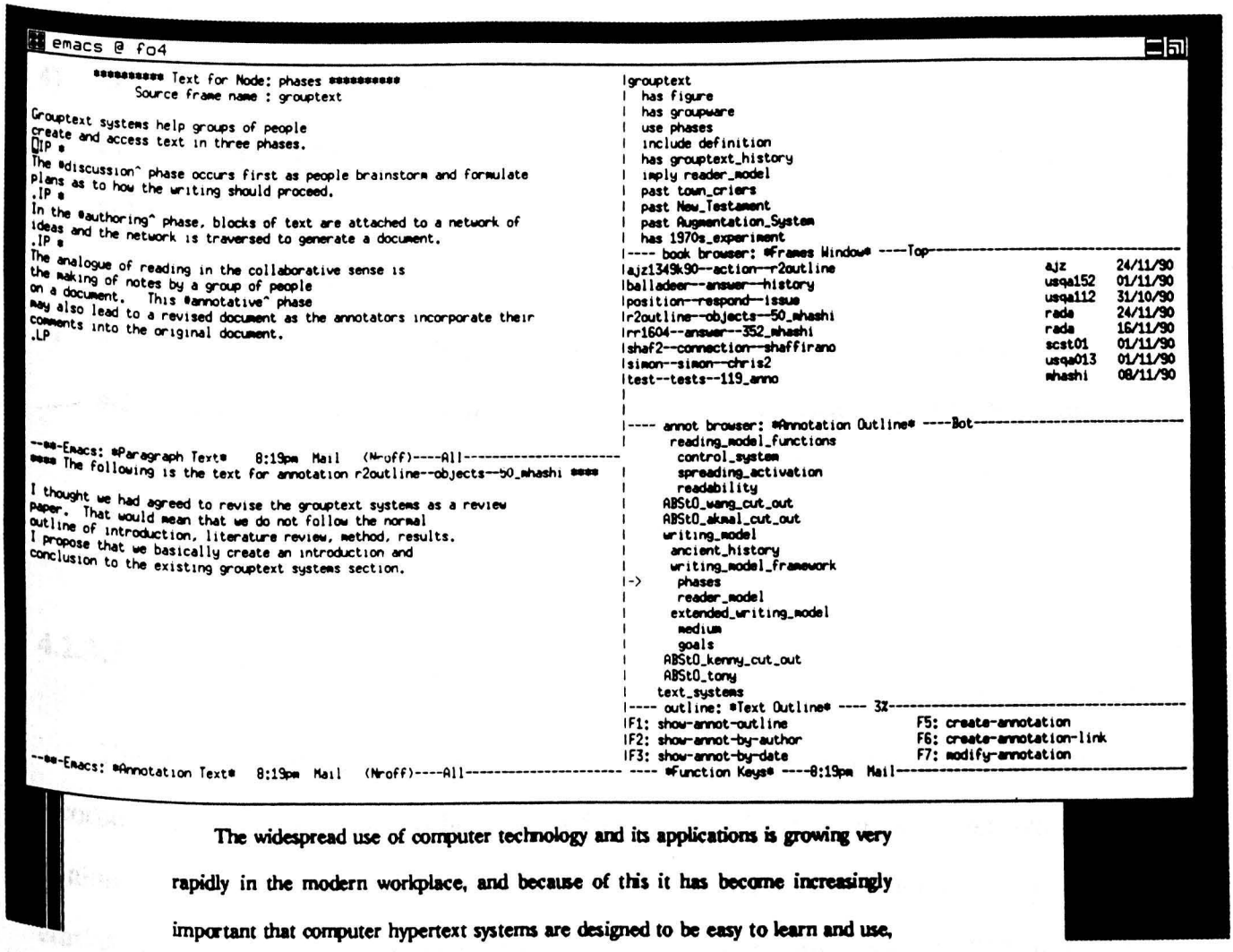
Furthermore, one may want to define Annotation so that it only points directly to a document. In this case, a comment on an annotation immediately becomes an instance of a Discussion.

The "Annotation mode" option brings forward a screen through which the user can perform the Annotation operations (see Figure 21). In order to create an annotation, the cursor should be in one of the two windows "*Frames Window*" or "*Annotation Outline*". The output of all reading operations will be displayed in a window called "*Annotation Text*". The following operations can be performed in the Annotation mode:

1) *Create Annotation:* Annotations can be created on a frame or on an annotation. If a frame has to be annotated, the cursor should be pointing to that frame either in "*Frames Window" or "*Text Outline" window. In order to create an annotation on an annotation, the cursor should be in the "*Annotation Outline*" window and pointing to the annotation that is to be annotated. The user is asked to label the node and link of the annotation and is given an option to create descriptive text which will be associated with the node and link. In this way the annotation can point to the document or to other annotations. A user may request annotations associated with a node of the document's semantic net or annotations created by certain users or at certain times.

2) *Create Annotation Link:* This function can be used to create new links between existing annotations and frames or between two annotations. (These links are extra links to give a meaningful look to the Discussion phase of the Annotation function.)

3) *Show Annotation Outline:* This function produces an outline of all the annotations on a frame to which the cursor is pointing. The cursor should be in "*Text Outline*" or "*Frames Window*". The outline for annotations is displayed in



The widespread use of computer technology and its applications is growing very rapidly in the modern workplace, and because of this it has become increasingly important that computer hypertext systems are designed to be easy to learn and use,

Figure 21 "Annotation mode": MUCH Screen. The large window near the center of the screen is the Emacs window which looks into the document and Annotation. Through command options of the Emacs interface, the user can open other windows, such as the one near the bottom of the screen which shows the formatted version of the document. The Emacs window contains several, labeled subwindows. From top to bottom on the right side: the "Frames Window" shows one frame; the "Annotation Outline" shows a list of links and nodes (in this case, as the result of a request for all annotations by one user); the "Text Outline" shows the outline of the document; and the "Function Keys" gives a menu of function keys and their actions. On the left side are two Emacs subwindows: "Paragraph Text" displays a text block of the document, and "Annotation Text" shows a text block of an annotation. When the user points to a term on the right half of the Emacs window and hits carriage return, then the associated text appears on the left half.

the "*Annotation Outline*" window. The user can browse through the outline by using the cursor keys. The text for a particular annotation can be read by pressing the <Return> key on that annotation. Text for that annotation is

displayed in the "*Annotation Text*" window.

4) *Show Annotations (by Author or by Date:)* "show-annot-by-author" can be used to obtain a list of all the annotations created by a particular author. The author's login name is required for this purpose. "show-annot-by-date" is used to get all the annotations created before, on, or after a specific date.

5) *Modify Annotation:* Modifying the contents of an existing annotation can be achieved by using this function. The cursor must be pointing to the required annotation in the "*Annotation Outline*" window. The user can save the changes by pressing the "store-in-database" key.

4.2.3.3. Writing

The writing function was created by the PhD student Geeng-Neng You and incorporates several features to help the writers in creating and updating hypertext. It aims to support writers who are involved in large and complex documents generating task. Its multiwindow user interface is consistent with other MUCH functions and allows the writers to view various referential information like frames, outlines and other texts while entering the target text paragraph. It also provides notes processing facilities for the writers to put their notes into the system. Other features include the outline generating capability and comprehensive indexing support for the writers.

To utilize these features, the writers can have their ideas recorded by using the notes processing tools to create, update and reference them. When the writers want to write the real text paragraph for the document, they have to create frames and link names for a specific paragraph before entering text for this paragraph. This step allows the writers to explicitly organize the hypertext as they wish. There are

also functions to handle the updating of the structuring entities such as frames and links, as well as real text entities like notes and text paragraphs. Under such writing environment, the support for the writers in the whole writing procedure becomes more organized and further strengthened.

The following is a brief description of the operation of this function:

1) The five windows for the writing function differ from the entry window by the "*Note Text*" window being replaced by the "*Write Text*" window. The "*Text Paragraph*" window is used for reference texts while the other three windows are still used for displaying referential information about the frames and outlines. These frames and outlines are manipulated by the writer during the writing process.

2) The referential information on the windows, other than the "*Write Text*" window, should be brought up or generated to help the writer obtain a clear idea about the current status of the hypertext document. The writer can thus select the write-function entry from the main menu to get into the writing function. The whole function is divided into two major parts, writing and updating.

3) Suppose the writer selects the creating function. The writer can choose to either enter the notes first by selecting the create-note option or be more straightforward by directly entering frames, links and the associated actual text paragraphs through help from the manipulation of the referential information in other windows. Options to enter indices before storing the text paragraph are also available.

4) If the writer chooses the updating function, functions to modify notes, text paragraphs, links or frames are provided. Again the updating process can be assisted from the manipulation of the referential information windows.

4.3. Case Study: MUCH as a Discussion System

Some literature and the results from the previous Chapter emphasize the importance of node and link types in the Discussion hypertext systems. In this study an attempt was made to explore some issues concerning Annotation and to find a suitable set of node and link types for Annotation. Being Allowed one to annotate an annotation was considered to be an instance of Discussion. Thus, the hypothesis was that the node and link types for Discussion are suitable for Annotation. The users were given an option to use:

1) their own node and link types;

2) a suggested fixed set of node and link types; and

3) both 1 and 2.

The usefulness of the suggested fixed set of node and link types for Discussion had been approved in some hypertext systems as well as in former work at Liverpool University. Thus, the hypothesis will be supported or refuted based on what the users are going to use.

Furthermore, an attempt was made to test issues such as: Should the system provide a fixed set of node and link types, or should users be given the option to add their own? Should a particular kind of structure be supported? Finally, the structure of the document is different from the structure of the Annotation. In the structure of the document, the text is attached to the link. By contrast, in the Annotation structure, the text is attached to the node. Thus, in this case study, an attempt was made to discover the advantages and the disadvantages of using different structures for document on the one hand and Discussion and Annotation on the other hand.

In this study, 9 students were asked to use MUCH, for one week, making Annotation on a Chapter in an on-line hypertext book, and to add annotations to

annotations (i.e an instance of Discussion). The Subjects had never used the system before the experiment. They were given one lecture on the topic of Annotation, and a 45 minute tutorial on the Annotation facilities of the system. They were also given guidelines about selecting a full meaning name to an annotation and to a link that connects two annotations. Some of the link types that were presented to them were: *responds, generates, supports, objects, comments, clarifies, questions*, and *suggests*. Also some of the node types that were explained to them were: *Issue, Position, Argument*, and *Comment*. They were asked to use these node and link types, or their own. The subjects were asked to create at least two annotations; one linked to the book, and the second linked to an annotation. It was expected that they would use the second annotation in resolving one or more issues collaboratively.

The subjects were given the option to create the annotations in either a hierarchy, or in a non-hierarchy (by creating cross-links between the existing annotations). The users were given the option of creating a link between any two existing annotations, for the purpose of nonhierarchical structure.

After one week, a total of 19 annotations had been created. Each students created two annotations which was adequate to answer the exercise. One student created three annotations. Seven annotations pointed to the Chapter as a whole. Two pointed to specific Sections of the Chapter, and the others were created as an annotation on an annotation (see Figure 22).

119_anno--comment--grouptext
119_anno_on_anno--solves--119_anno
179annot--comment--grouptext_interface_principles
179annot_annot--comment--179annot
PETERannot1--comment--MUCH
PETERannot2--comment-PETERannot1
agree--comments--simonannotate
balladeer--answer--history
chris1--answer1a--reader_model
chris2--answer1b--chris1
history--question-grouptext
issue-generates--grouptext
martyn--describes--grouptext
position--respond--issue
shaf2--connection--shaffirano
shaffirano--comment---grouptext
simon_annotat--simon_ann--119_anno
simon_reply--simon_link--chris2
simon_annotate--simon_link--grouptext

Figure 22 "Experiment Annotation outline": The Annotation outline. The format of each entity is source--link--target.

The structure was a tree, but it was very shallow (two levels) and one cannot infer from this result that the supported structure in Annotation is a tree (see Figure 23).

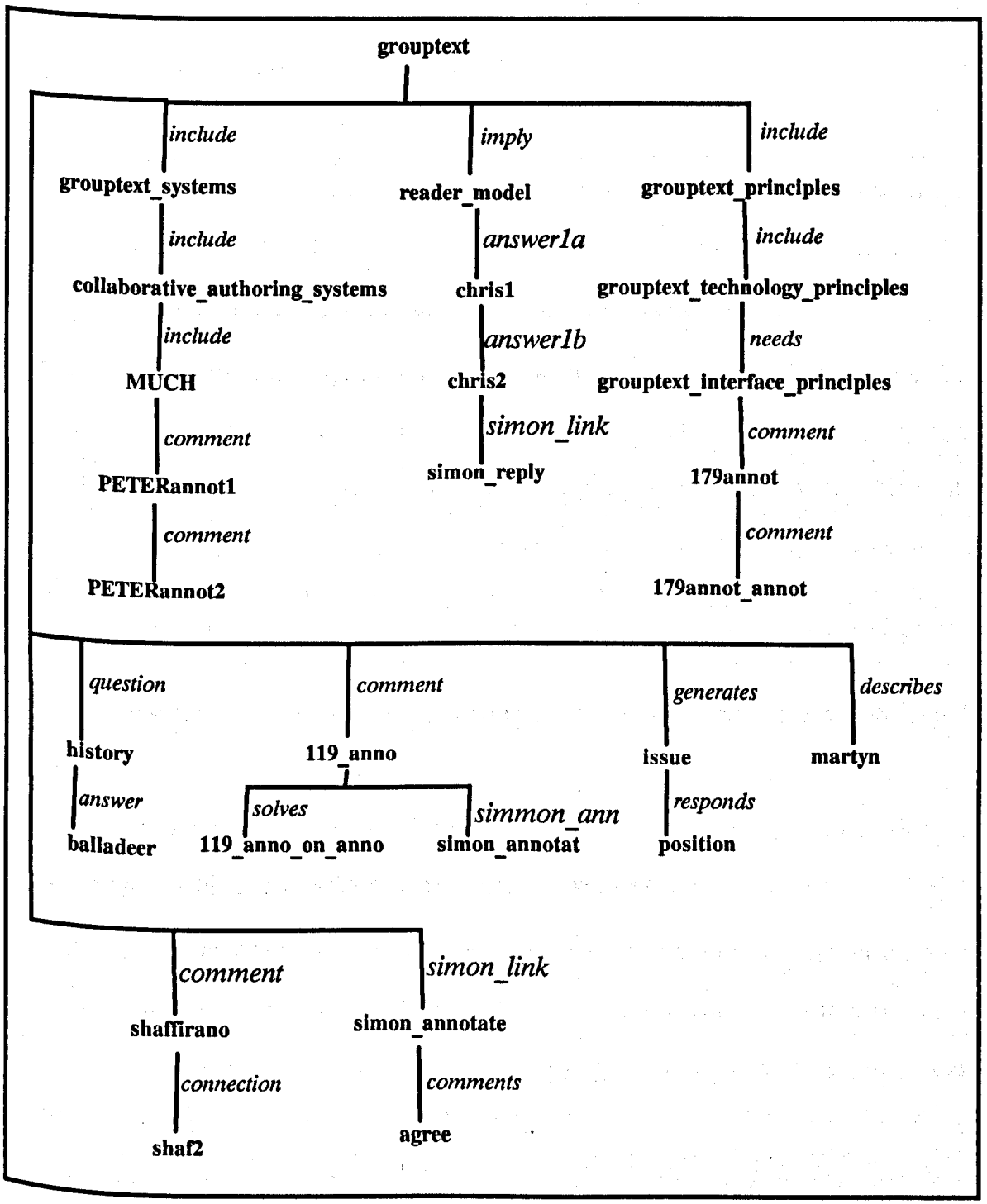


Figure 23 "Experiment Annotation tree": The Annotation tree structure

The data was collected and analyzed. The annotations that were created can be classified as follow:

1) system annotations: one annotation raised an issue, three annotations tried to resolve this issue, and five annotations concentrated on the advantages and the disadvantages of the system itself;

2) document annotation: four annotations raised issues about the the content of the Chapter, and two annotations tried to resolve some of these issues; and

3) nonsensical annotations: four annotations were nonsensical.

Of the presented link types, 68 percent were used by the subjects with a frequency ranging from 1 to 7. The remaining (i.e 32 percent) were added by the subjects. Regarding the suggested node types, only eight percent were used. This could be due to a discussion not having been created. The subjects raised some issues regarding the Chapter and the system, but they did not try to resolve any of these issues. However, the node and link types which were added by the subjects were given to two experts who knew the book to test the usefulness of these node and link types. They found that 40 percent of the link types were useful, 40 percent were mis-used, and 20 percent were similar to those that were presented. In contrast, all the node types which were added by the subjects were meaningless.

One of the problems concerned the structure Annotation: annotation pointing to a frame not to a link object. Subjects didn't always mention the link object in their texts. This caused some difficulty in specifying the text block which the annotations referred to. Another problem found in this structure was generating one outline for both the document and the Annotation. In the current structure, two separate outlines were generated, one for the document and another for Annotation. It was difficult to go back and forth between two different outlines. It would be helpful to see the outline for both the text and the Annotation in conjunction with each other, and this is not possible when using different structures.

4.4. Annotation Node Types

In the previous case study (Section 4.3) which was performed to test the usefulness of using some Discussion node types for Annotation, they were found to be less useful for making Annotation than they were for use in Discussion. This led to the exploration of some Annotation node types, define their meaning, and show how they might work in a document.

A document can be divided into two kinds: evolving document (i.e., a draft) and a published document (such as scientific papers or books) both of which are stored electronically in a form of hypertext. Documents contain two major types of information: content and organization. Content refers to the words and pictures that are the subject of the document. Organization specifies the order in which topics are presented and the relationships between them. Organization determines that topic X is a sub-topic of topic Y or that topic A precedes topic B.[Horton1989, Walker1988] Changing a document is dependent on the kind of document and the user. In terms of the evolving document, the change will include both types of information (content and organization). In contrast, the changes regarding the published document will be made only on viewing the organization of the document for presentation on the screen (i.e the original organization of the document in the storage will not be affected at all).

Most of the annotations created by users might be related to the two types of information embedded in a document. Annotators may create annotations suggesting what the author(s) should do, but they do not suggest how they should do it. For example, annotators see the word "Hypertext" in a document and suggest that this word should be defined, but they do not suggest a definition for "Hypertext", nor do they provide guidance to the author(s) about defining the term "Hypertext".

Sometimes, however, some users suggest both what should be done, and how the author(s) should do it.

Two groups of Annotaton node types are suggested, based on the proceeding information. Of course, the following set of Annotation node types will not cover all the possibilities of annotations; some of these Annotation node types might be ignored and some new types might be added during the next experiments and case studies. The suggested Annotation node types are as follows:

1) Organization Annotation node types which include *Good-Organization (GO), Bad-Organization (BO), Bad-Suggests-Organization (BSgO) and*

2) Content Annotation node types which include Good-Content (GC), Bad-Content (BC), Bad-Suggests-Content (BSgC), Worse-Content (WC), and Worse-Suggests-Content (WSgC).

Good-Organization Annotation type means that the entity in the outline (table of contents) which is attached to that annotation should be moved next to the previous entity with a *Good-Organization* Annotation type. If the first one which is attached to it is a *Good-Organization*, then it should be moved up to become the first entity in its level of the outline. For example, as in (B) Figure 24, the entity "Introduction" should be moved to the top of its level in the outline as it is in (C). While the entity "Discussion" in (C) should be moved next to the entity "Annotation" as it is in (D).

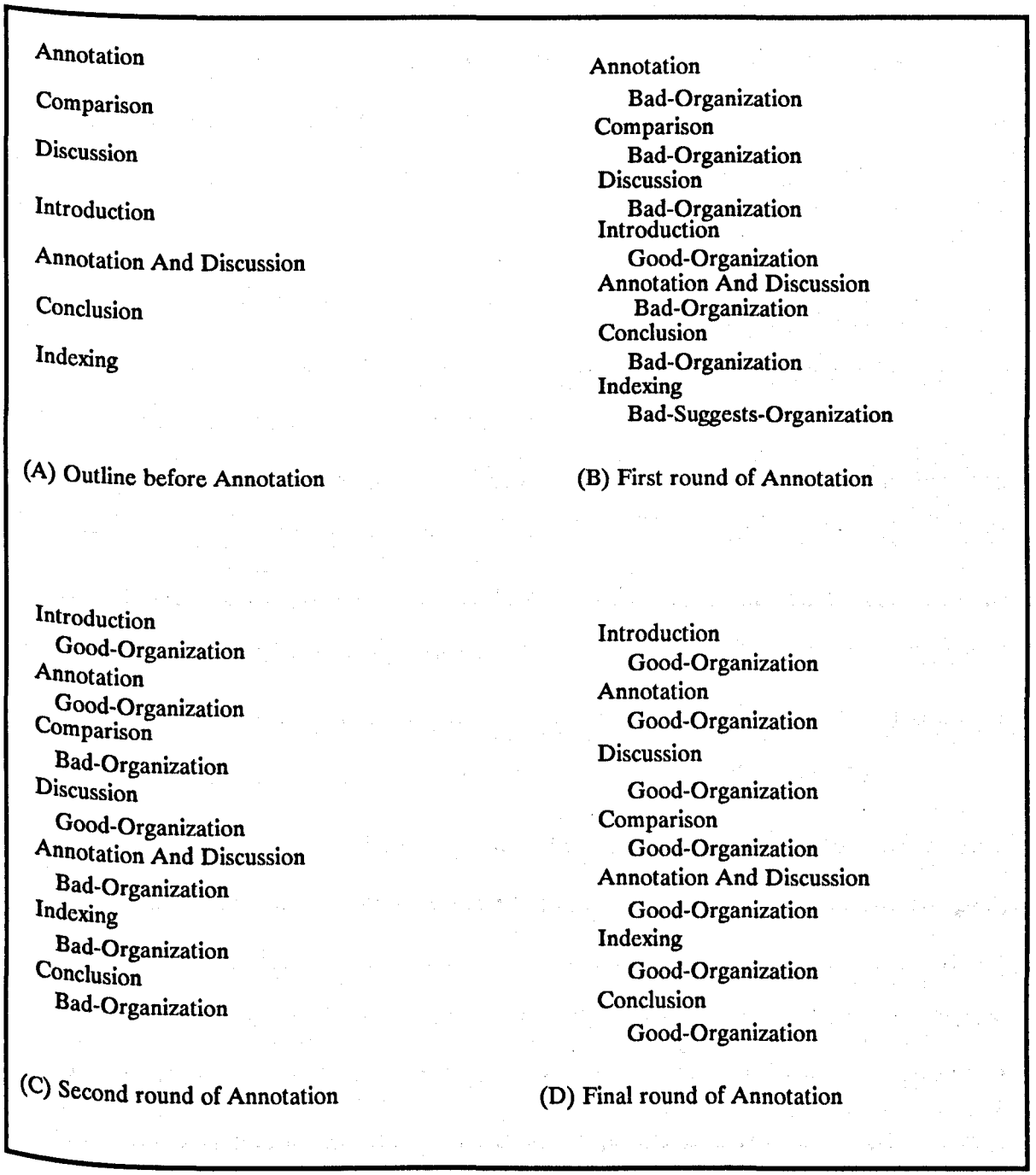


Figure 24 "Re-organization example": Affect of Annotation on the organization of a document.

Adding the word "Bad" in the annotation type, means that a user has made a suggestion in the text which is attached to that annotation, and that this is concerned with what the author(s) should do, but offers no suggestion as to how it should be done. Adding the word "Suggests" means that a user suggested in the text attached,

both what should be done, and how it should be done. A *Good-Content* Annotation node type means the text attached to the link object is good (consistent with its heading, coherence, no English errors nor spelling mistakes, and needs no modification). A *Bad-Content* Annotation node type means the opposite of *Good-Content* Annotation node type. A *Worse-Content* Annotation node type means the text which is attached to the link object should be cut out, or needs a lot of work prior to fixing.

Systems provide an outline (a hierarchical table of contents) of a document to help annotators visualize the structure of that document. In MUCH, users can view individual frames or hierarchically organized frames following outlines that are automatically generated by traversals of the semantic net. However, the user should explicitly define a linear ordering of the links and specify a starting frame name to generate an outline. The outline consists of the nodes related to the document and Annotation. The two can be distinguished from each other by looking to the first letter to see whether it is a lower-case (for text) or the letter "A" (for Annotation).

A good document should have good content, good organization, and a good relationship between the headings and the text. Having one of them is not, in itself, enough for a good document. An example will make this point clearer. If we assume that all the scientific papers published in journals are good documents, we cannot similarly assume that a good document will result by taking an organizational structure which is good (according to the assumption) and replacing its text with texts from different papers. This is will not result in a good paper, because the relationship between the headings and the text will not be consistent. These three factors might be used by authors as an indication of whether they are close to, or far from, reaching a good document. This can be done by looking at the document they are creating to see whether the four factors are satisfied or not.

One might ask: Could the computer help authors in satisfying some, or all, of these factors? Computers might help authors in the organization. This may be done by attaching annotations to the different parts of the document. In terms of the organization, users attaching the Annotation types *Good-Organization* and *Bad-Organization* to the different parts of a document, might allow the computer to help in reorganizing the document, so that it becomes closer to a good organizational state, which is one of the most important factors for a good document. In the next Section, two experiments are reported which test this hypothesis, test the usefulness of the suggested Annotation types, and explore some issues and new node and link types concerning Annotation.

4.5. Investigations of Annotational Organization and Content

A case study which was performed to test the usefulness of Discussion node and link types for Annotation (Section 4.3), has shown that it is difficult to perform certain operations when different structures and different interfaces are used. It has also shown that it is difficult to read annotations when their outline is separate from the outline of the document. In order to solve some of these problems, a single structure and interface for the document, Annotation, and Discussion was used. The writing document function (Section 4.2.3.3) was used also for Annotation and Discussion. But, how could a user distinguish between document, Annotation, and Discussion? As a temporary solution, the user might type the name for a node related to the text in small letters. For the Discussion and Annotation, the user might add the capital letter "D" or "A" at the beginning of the node name, to denote the beginning of a Discussion or an Annotation respectively.

In the previous structure of Annotation (annotation pointing to a frame), a user

should mention the link object in the annotation text. In this new structure (annotation pointing to a link object), if a text block consists of two or more paragraphs and the user wants to make an annotation on a specific part of the text, the user should specify that fact by explicitly writing (eg. "paragraph 2, line 3") in the text of the created annotation.

In the following two Sections, two experiments were performed. The first experiment concerned the organization of a document and the second experiment concerned the content of a document. In addition to the exploration of some issues about Annotation in both experiments, the usefulness of using Annotation organization node types was tested in the first experiment. In the second experiment, the usefulness of using Annotation content node types was tested and a comparison between three different approaches of making annotations on a document was made.

4.5.1. Experiment 1: Annotation Organization

In this experiment, we tested the role of hierarchies in Annotation by trying to answer the question: Could the computer help an author(s) satisfy one or both document requirements of good content and organization? If an author wants to create a good document (such as a scientific paper, book, or report), then that document should contain good content and organization. During the writing process, an author wants to know how close he/she is to reaching the final document. An author wants to decide whether the final document has been reached or not. Some more questions need to be answered during the writing process. Some of these questions are: What are the most important parts of the document that should be written first? What are the priorities for making a response to some annotations made on the document? This experiment was performed in order to answer such questions and

concerns, and in order to test the organization Annotation types which are suggested in the previous Section.

It was hypothesized that authors could be provided with indications about the quality of their documents. This is would be achieved by by moving the entities, including the text, which is also associated to the links, with the "Good-Organization" Annotation node type to the top of its level in the outline; and by using the "Bad-Organization" and "Bad-suggests-Organization" to move an entity from one place to another would give an indication to author(s) about the quality of that document. Assume that we have a good Chapter from a book and the organization of that Chapter is changed. In such a case, the hypothesis can be tested before modification by comparing the outline produced by the subjects at the end of the experiment to the original outline of the Chapter. If the two outlines are identical, then the hypothesis is fully supported. Otherwise, a statistical test might be needed. Whether or not the hypothesis will be supported depends on how close the produced outline is to the original outline of that Chapter.

Before going to the experiment, one important issue needs to be resolved. If two users disagree: one attaches to an entity in the outline the *Good-Organization* Annotation type and the other attaches the *Bad-Organization*, then the following question might be asked: How could the computer decide whether that entity in the outline is good or bad organization? In answering such a question, different methods might be applied:

1) User importance: the users are authors and annotators. An author is more important than an annotator and the user who has the higher degree is more important than the other, hence a Professor is more important than a Doctor. The authors might then be divided into two groups. Authors can make changes by consulting others before the changes are implemented, and authors can

make changes without consultation. The annotators also might be divided into two or more groups based on the relation between the annotators and the authors. For instance, annotations made by an annotator who has received assurance that his name will appear in the acknowledgement Section might be different from annotations made by annotators who do not expect acknowledgement. Thus, a system supports user importance, then the annotation created by the more important user will be considered.

2) User access: The user who has "write-access" to the document has priority over a user who has "read-access" only to that document. A system that supports the access privileges will consider the annotation that is created by a user who has that "write-access".

3) Argument support: one user might add an argument supporting his response, while the other does not. In such cases, the annotation that is supported by an argument is considered. If both users add an argument and these arguments are considered, and one of them uses mathematical, experimental, or psychological proof, mathematical proof is considered strongly than experimental or psychological proof.

4) Date: the annotation that was created most recently would be considered more important than others. For example, a user creates an annotation before a requested modification takes place, and later another user creates an annotation after the modification takes place. The last annotation might be more accurate than the previous one.

5) Voting scheme: this method is a solution for the following cases. First, when a system has no support for any one of the methods described above. Second, when the two annotations have the same properties: both annotations created by two users with the same degree and access, both annotations contain

arguments (mathematical, experimental, or psychological), or both annotations were created on the same date. In the case of voting Yes/No, there is no problem if there are an uneven number of users. Where there are an even number of users, they can decide from the beginning what should be considered in such.

The subjects participating in this experiment were 4 postgraduate students. Three of the students had read the book before the experiments began, and each one of them is an expert about part of the book. They had annotated different parts of the book. By contrast, the fourth subject had not read the book. Thus, three students might be considered as experts as opposed to the fourth subject who might be considered as a novice.

A Chapter was selected randomly from an on-line hypertext book. The first two levels of the outline for the selected Chapter were chosen for use in the experiment. The structure of these two levels of the outline had been changed and some Sections from different Chapters selected and randomly added, to make sure that the organization of that outline was completely changed (see Figure 25).

The outline shown to the subjects appears in Figure 26. A tutorial was given to the subjects explaining the use of the Annotation types by demonstrating some examples, and explaining some other related issues such as how a decision can be made when an objection occur to a specific organization of two entities in the outline. The subjects were asked, in their own spare time, to follow one of the three cases concerning annotation:

1) Creating an annotation with "Good-Organization" type and linking it to an entity about which they are sure of the order of the outline;

2) Creating an annotation with "Bad-Organization" or "Bad-Suggests-Organization" type (depending on whether or not they know the reason for

```
expertext
        include figure
        include definition
        has Dynabook_history
        include expertext_principles
                past expert_system_trends
                example INTERNIST
                include synergism
                include expertext_link_principles
                include distributed_expert_principles
        include expertext_systems
                include microexpertext_systems
                include macroexpertext_systems
                include groupexpertext_systems
                example intelligent_requirements_tools
        include expertext_requirements
                extend artificial_intelligence
                use semantic_net_reasoning
                use procedures
                use collaborative_authoring
                use network_consrtuction
                use knowledge_bases
        include expertext_exercises
                apply inheritance
                has outline
                apply metric
                apply resolution_algorithm
                apply logic_microexpertext
                apply procedural_microexpertext
                apply adaptive_weights
                apply computer-assisted_instruction
                apply spreading_activation
                apply groupexpertext_messaging
```

Figure 25 "Organization experiment outline" (A): The first two levels of the outline for Chapter "expertext" before modification.

moving a specific entity after or before another entity), and linking it to an entity in the outline. This case might be used when the first case is not valid or when there is the need for objection against assigning an annotation to an entity; and

3) Creating no annotation at all if the subjects are not sure about a specific order for an entity.

```
expertext
        include expertext_systems
                example intelligent_requirements_tools
                include macroexpertext_systems
                include groupexpertext_systems
                include microexpertext_systems
                has microexpertext_system_creation
        include figure
        has text_principles
        include definition
        include expertext_requirements
                use network_construction
                extend artificial_intelligence
                use knowledge_bases
                use procedures
                use collaborative_authoring
                use semantic_net_reasoning
        has Dynabook_history
        include expertext_principles
                has text_principles
                include synergism
                include distributed_expert_principles
                example INTERNIST
                past expert_system_trends
                include expertext_link_principles
        include expertext_exercises
                apply groupexpertext_messaging
                apply resolution_algorithm
                apply inheritance
                apply metric
                has outline
                apply logic_microexpertext
                apply procedural_microexpertext
                apply adaptive_weights
                apply computer-assisted_instruction
                apply spreading_activation
```

Figure 26 "Organization experiment outline" (B): The first two levels of the Chapter "expertext" outline after modification.

Creating one annotation by one user and linking it to an entity is enough if that annotation is the same option for all the others. Otherwise, such as in the case of an objection, a Discussion might be needed. If there is no consensus, then all the opinions should be expressed by creating an annotation and linking it to that entity, in order to make a decision by using the Yes/No voting scheme.

The re-organization process will be made manually by the facilitator in two

phases. In the first phase the outline will be re-organized in terms of the "Good-Organization" Annotation type so that the entities at a specific level in the outline with the "Good-Organization" Annotation types should be moved to the top of that level in the hierarchy, and the others moved to the bottom. In the second phase, the outline will be re-organized in terms of the "Bad-Organization" or "Bad-suggests-Organization". In this phase the facilitator should read the text of that annotation, which is attached to the link, and change the order according to to this text. As a result of the re-organization process, a new outline can be generated automatically. The process could be repeated several times; where "several times" can be determined either by reaching a state where all the entities are annotated with a "Good-Organization" Annotation type, or by a time limit. To avoid the experiment ending at an unsuitable point, the subjects were asked to create annotations on the first two levels of the outline. After that, if they still had time to make more contributions, then the same process would be continued on the remaining levels. Of course the subjects were not restricted to a specific starting level (i.e the users could make their annotations hierarchically or nonhierarchically).

After three days, 9 nodes, 23 links, and 9 blocks of text had been created. Three different types of annotations were used (see Figure 27. All the annotations that were attached to an entity were ordered by time (i.e the annotation which is next to an entity is created before the others). At the time of conducting this experiment, the name of the user who created a node name was added to the Annotation node name (such as AGO_tony). The subjects started creating their annotations on the first level of the outline, then they began to continue making annotations on the second level. However, because most of the subjects had no time to continue the experiment, they did not finish making annotations on the second level.

The outline was re-organized twice. Firstly, the outline was re-organized in terms of the "Good-Organization" Annotation type. Secondly, the outline was re-organized in terms of "Bad-Organization" Annotation type. In the first phase of the re-organization process, all the entities are searched from the top of the first level of the outline to the bottom. All the entities that were attached to them, the "Good-Organization" Annotation type were moved to the top of its level. For each Annotation "Bad-Organization" that was attach to an entity, the text that was attached to the link was read. The entity was moved to the place according to the text attached. Each subject then started expressing his opinion by creating an annotation and attaching it to the intended entity. Some of the entities had just one annotation attached to them. Some of the entities had two, and others had more. With those entities that had one or two annotations attached to them there was no objection to their organization. On the other hand, there was objection to the others, and every subject had expressed his opinion by creating an annotation for the voting scheme purpose.

The subjects agreed in a face-to-face meeting before they began the experiment that where a Yes/No voting scheme threw up a 50:50 result about an entity being "Good-Organization", applying that scheme on the first level of the outline would not necessitate any more re-organization. So, as we see in Figure 27, the structure of the produced outline was different from the original outline before modification (see Figure 25). According to the testing plan, a statistical test is needed to test the significance of the number of interchanges which are needed to make the produced outline identical to the original outline. Since only one entity needs to be changed, compared with eight entities which are the same as the original, it is clear that there is no need for a statistical test. Thus, one can conclude that the *Annotation organization node types* can be used as a viable method of giving an author an indication of

the good quality, good content and organization of the document (assuming the content is good).

```
expertext
  include figure
        corrects AGO_tony
        suggests ABO_kenny_cut_out
        corrects AGO_akmal
        corrects AGO_wang
  include definition
        corrects AGO_tony
  has Dynabook_history
        corrects AGO_tony
        suggests ABO_kenny_cut_out
        corrects AGO_akmal
        corrects AGO_wang
  include expertext_principles
        correct AGO_tony
  include expertext_requirements
        suggest ABO_tony
        corrects AGO_tony
        suggests ABO_akmal
        corrects AGO_kenny
        suggests ABO_wang
  include expertext_systems
        suggest ABO_tony
        corrects AGO_tony
        suggests ABO_akmal
        corrects AGO_kenny
        suggests ABO_wang
  include expertext_exercises
        suggests ABO_tony
        corrects AGO_tony
  has text_principles
        suggests ABO_tony_cut_out
```

Figure 27 "Organization experiment outline" (C): Part of the first level of the outline for the Chapter "expertext" at the end of the experiment.

Some problems were faced in using the system in this experiment. The subjects needed to enter most of the information by typing. It was also sometimes for the subject to remember all Annotation node types. In addition to this the system was slow, especially in generating the outline, which is the most important operation in this method. This is because the modification cannot be seen unless the outline is re-generated. The greatest disadvantage of this method is that without any control

on making annotation, a side-effect might occur. For example, assuming that the entity "K" is the latest entity with "Good-Organization" at the first level, one subject intends to put entity "L" next to entity "K" and another subject intends to put entity "M" next to entity "K". The subject who performs the operation first will cause confusion to the other subject.

4.5.2. Experiment 2: Annotation Content

In this experiment, a task has been created for the purpose of making annotations on this dissertation. The subjects were the author and the advisor. The goal of this test is to improve the first draft of this dissertation by making annotations and to explore some issues about Annotation. This experiment occurred in three phases. In the first phase the annotations were made on the paper version of the dissertation. In the second phase the annotations they were entered into an ascii file and in the third phase annotations were entered directly into the MUCH system. The hypothesis being that using the computer will diminish the need for face-to-face Discussion.

The comments on paper were made over a period of 4 days and took about 5 hours of the advisor's time. There were 102 annotations and they could be directly transcribed into approximately an equal number of lines of text. In other words, each annotation was very short and derived some of its strength from the pointer to the relevant part of the dissertation which was being critiqued. The annotations were classified into five groups based on the level of organization in the document to which they pointed. The 120 annotations could be classified as: 13 concerning the whole document, 13 about Chapters, 20 about Sections, 26 about paragraphs, and 30 about a specific region or term. In response to two annotations, the subjects had a

Discussion that lasted for two hours and there was nothing documented at the end of the Discussion.

To avoid going back and forth between the screen and the paper copy when making a response, the annotations from the paper version were entered manually into an ascii file. Some difficulties were found in determining the pros and cons of annotation. There were different strategies for making annotations on the paper version which were difficult to implement in the ascii file.

The total number of annotations in this approach was 140 annotations (256 lines of text) including most of the annotations from the paper version. Some difficulties were found in making annotations in the ascii file. Some of these difficulties are:

1) The difficulty of representing the history of annotation (such as making first annotation, response on that annotation, response on response, and so on);

2) The difficulty of making annotation on a group of text blocks; and

3) The difficulty of finding the new responses.

During 45 minutes working on a flat ascii file, 30 minutes of this time was face-to-face Discussion to explain how the response can be found. However, understanding the annotations which were created in this phase needs less Discussion than the paper approach, and more detailed information was written.

The difficulties in both paper and flat ascii file led to the use MUCH system. The electronic version of the dissertation was automatically loaded into the MUCH system and then the annotations which were not responded to in the paper version were manually entered into MUCH. This manual transcription proved somewhat tedious, particularly when entering 30 annotations which were concerned with specific parts of paragraphs. The MUCH system did not allow the user to mark

parts of a paragraph and then point the annotation at those parts. Rather the annotation could only point to the semantic net and then within the text of the annotation there could be explanation of the part of the paragraph to which the comment was applied.

About 5 hours were spent over several days with the two subjects both making comments through the MUCH system. Only 24 annotations were created but they occupied 460 lines of text. Eight of these annotations led to a Discussion. These annotations varied widely in size, as one contained over 40 lines of text and another contained no other text beyond the node and link label. The link types which were applied also varied widely and included "comment", "support", "suggest", and "bad'.

The Annotation process stimulated verbal Discussion which occupied a large percentage of the 5 hours. Seven of the 24 annotations addressed the weaknesses of the MUCH system itself, rather than the weaknesses of the dissertation. In one hour long session, after the users had properly initiated the MUCH system and prepared to type comments, they then instead spent the final period of 45 minutes in verbal Discussion, of which 35 minutes concerned the features of the Annotation system rather than the document.

Version management became a problem as the two users were modifying the document during the same period of time that annotations were being made. In this way, two of the annotations became impossible to understand, as the text to which they were referring had been removed.

CHAPTER 5

Browsing and Searching

5.1. Introduction

The browsing function of hypertext systems enable the reader to inspect the node contents and to navigate through the network by selecting the path to follow, on the basis of interests emerging along the way.[Lucarella1990] This function is supported by all the hypertext systems (see Chapter 2). However, there is a striking consensus among many of the experts in the hypertext field that navigation is the greatest difficulty for users of hypertext.[McKnight1991] The function offered by most hypertext systems does not meet the needs of readers who are unfamiliar with the information being presented.[Zellweger1989] Different researcher have different ideas about how a hypertext should be navigated. Each new implementation of a hypertext browser works slightly different from previous ones.[Furuta1989] This Chapter explores a new method for browsing and searching a single document by:

1) using word-based indexing of paragraphs for searching and
2) automatically arranging the so-derived index terms in a hierarchical semantic net for browsing.

Macrotext emphasizes the links that exist between many documents rather than within one document. Microtext is a single document with explicit **links** among its components. In many situations a single document must serve a wide range of users, and the user's needs determine the style of access. Browsing and searching are two different styles of access to a document.

Discussion, Annotation and the draft textbook. It was hypothesized that the statistically derived index terms were important indicators of the paragraphs' contents. The first set of experiments performed on regular text proved the accuracy of the frequency-based index terms and the usefulness of STADBUILDER in recognizing the relationships between the text blocks. The relationships between the text blocks drawn by applying an algorithm called RELATION to the index terms and the hierarchical semantic net.

The positive results encouraged the testing of the following hypothesis: the word-based indexing of paragraphs and the automatically derived semantic net could be helpful in hypertext searching and browsing. Two additional experiments were performed to assess the usefulness of the statistically-generated index terms and the relationships between them for searching and browsing. The results suggest that word-based indexing of paragraphs and the automatically derived semantic net are useful also for browsing and searching.

The accuracy of word-indexing text blocks and the automatically derived hierarchical semantic net and their usefulness for browsing and searching led us to use Mili's algorithm to test the similarity of the Discussion and Annotation. Two further experiments were performed. The first experiment tests the similarity between the text of the Discussion and the text of the Annotation. The second experiment tests the similarity between the Discussion nodes relationship and the Annotation nodes relationship.

In Section 5.2, the different algorithms: Indexing, STATBUILDER, and RELATION will be described. The accuracy of these algorithms will be tested in Section 5.3.1. The new method of browsing and searching will be tested in Section 5.3.2. Finally, the application of this new method of browsing and searching on the two functions Discussion and Annotation will be tested as well as testing the similarities

A semantic net whose nodes point to blocks of text may support both microhypertext and macrohypertext. In the microhypertext case, the blocks of text may be of paragraph size, while in the macrohypertext case, the blocks of text are entire documents. The semantic net of hypertext is an important foundation for browsing and searching, but is also notoriously difficult to develop.[Rada1987] It is this difficulty that motivated the search for strategies based on word-frequency.[Rada1989b]

While the knowledge in people's minds is structured more like a semantic net than like prose,[Kintsch1988] people have little training in directly representing semantic nets. Researchers in natural language processing have been working for decades on the problem of translating documents into semantic nets or other knowledge representations. The parsing of natural language has been successful in narrow domains where the computer could be provided with a substantial knowledge of that domain [Carbonell1984] and some success has been attained in the automatic augmentation of knowledge bases from text.[Lebowitz1986]

Research in information retrieval has shown the value of word-frequency strategies for indexing and searching.[Salton1983] Word-frequency indexing takes advantage of the observation that terms which occur relatively frequently in a document are important indicators of that document's content.[Luhn1958] Patterns of co-occurrence of terms may also give important indicators of relationships among terms.[Raghavan1979] Mili[Mili1987] went a step further and proposed an algorithm that organizes a set of index terms into a hierarchy based on their frequency of co-occurrences in documents' descriptions.

Salton's algorithm[Salton1983] was applied to index Discussion text blocks, Annotation text blocks, and the paragraphs of a draft textbook entitled *Hypertext* written by Roy Rada using MUCH. Then, Mili's algorithm[Mili1987] was used to organize the derived indexing vocabulary in a hierarchical semantic net separately for the

between these two activities in Section 5.3.3.

5.2. Algorithms

A key feature of hypertext systems is that they enable users to browse through semantically related parts of a document. To this end, one must provide the means for: 1) describing the contents of the various parts of a document to allow direct access to those parts, and 2) describing the various semantic relationships that exist between the parts of a document to allow content-based browsing. In this Section, it will be shown how: 1) frequency-based methods can be used to extract content descriptors from textual units and 2) Mili's algorithm can be used to organize these content descriptors into a hierarchical semantic net, thus providing semantic access paths between the parts of a document. In Section 5.2.1, a frequency-based content analysis method that is commonly used in information retrieval systems is described, and its applicability to this case is discussed. In Section 5.2.2, Mili's algorithm is described, followed by a description for the RELATION algorithm in Section 5.2.3.

5.2.1. Indexing Algorithm

Two of the existing basic approaches to document content analysis are the *interpretative* and *structural* approaches. In the interpretative approach, semantic information is provided about the domain and often about the English language itself, so that the document may be understood before it is indexed.[Kaplan1990] In contrast, the structural approach uses the frequency of usage of words in natural text as an indicator of content relevance, and no semantic interpretation is assigned to any word. The index selection can also take into account other structural information such as word adjacency. The structural approach is based on the observation that writers

usually repeat certain words as they advance or vary their arguments and as they elaborate on an aspect of a subject. This approach has motivated a number of automatic indexing algorithms for information retrieval systems.[Salton1983] The advantages of these algorithms are practicality and ease of implementation. A structural approach, and use a variant of frequency-based indexing will be applied.

A typical frequency-based indexing algorithm proceeds as follows:[Salton1983]

1) Identify all the unique words in the document set.

2) Remove all common function words included in a stop list[Rijsbergen1979] (such as "a", "the", "and", "is", ..., etc.)

3) Remove some suffixes and combine identical word forms (stems). This reduces a variety of different forms such as analysis, analyzing, analyzer, and analyzed, to a common word stem "analy."

4) Determine the frequency of occurrences of the resulting word stems,

5) Give a range of frequencies for terms to be considered for indexing. All terms that are within the range are used as index terms.

More elaborate filtering mechanisms exist that use entropy-based measures that attest to the extent to which a given index term discriminates between documents in the document set.[Salton1983] Terms with poor discriminatory power are disregarded.

Variations of the above algorithm have been used extensively in information retrieval systems, despite harsh criticisms from the proponents of knowledge-based interpretive indexing. Salton argued that the interpretive methods have yet to justify their added complexity, both computationally and conceptually, by provably better retrieval performance.[Salton1986] However, there remains a major problem in this case: while frequency-based indexing generates a flat description, i.e., an unordered set of index terms for each document, MUCH documents are indexed by binary

relationships between terms. For example, a document that defines semantic nets might be characterized by the phrase "*a semantic net* IS-A *graph*", where *a semantic net* and *graph* are both frames (see § 3.3). Salton's algorithm might identify *a semantic net* and *graph* as index terms for that document, but it would not specify the relationship between them. Therefore, the decision was made to index the documents in two stages:

1) Identify the important content descriptors for each document, and

2) Apply Mili's algorithm to identify the relationships between co-occurring index terms

Mili's algorithm is described in the next Section.

5.2.2. Mili's Algorithm

The algorithm developed by Mili[Mili1987] is based on the widely accepted hypothesis that words (or terms) that often co-occur in textual documents tend to be related according to some relationship important to the document space. [Lesk1969, Soergel1974, Raghavan1979] He goes one step further, using empirical data from [Kaplan1990] and [Jones1980], showing that the most commonly used content descriptors within a document collection tend to be more general. He then hypothesized that the most dominant relationship between related terms with different scopes (or degrees of generality) is "Broader-Term." However, he provides no heuristics for identifying other kinds of relationships, either hierarchical or non-hierarchical, that may be equally important to the document space. His algorithm–called STATBUILDER, organizes a set of content descriptors (or index terms) into a hierarchy using the above hypotheses.

STATBUILDER relies on the availability of frequency information consisting of:

1) the number of times each content descriptor is used in the document set. This information is used to infer the "scope" of the various content descriptors.

2) and the number of times two content descriptors are used jointly to describe a document, for all pairwise combinations of content descriptors.

This information is embodied in a matrix MAT, where MAT(i,j) is the frequency of co-occurrence of term t_i with term t_j. MAT(i,i) is the number of documents in which t_i was used. The rows and columns of the matrix are organized by decreasing order of MAT(i,i) so that for $j < i$, t_j has a broader scope than t_i. The algorithm is as follows (assuming N index terms):

1) Normalize the elements of the matrix M by dividing M(i,j) by the square root of M(i,i)× M(j,j).

2) Choose the terms to be included in the first level of the hierarchy. Make these terms point to the root of the hierarchy. The purpose of the root node is to connect the 1st level terms. The root node can be called anything; call it KBROOT (Knowledge Base ROOT). Assume that the terms t_k through t_{k_1} were chosen to be included in the first level.

3) For $i = k_1 + 1$ through N

   3.1) Find the maximum of the elements M(i,1) through M(i,i-1). Note that because of the ordering of rows and columns (step 2), these are the frequencies of co-occurrences of t_i with the terms whose occurrences are higher than that of t_i.

   3.2) Create a link between the term t_i and all the terms t_j such that $j<i$ and M(i,j) is the maximum found in 4.1.

Mili reported an experiment where STATBUILDER was tested on a set of index terms used to describe articles from biochemistry literature.[Mili1987] The resulting hierarchy– call it STATKB, was inspected for the predominance of "Broader-Term" relations. About 50% of the hierarchical links were indeed "Broader-term". To assess the significance of the remaining links, STATKB was evaluated for the extent to which it could be effectively referenced by a semantic-net based concept-matching algorithm– called DISTANCE, which measures the relevance between documents and queries.[Rada1989a] DISTANCE's measurements were compared to people's relevance assessments for the same set of documents and queries. The DISTANCE evaluation correlated well with the human evaluation and suggested that the statistically-derived relations were cognitively meaningful.[Mili1987]

In this case, the relationships generated by STATBUILDER would be used to replace the statistically-derived flat set of index terms used to describe each document by $<term_1$ Broader-Term $term_2>$ triples. For example, if a document D were indexed by the set I={*a semantic net*,*graph*}, and if STATBUILDER identified a broader-term relationship between *a semantic net* and *graph*, then I would be replaced by the singleton I' = {<*a semantic net*,Broader-Term,*graph*>}. While this would account for neither all the terms in an index set, nor all the relationships between these terms, it would provide a useful partial description of the document's contents.

5.2.3. The RELATION Algorithm

Salton's algorithm identifies the important content descriptors for each text block. Applying Mili's algorithm to these descriptors identifies the relationships between them. The RELATION algorithm identifies the relationships between the

indexed text blocks. Assuming that:

1) there are n text blocks (TB_n) where each text block is indexed by a set of index terms (IND) and

2) the derived semantic net is produced by applying STATBUILDER to the index terms and their co-occurrences.

The relationships in the derived semantic net will be in the form $term_k$ *isa* KB (i.e the index terms which are chosen to be included in the first level) or $term_j$ *isa* $term_m$, where $term_j \neq term_m$ (those index terms which are in the second level or lower). The RELATION algorithm goes as follows:

1) For i = 1 through K

   1.1) Find all the text blocks which are indexed by $term_k$.

   1.2) If there is no link created between any two text blocks in 1.1, then create a link.

2) For i = 1 through j

   2.1) Find all the text blocks which are indexed by $term_j$ and $term_m$.

   2.2) If there is no link created between any two text blocks TB_j and TB_m in 2.1, then create a link.

An output from this algorithm can be seen in Figure 30.

5.3. Experiments

In this Section, three groups of experiments were conducted using the index terms generated by Salton's algorithm and the relationships identified by Mili's algorithm. In the first group of experiments, the accuracy of indexing and the usefulness of the hierarchical semantic net were tested in recognizing the relationships between

different text blocks of a document. However, the relationships which are identified by Mili's algorithm may nevertheless prove useful in the context of hypertext systems, in the same way that Mili showed the usefulness of his hierarchy in the context of information retrieval.[Mili1987] The second group of experiments were conducted to assess the usefulness of the semantic net generated by STATBUILDER for typical hypertext tasks. All the hypotheses in these experiments were supported. This led to the use of the index terms and the hierarchical semantic net relations as a testing tool. In the third and final group of experiments, the similarities and differences between Discussion and Annotation were tested.

5.3.1. Accuracy and Usefulness of indexing and STATBUILDER

In this Section, two experiments are described. In the first experiment, the accuracy of indexing was tested by trying to answer the following question:

> Given that a document D was assigned the index terms t_1 and t_2 by Salton's algorithm, and given that Mili's algorithm identified a relationship between t_1 and t_2, does D "talk" about a relationship between t_1 and t_2.

A positive answer would validate both the index terms generated by Salton's algorithm, and the relationships between them identified by Mili's algorithm. In the next experiment, the usefulness of both the index terms and the hierarchical semantic net was tested in recognizing the relationships between different groups of text blocks.

5.3.1.1. Experiment 1

This experiment tests the accuracy of indexing derived from a combination of Salton's algorithm and Mili's algorithm. A random sample of 20 paragraphs was used, and these were assigned index terms by Salton's algorithm, so that Mili's algorithm identified a relationship between two of the terms. For example, a document D with index terms $\{t_1,t_2,t_3\}$ such that STATBUILDER generated the relation $< t_3$ Broader-Term $t_1 >$ was a candidate for this experiment. The purpose of this experiment was to determine the extent to which D *truly* describes a relationship between t_3 and t_1.

The 20 paragraphs and their related pairs of index terms were given to experts who were familiar with the topic area. The subjects were asked to read each paragraph and to decide whether that paragraph described a relationship between the pair of index terms. The subjects found that 75 percent of the documents did indeed describe a relationship between the related pairs of index terms.

5.3.1.2. Experiment 2

This experiment tests the accuracy of Mili's algorithm and the index terms in recognizing the relationships between different groups of text blocks by trying to answer the following question:

> Given that three text blocks TB_1, TB_2, and TB_3 were assigned the index terms t_1, t_2, and t_3 respectively by Salton's algorithm, and given that Mili's algorithm identified a relationship between t_1 and t_2 only, do TB_1 and TB_2 exist in one Section and does TB_3 exist in a different Section.

A positive answer would validate the accuracy of both the index terms and the

derived semantic net in recognizing the relationships between the text blocks. The relationship between the text blocks can be drawn by applying the RELATION algorithm to the index terms and the hierarchical semantic net.

In this experiment frequency-based indexing algorithm was applied to 7 Sections (see Figure 28) which were selected randomly from different Chapters contained in Rada's book *Hypertext.*

Section Order Number	Text block
1	3, 12, 14
2	1, 2
3	5, 6, 7, 15
4	8, 9, 10, 13, 16, 17
5	11, 18, 20
6	4, 19, 21

Figure 28 "Sections correspondent text blocks": shows the Sections which are used in this experiment. Each Section consists of a group of text blocks. The text blocks numbers are the order of the text blocks as they appear in the directory.

The initial number of potential index terms was 44, with frequencies ranging from 1 to 4. STATBUILDER generated a two-level hierarchy, with 45 relationships; some of these relationships can be seen in Figure 29.

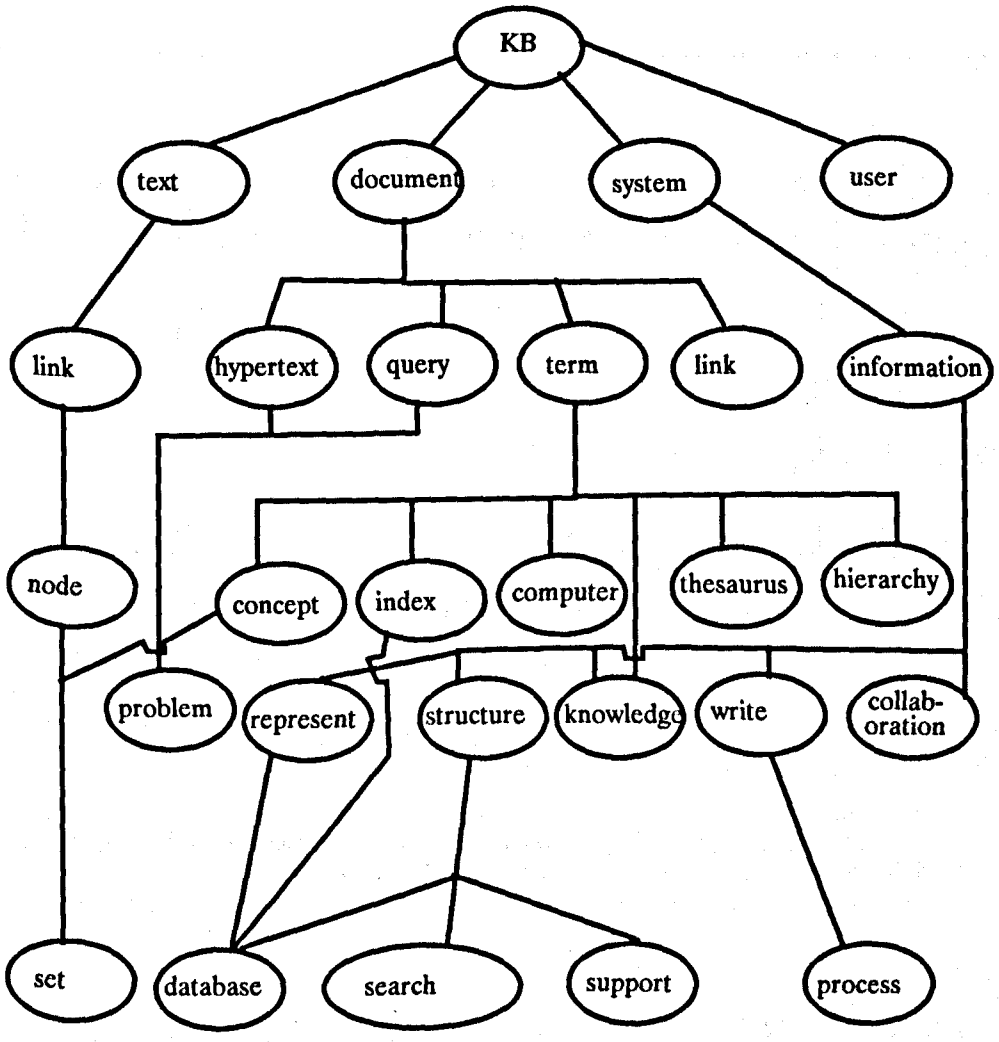


Figure 29 "Document hierarchy": Part of the text book hierarchy generated by STATBUILDER.

The RELATION algorithm was applied to these hierarchy relationships and the index terms. The result of this algorithm can be seen in Figure 30. In the comparison between the input as in Figure 28 and the output as in Figure 30, four Sections were completely recognized; the other two Sections were also recognized but one text block was disconnected from each one. Furthermore, four Sections were completely separated, while the other two were linked with each other through one relation only.

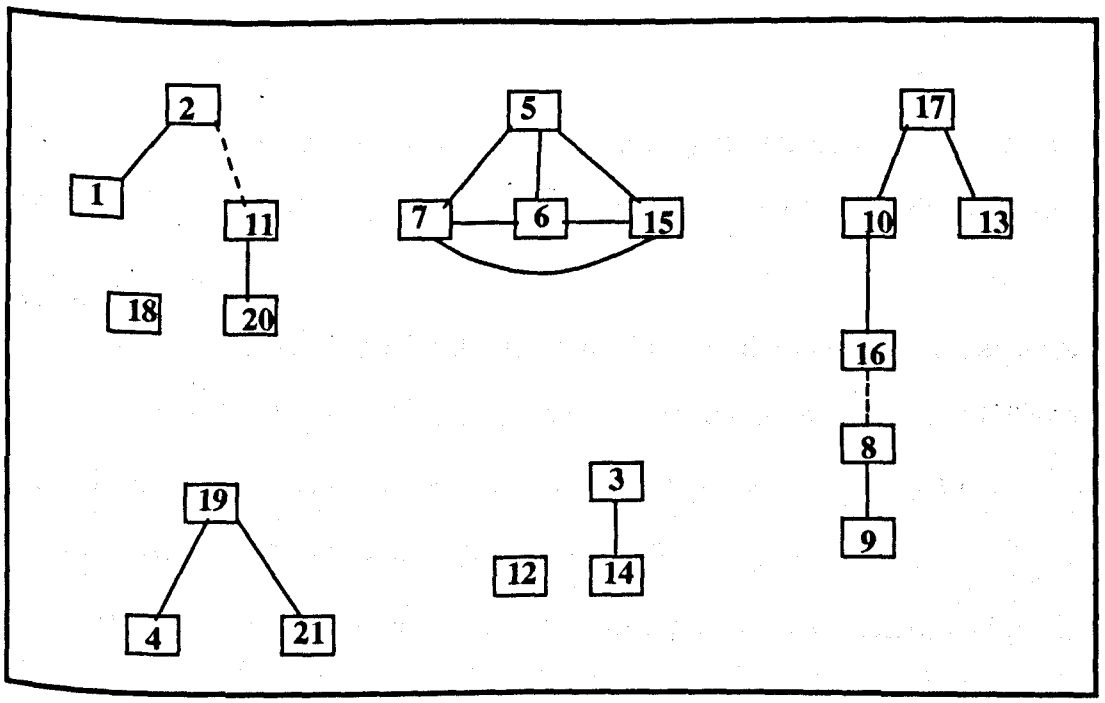


Figure 30 "Document relationships": the relationships between the text blocks of a document which are produced by applying the RELATION algorithm on the index terms and the hierarchical semantic net. The dashed line means only one relationship between the two text blocks. The solid line means more than one relationship.

5.3.2. Usefulness of Index Terms and Relationships Experiments

In this Section, two experiments have been described. In the next two experiments, the usefulness of the index terms and the relationships to support a variety of hypertext tasks were tested. In particular, the first experiment tests the value of flat index sets to measure the similarity between two documents. Such a measure may be used to support searching. The second experiment tests the usefulness of relationships generated by STATBUILDER to relate documents. Such relationships may be used for browsing.

5.3.2.1. Experiment 1

Searching involves concept matching. In keyword-based searching, the user enters one or more keywords, usually separated by logical connectives, and the system returns the set of documents that have those keywords in their indices. When none of the documents of the database completely matches the query (i.e., has all the required keywords), a retrieval system might rank the documents that match the query partially, by degree of match, and present them to the user in that order. Depending on the quality of indexing, that order may correspond to the order in which the user might like to consult the documents.

In this experiment, a variation on the above method was tested, and it explored the extent to which word-frequency based indexing could support similarity measurements between paragraphs. A similarity measure "SIMILAR" between two documents was defined as follows:

> Let doc_1 and doc_2 be two documents with index sets I_1 and I_2, respectively. The similarity between doc_1 and d_2 is measured by:
>
> $$\text{SIMILAR}\ (\ doc_1,\ doc_2\ ) = \frac{2 \times (\text{card}(\ I_1 \cap I_2\ ))}{\text{card}(\ I_1\ ) + \text{card}(\ I_2\ )},$$
>
> where card(X) is the cardinality of set X.

Note that this measure is normalized, i.e. $\text{SIMILAR}(doc_i,\ doc_j) = 1$ when $I_1 = I_2$, and less than one otherwise.

A random sample of 20 paragraphs, p_1 through p_{20} was selected, and computed $\text{SIMILAR}(p_i,p_j)$, for all $1 \leq i < j \leq 20$. The values ranged from 0 and 0.26. The same 20 paragraphs were given to subjects who were asked to read the paragraphs and to assign a measure of similarity between 0 and 1 to each pair of paragraphs. Thus, for the 20 paragraphs, the $\frac{20(20-1)}{2}$ numbers produced by SIMILAR were

compared to those produced by people using a correlation coefficient test[Alvo1985]

Let X = the similarity produced by the experts.

Let Y = the similarity produced by the machine.

$$r = \frac{\sum XY - \frac{(\sum X)(\sum Y)}{n}}{\left[\left(\sum X^2 - \frac{(\sum X)^2}{n} \right) \left(\sum Y^2 - \frac{(\sum Y)^2}{n} \right) \right]^{\frac{1}{2}}}$$

By substitution in the above formula, r = 0.45. Thus the correlation between the manual method and the automatic method is supported at the 5% level.

5.3.2.2. Experiment 2

In this experiment, it was hypothesized that an author with a set of index terms for a paragraph and the derived semantic net could find a related paragraph. To validate this hypothesis, a random sample was taken of 18 pairs of paragraphs (P_1,P_2) so that STATBUILDER identified a relationship between a term indexing P_1 and a term indexing P_2. These pairs were given to subjects familiar with the topic area, and they were asked to assess the relatedness of the paragraphs within a pair. The subjects found that 84 percent pairs of paragraphs were, indeed, related.

5.3.3. Similarities of Discussion and Annotation

The index terms and relationships which are generated by Salton and Mili's algorithms prove useful in recognizing the relationships between the index terms and the different text blocks. Also, they prove useful for browsing and searching. So, the index terms and the relationships have been used as a testing tool to test the similarity of the text for both Discussion and Annotation. In this Section an attempt to explore similarities of Discussion and Annotation was made. Two experiments

were performed. In the first experiment, the similarity between the text of the Discussion and the text of the Annotation was tested. In the second experiment, the similarity of the relationships between the Discussion text blocks and the relationships between the Annotation text blocks was tested.

5.3.3.1. Experiment 1

Discussion focuses on an unresolved issue, whereas Annotation points to a document. In other words, Discussion text blocks are more likely to be related to one another than are Annotation text blocks. To examine this point from a quantitative perspective, word-frequency based measures of relatedness are applied.

The text associated with each node in HERD can be indexed with some of the terms in the text. Given two index terms, $term_1$ and $term_2$, which frequently co-occur in the text associated with different nodes, it might be hypothesize that $term_1$ is related to $term_2$. Furthermore, it might be said that a node whose text is indexed with $term_1$ is related to a node whose text is indexed with $term_2$. The STAT-BUILDER algorithm extends the above approach by organizing a set of index terms into a hierarchy based on the frequency of co-occurrences of index terms in texts.Mili1987, Mhashi1990

Let x and y be ideas, terms, or concepts. Two nodes are related if one of the nodes contains x and the other contains x and y. The text of Annotation is similar to the text of Discussion, if there is no significant difference between the percentage of related nodes in the tree resulting from the Discussion and the percentage of related nodes in the tree from the Annotation.

The contents of the 55 Annotation nodes were examined. The size of each node ranged from one sentence to four paragraphs. Each node was characterized by

a set of index terms. The index terms were produced for these nodes manually. The number of terms in the final indexing vocabulary was 96, and the frequencies ranged from 1 to 5. STATBUILDER generated a two-level hierarchy containing 40 relationships. 34 pairs of nodes were suggested by STATBUILDER, where there was a relationship between the two nodes in each pair. All the nodes were given to subjects familiar with the topic area, and they were asked to assess the relatedness of the nodes. The meaning of relatedness was explained: Two nodes are related if they have something in common, such as both of them being concerned with an idea, concept or term. It was found that 29 pairs of nodes were, indeed, related to each other (i.e 53 percent of the nodes were related to each other).

The text associated with the first 29 Discussion nodes from case study one were also manually indexed. STATBUILDER was then applied to this collection and suggested that 26 pairs of nodes were related to each other. When the 26 pairs of nodes - in addition to the 3 nodes which STATBUILDER suggested were unrelated to any node - were given to the experts, they found that 26 pairs were, indeed, related to each other (i.e 89 percent of the nodes were related).

5.3.3.2. Experiment 2

In this Section two experiments were performed. The same procedure which was performed in Section 5.3.1.2, was repeated again here, but with regard to the text of Discussion and Annotation rather than the text of a document. An attempt was made to test the similarity between the relationships among the Discussion text blocks on one hand and the relationships between the Annotation text blocks on the other hand.

In the Discussion on requirement guidelines, 59 nodes were created. The

frequency-based indexing algorithm was applied to the first 29 nodes. The initial number of potential index terms was 66, with frequencies ranging from 1 to 9. Terms that occurred less than 2 times, were eliminated which reduced the set of index terms to 20 terms. STATBUILDER generated a three-level hierarchy, with 27 relationships; some of these relationships can be seen in Figure 31.

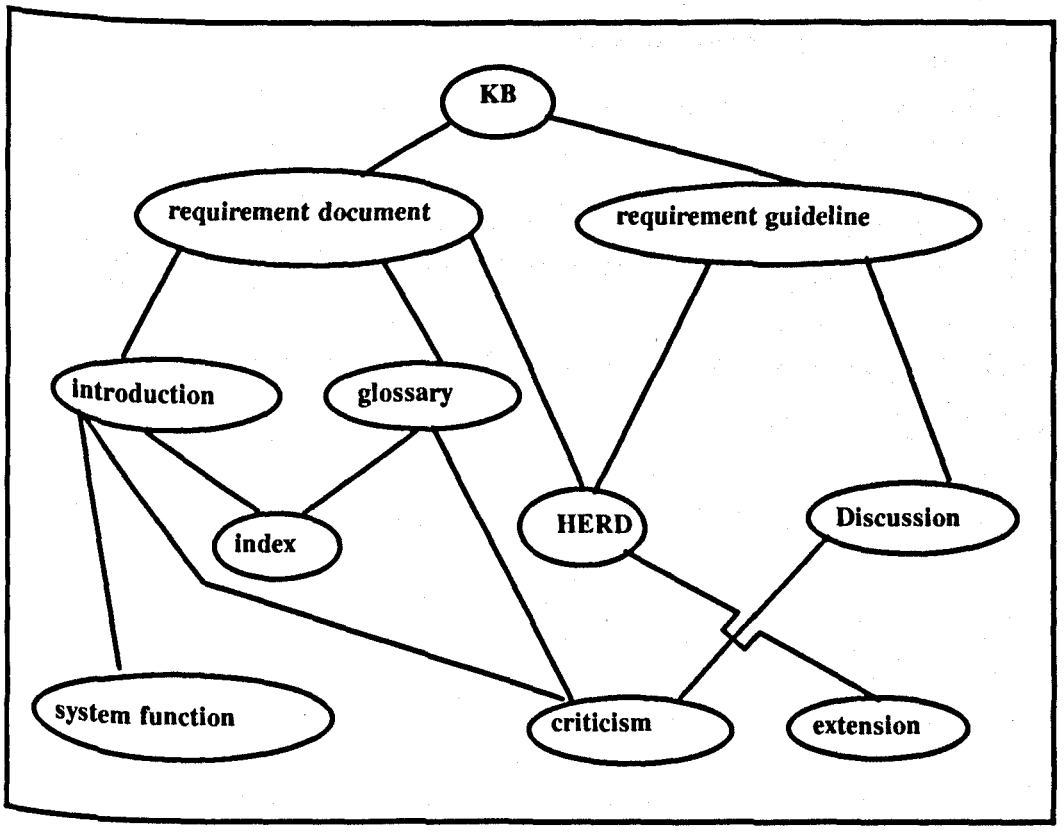


Figure 31 "Discussion hierarchy": Part of the Discussion hierarchy generated by STATBUILDER.

The RELATION algorithm was applied to these relationships and index terms. Three text blocks were completely disconnected. The remaining 26 text blocks were linked to each other through various relationships. A part of the graph can be seen in Figure 32. The number of relationships which connected the nodes ranged from 6 relationships to 22 relationships.

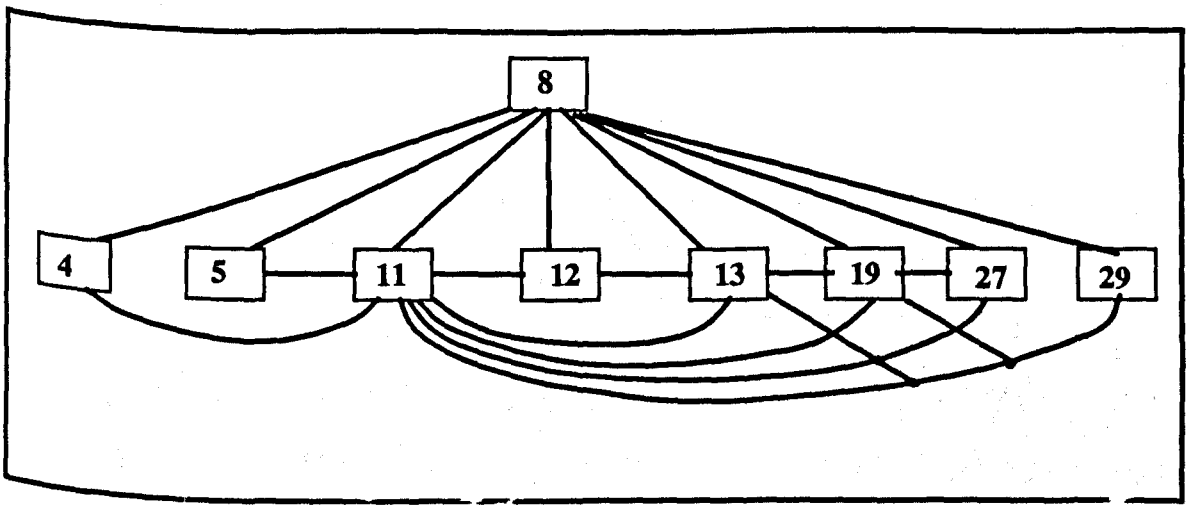


Figure 32 "Discussion relationships": shows part of the relationships between the text blocks of the Discussion which are produced by RELATION algorithm.

In an Annotation case study (Section 5.3.1), 55 nodes were created. The relationships between these text blocks as they were originally created can be seen in Figure 33.

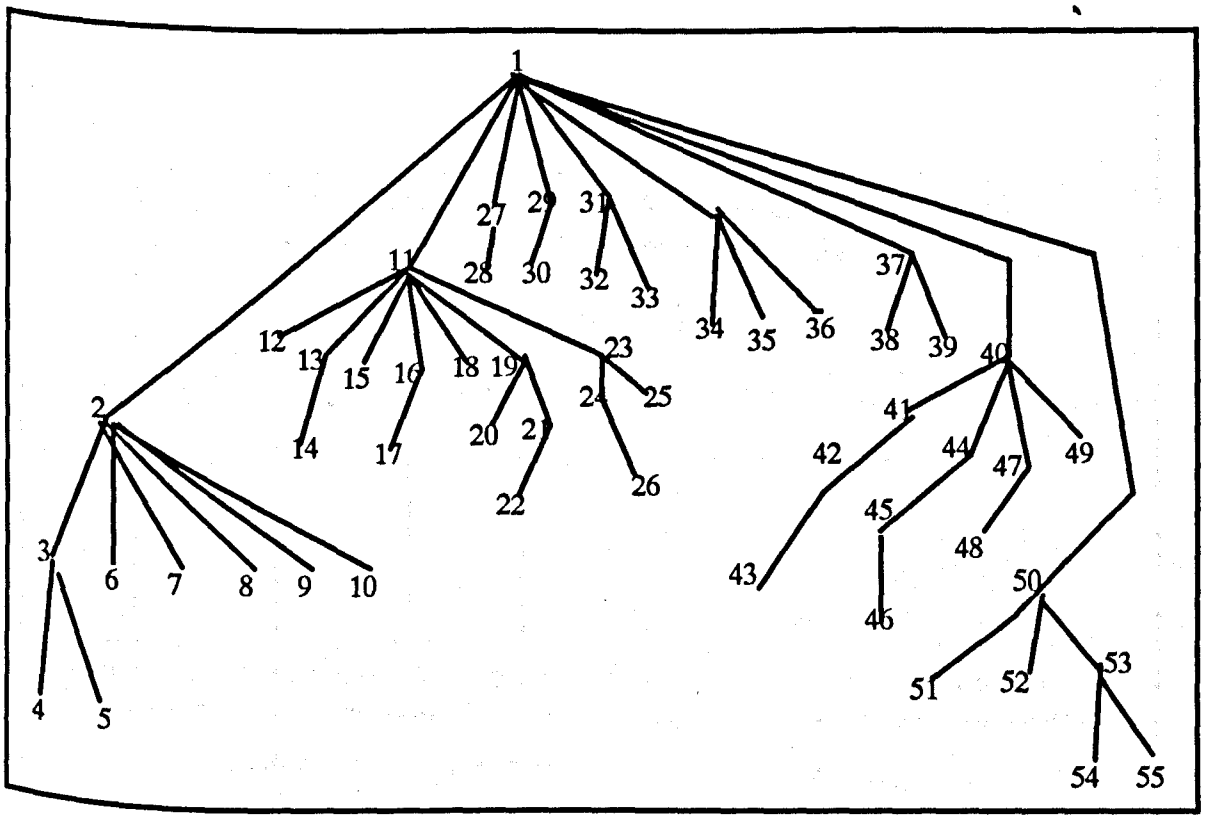


Figure 33 "Annotation tree": This shows how the relationships between the Annotation text blocks were originally created by the users.

The frequency-based indexing algorithm was applied to the 55 nodes which were created in the Annotation case study (Section 5.3.1). The initial number of potential index terms was 96, with frequencies ranging from 1 to 4. Terms that occurred less than 2 times, were eliminated which reduced the set of index terms to 23 terms. STATBUILDER generated a two-level hierarchy, with 39 relationships; some of these relationships can be seen in Figure 34.

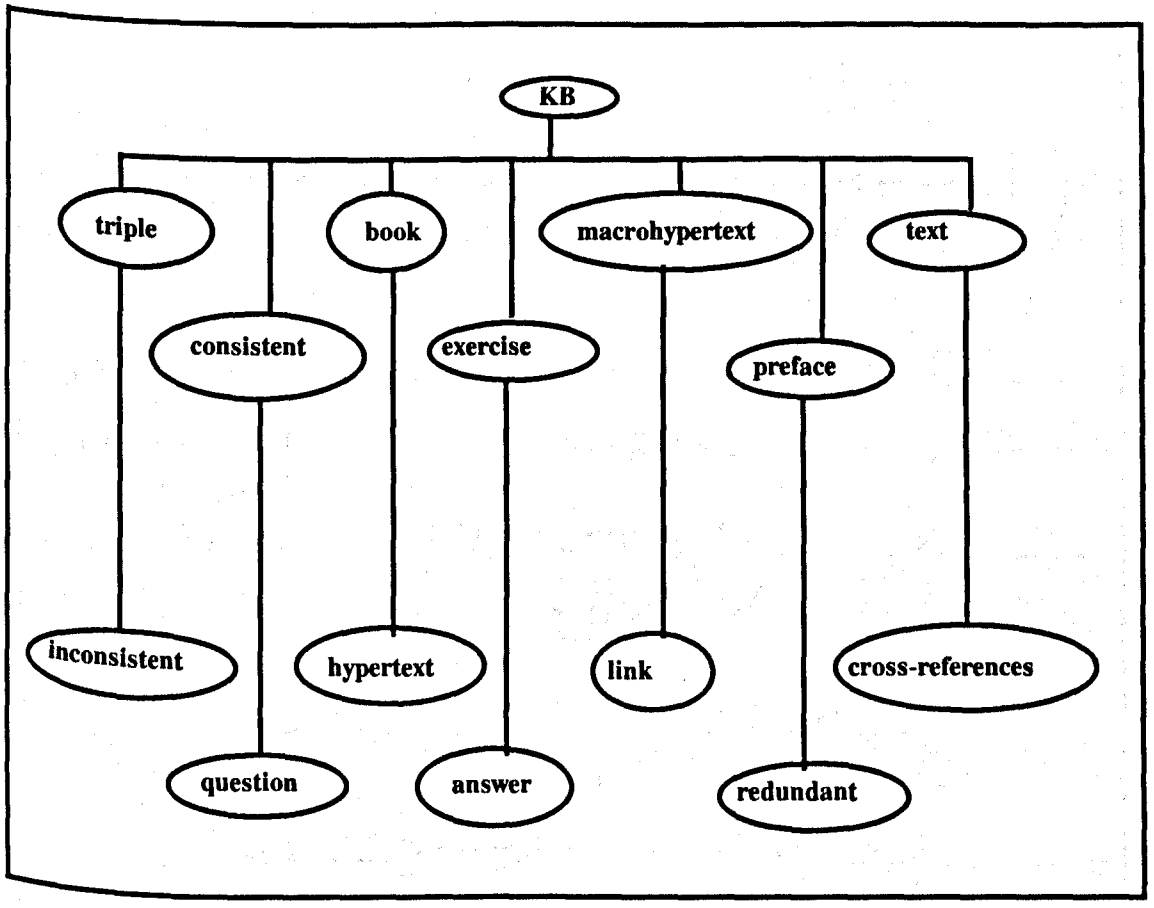


Figure 34 "Annotation hierarchy": Part of the Annotation hierarchy generated by STATBUILDER.

The RELATION algorithm was applied to these relationships and index terms. The relationships among the annotations can be seen in Figure 35. Thirty three annotations were completely disconnected. The remaining 22 annotations were divided into eight groups. The number of Annotation text blocks in these groups ranged from 3 to 5.

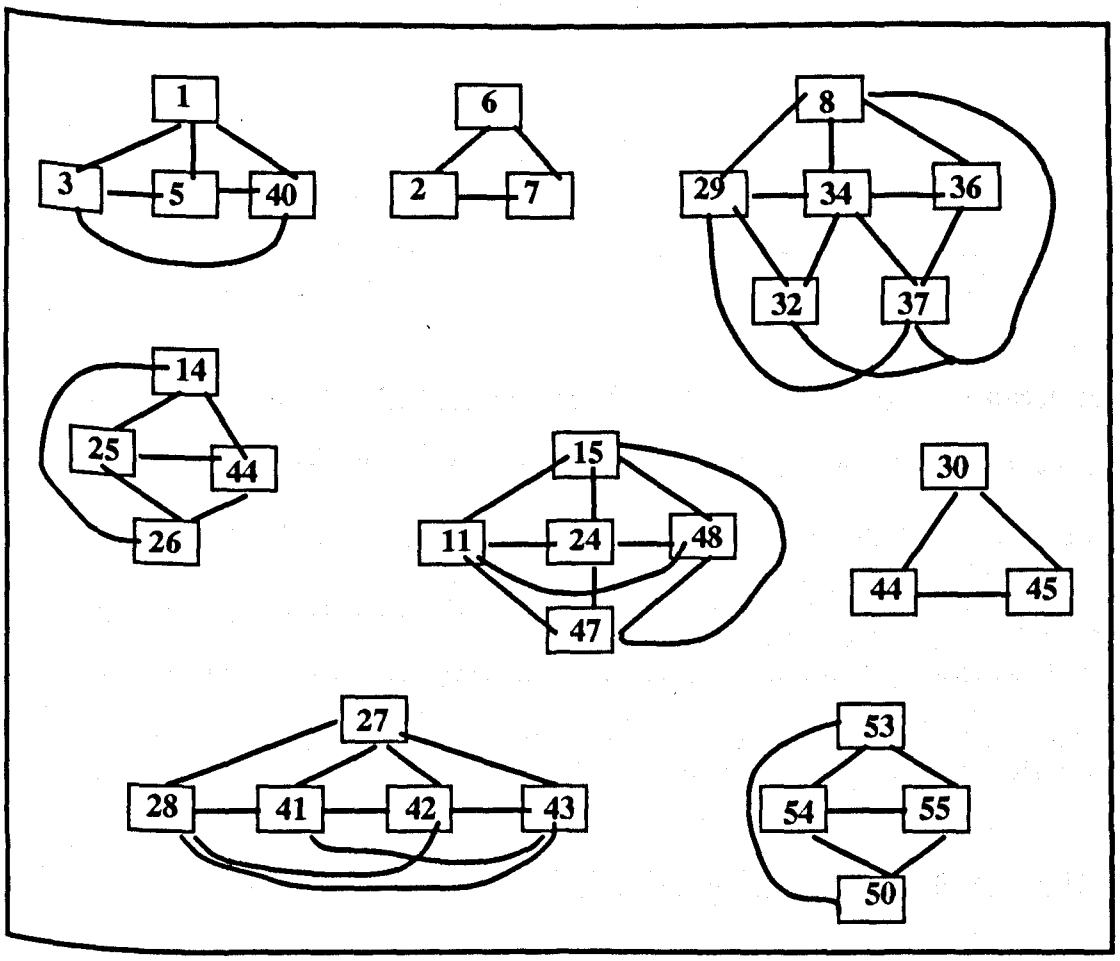


Figure 35 "Annotation relationships": shows the relationships between the text blocks of Annotation which are produced by RELATION algorithm.

CHAPTER 6

A Discussannotation Hypertext Model

6.1. Introduction

We have seen that both the literature (Chapter 2) and the previous experiments and Case Studies (Chapters 3 and 4) suggest that the Discussion and Annotation are important and needed in collaborative document writing. A Discussion supports the resolution of conceptual conflicts and the selection of conceptual and thematic alternatives. Once themes have been identified and initial drafts of the document written, comments or annotations on the document guide modifications that can be made.

Research on Discussion systems has taken many approaches. Fischer's work with JANUS[Fischer1989] has shown that Discussion and construction (or authoring) are two separate phases of collaborative activity. A knowledge base of rules was added to JANUS to help connect the information in a Discussion to the information contents of a document. Work with graphical Issue-Based Information Systems has shown the value of color as a tool to guide people in their understanding of Discussion.[Conklin1989] Experimentation with the COLAB Discussion system has shown that people need constraints to help them focus the Discussion.[Stefik1987] Studies of argumentation have shed light on the means by which Discussions can lead to conclusions.[Bernstein1989, Streitz1989]

Most of the Discussion systems do not support making annotations, whether these annotations are made with regard to the document or to the procedures and conventions for use in the system itself. In the issue-based information Discussion

systems, such annotations need to be treated as issues and their argumentations, but such annotations are very difficult to be classified as issues or arguments. Some systems, like KMS,[Aksc88] support making Annotation, but it does not support making an annotation on an annotation. If a user has created an annotation and another user wants to create an annotation objecting to the first one, then how can such a case be manipulated? It can therefore be seen that support for both Annotation and Discussion is needed.

The Colab 'idea organizer' divides a meeting into three phases: brainstorming, organizing, and evaluating. In the evaluating phase, participants review the structure of the linked ideas and eliminate peripheral ideas. The system prepares an outline by traversing the idea graph. Is the organization of the outline generated by traversing the idea graph always perfect? If a user wants to make some annotations on the generated outline, then how are such annotations represented?

In addition to these considerations, the Discussion systems ranged from supporting of node and link types such as gIBIS to none at all such as SYNVIEW and WE. The difficulty of using just three nodes and three link types was seen (Section 3.3.1). Furthermore, the Discussion systems which are supporting node and link types, are designed for Discussion, not for Annotation, and it was seen in Section 3.3.3 that the Discussion node types are not as useful for Annotation as they are for Discussion. It is suggested that while supporting the making of Annotation, the Discussion system should support Annotation node types as well as Discussion node types.

Considering another perspective, the InterNote Annotation system supports both review and revision processes and does not support Discussion making. However, there are many issues raised during the revision process which need Discussion. If there is a set of annotations created by different users on a specific term, then

which one of these annotations needs to be considered or incorporated? On the other hand, if one annotation has been made and the effect will be seen in more than one location in the document, how then can the users decide which locations need to be changed according to that annotation. So, how might such issues be resolved without a Discussion? One method for resolving such issues is to have a Discussion and then vote. Of course, the Discussion might be either a face-to-face Discussion or a Discussion system.

Some Annotation systems (such as Quilt) focus on making annotations only. In Quilt, creating an annotation on annotation is not supported. If there is some information in the document which has been written incorrectly and one user had created an annotation which leads to a Discussion (making an annotation on an annotation), then how can such a case be resolved? If the users want to make a plan about what they did or what they are going to do, then how can such a plan be made?

Furthermore, as in the Discussion systems, the Annotation systems are ranged from non-supporting Annotation node types such as PREP to systems which support some Annotation node types. The requirements Discussion Case Study (Section 3.3.1) proved that supporting node types in the Discussion systems are important. Thus, as in the Discussion systems, in order to make Annotation systems support a Discussion, Discussion node types should be supported as well as Annotation node types. As a result, the Discussion systems do not achieve their objectives without using Annotation. Also, the Annotation systems do not achieve their objectives without using Discussion.

Additionally, One of the major problems of human-computer interaction is the ever-growing complexity of software systems. The problem is compounded when users must cope with a multitude of systems that have inconsistent data models. Systems that support collaborative writing have a rich design space and present

designers with many agonizing tradeoffs.[Akscyn1988] In the implementation, designers may avoid facing up to these tradeoffs and thus permit the system to become too complex. This complexity affects the efficient use of the system. One way to reduce such complexity is to decrease the number of modes in the system. This can be done by combining two or more functions. This Chapter addresses the issue of combining support for Discussion and support for Annotation, which are currently available as separate activities, into a single integrated support facility.

In terms of supporting Discussion and Annotation on document creation, existing hypertext systems differ from one to another regarding the level of support for making Annotation and Discussion on a document creation. However, there are no hypertext systems which support Annotation and Discussion on all the different components of a document (such as Section, Chapter, or a whole document) and which generate different outlines. Nor is there any proposed system for the creation of Annotation and Discussion on system or administrative matters.

In summary, a model that combines Discussion and Annotation which is easy to design is suggested. The requirements for combining both Discussion and Annotation in one single system are described in Section 6.2. The model and a description of its main components: the storage level; the presentation level; and the storage-presentation specification, and the user interaction are presented in Section 6.3.

6.2. General Requirements

From both the literature (Chapter 2) and this work (Section 3.3.3 in Chapter 3, Section 4.3 in Chapter 4, and Section 5.3.3 in Chapter 5), many similarities and some differences between Discussion and Annotation have been found. Based on these similarities and differences, a set of requirements and designing issues to facilitate

the design of a single computer system for supporting both Annotation and Discussion are described. Thus, the aim of this Section is to explore the boundaries of the partly overlapping processes of Annotation and Discussion and to consider the implications of these processes for computer support. Section 6.2.1 identifies the high-level functional requirements. Section 6.2.1 presents the implementation requirements for such high-level functions.

6.2.1. Re-analysis of Previous Work

The results of the experiments and Case Studies that were performed in the Chapters 3, 4, and 5 suggest that the following are general requirement specifications for a Discussannotation model.

1) Both Discussion and Annotation activities should be supported.

2) The model should support a specific set of Annotation and Discussion types with the option for the user to modify (delete or add) them.

3) The model should support a specific set of node and link types with the option for the user to modify them.

4) Both hierarchical and non-hierarchical structures should be supported with the option to convert a non-hierarchical structure to a hierarchical structure.

5) Decision-making, different views, and browsing should be supported.

Such requirements need to be addressed in the design of such a model.

6.2.1.1. Discussion Handiling

In the document creation context, different Discussions need to be created. Discussions need to be created on documents, on the Annotations, with regard to the system itself, and with regard to the administrative matters (Section 3.3.3). So, the Discussion could be divided into two major types:

1) document Discussions: the Discussions which are intended to resolve some issues in the document and

2) non-document Discussions: these are Discussions which are intended to resolve issues which are not in the document. For example, they may include Discussions for making a plan about what should be done next, or who should do what, or Discussions about the system, or administrative matters.

Most hypertext systems have a variety of node and link types (such as IBIS, gIBIS, NoteCards).[Halasz1987] Some systems have only one type of node. For example, KMS has only one frame type, and is similar in that respect to HyperCard.[Akscyn1988] Sometimes, restricting the user to a set of node and link types causes difficulty for the user (Section 3.3.1). On the other hand, using specific types of node and links may be useful for automatic operations and intelligent processes (see Case Study 1 Section 3.3.1). Therefore, the system should support a specific set of node and link types with the option to modify the existing types and to add some more. From the experiment Section 3.4, it would appear that the following are useful link types: *summarizes, comments, decision-made, supports, suggests, criticizes, clarifies, objects, replaces, contains, extends*, and *answers*.

A collection of linked nodes may be considered as a graph which may be a hierarchical structure. Using hierarchy and decomposition is useful for Discussion. Some of the advantages of using hierarchy and decomposition are (see Case Study 1

Section 3.3.1):

1) it diminishes the disorientation problem (there was no restructuring or drawing of intermediate conclusions by summarizing);

2) it reduces the non-hierarchical links (one non-hierarchical link in the decomposition versus 11 non-hierarchical links in the non-decomposition part); and

3) it leads to a conclusion more quickly than other approaches do. All the nodes and the data which are created in the decomposition part are needed, while some nodes and data are unnecessary in the non-decomposition part.

The results of Case Study 1 (Section 3.3) suggest that the hierarchical decomposition structure is preferable to the non-hierarchical structure. It is also essential for making Annotation and Discussion to specify the boundaries of a group of text blocks (such as Sections, Chapters). However, the hierarchical structure does not satisfy all user needs in the storage nor on the display. Thus, supporting the hierarchical and non-hierarchical structures separately or in conjunction with each other on the display and in storage is necessary. Furthermore, understanding the problem and how it can be decomposed is a pre condition of using hierarchy and decomposition. One of the solutions might be to use a face-to-face discussion or Kunz's method.[Kunz1970]

The hierarchical structure may help focus attention and facilitate decision-making. There are many advantages for using decision-making in Discussion systems (see Case Study 1 Section 3.3.1). Two of these advantages are:

1) Assigning a decision to a node will be very useful in facilitating the reading process. All the issues will be clear, whether a specific issue has been resolved, discussed, or postponed. The participants can go directly to those *issues* which are not yet resolved, and they might be given an option to list all resolved or

unresolved *issues*. Unlike IBIS, and gIBIS, the users should try to find out whether an issue is resolved or not. This is done by finding either an aggregation of Issue-Position-Argument nodes, or the occurrence such of a "breakthrough" where there is no need for further Discussion.

2) Decision-making forces users to focus on a specific issue and encourages them to contribute as best they can. Also, it encourages users to participate in the Discussion. As a result, the conclusion will be more accurate and more powerful than having a Discussion without decision-making.

Finally, authors may want to modify the text which they have written. A good authoring system allows these changes while keeping a record of what was originally present. The strategies for version control are, however, not relevant to a Discussion system because the Discussion structure is intended to reflect the dynamics of the user interactions. To allow an author to change a node which had said "We should have peace" so that it then says "We should have war" would place any previously recorded responses in an incorrect context.

6.2.1.2. Annotation

The system should suggest a set of Annotation node and link types to the users. Most of the link types for Discussion are useful for Annotation, while the Discussion node types might be considered as a complement to the Annotation node types. The structure of the Annotation should be similar to the structure of the document and the user should be able to generate one outline for both the document and the Annotation (Case Study Section 4.3).

In document creation different Annotations need to be created on the document involved as well as the medium (see experiment 2 Section 4.5.2). The annota-

tions can be divided, according to their contents, into two broad major types:

1) Triple Annotation: the annotation which is pointing to a document, another annotation, or Discussion and the content of the Annotation also intended to change the corresponded node and

2) non-Triple Annotation: the annotation which is pointing to the Triple, but the content intended to change non-Triple activities such as annotations pertaining the system or the administrative matters.

Each one of these Annotation types, in turn, can be divided into a number of Annotation sub-types. For example, the annotation about document can be divided into *Annotation Organization* and *Annotation Content* node types. These Annotation node types approved to be useful for giving an author an indication of the good quality; good content and organization; of the document (experiment 1 Section 4.5.1). Therefore, the users should be given an option to reorganize the different entities in the outline by using the *Annotation Organization* node types.

A good hypertext system matches the structure in the database (hierarchy and non-hierarchy) and the structure on the display when the users want to view the data stored in the database. Non-hierarchy can be converted to a hierarchy by different methods (such as depth-first and breadth-first). Applying the *Organization Annotation node types* on an outline generated by one of the traversal methods was found to be a useful method (experiment 1 Section 4.5.1) in that it allows users to view the data in the storage according to one or more different perspectives.

Indicating which part of the document was being annotated is very important in the various methods of making annotations (experiment 2 Section 4.5.2). So, Annotations should point directly to the document. Annotator should be able to create an annotation and link it to any part of a document, such as a specific term, a region of

text, a text block, a group of text blocks, or the whole document (Section 4.5.2). Therefore, the system should support the *Composite nodes* method (described in Section 4.5.2).

6.2.1.3. Similarities of Discussion and Annotation

The electronic computer has opened wide new vistas for the collaborative work of writing a document. The links between the documents, between the people, and between the systems must be united in order to serve the collaborative work.[Rada1991b] One way to satisfy this is to combine Discussion and Annotation, which are both needed in collaborative work when writing a document. Collaborative work systems for document writing help groups of people create and access text in three phases.

1) The Discussion phase occurs first as people brainstorm and formulate plans as to how the writing should proceed.

2) In the authoring phase, blocks of text are attached to a network of ideas and the network is traversed to generate a document.

3) In the Annotation phase where the analogue of reading in the collaborative sense is the making of notes by a group of people working together on a document. This Annotation phase may also lead to a revised document as the annotators incorporate their comments into the original document.

Thus, the first common point between Discussion and Annotation is that both of them are essential activities in one area, which is the collaborative writing of a document. Annotations and Discussion need to be created on documents, with regard to the system itself, and with regard to the administrative matters. Annotations need to be created on Discussions, and Discussions need to be developed from

Annotations (experiment 2 Section 4.5.2).

Both Discussion and Annotation systems could be in a hypertext form such as the hypertext Discussion system gIBIS and the hypertext Annotation system Internote. So both Discussion and Annotation systems could have nodes and links. These nodes and links could have types, names, and identifiers. A problem can be resolved using a Discussion system,[Lowe1985, Smith1986, Hershey1985] or it might be resolved by using an Annotation system which uses argumentative annotations (i.e each annotation can be annotated).[Leland1988] A starting point is necessary for a Discussion as well as for an annotation. The starting point for a Discussion might be an issue or it might be an annotation to a document. When reviewing the contributions made by colleagues, a writer may want to know who has made what contribution over a period of time. Accordingly, the recording of author and date for each node in both Discussion and Annotation is important.[Rada1990b]

A node in many Discussion and Annotation systems is an arbitrarily large or small text block or other information, such as graphics. Systems such as KMS fix the size of a node to the amount of information which fits on the screen, but for Discussion or Annotation more flexible solutions might be appropriate. A limited menu of responses may be helpful in certain circumstances, for instance, a 'yes' or 'no' vote could be the contents of a node, or the contents might be empty (Case Study 1 Section 3.3.1 and experiment 2 Section 4.5.2).

Link types provide information about the nature of the relationship between two nodes and may support some automated processes. Some hypertext systems have a predefined set of link types such as HERD, but some let the user define the types (i.e typing was left open for the user) as does MUCH. A small, fixed set of link types may help users (Case Study 1 Section 3.3.1). The same set of link types were not enough and did not satisfy the users' needs (Case Study 3 Section 3.3.3).

In another Case Study, Section 4.3, a set of node and link types was suggested to the users, but they didn't have to use them. The results from this Case Study suggest that users find link types difficult to understand or to systematically apply. Thus, a balance between these two situations is necessary. In other words, the system should support a set of link types for both Discussion and Annotation with the option to add and modify the current types.

For creating or viewing an Annotation or Discussion node, the user should be able to see the source of the Annotation/Discussion and the Annotation/Discussion simultaneously. Regardless of whether he/she is making Annotation or Discussion on the hierarchy of a document (such as Sections or Chapters) the user should be able to see the outline for the corresponding part. The user should be able to either annotate an existing annotation or make a non-hierarchical link to or from an existing node in the document, Annotation, or Discussion (experiment 2 Section 4.5.2).

As shown in MUCH and HERD, the annotator or discussant should perform a number of steps to create a very simple node. Therefore, one of the major requirements for the annotator or discussant is that the creation of Annotations or Discussions should be a one step process. By making a selection and issuing a single command, an annotator or discussant has to be able to create a node or to enter his/her text.

The results of experiments 1 and 2 (Section 5.3.2) suggest that word-based indexing of text blocks (generated by Salton's algorithm) and the automatically derived hierarchical semantic net (generated by STATBUILDER) are useful for browsing and searching. This new method of browsing and searching is working well on the regular text (see experiment 1 and 2 Section 5.3.2) and the Discussion text (see experiment 1 Section 5.3.3.1). However, it is not useful on the Annotation text (see experiment 5.3.3.2). This is because the relationships between the Anootation

text blocks are not strong enough. Therefore, this method and the similar methods of browsing and searching could be applied only on the regular text and Discussion text only.

6.2.2. User Environment for the Model

One of the requirements for the user who has a write-access is the ability to edit the document without making annotations, such as spelling correction or sentence rewording. This might, however, cause a side effect. For example, if the user is updating a text block in the document, Discussion, or Annotation, at the same time as another user is annotating it, the first user may end up with annotations and corrections that are out-of-place. This might happen when making good/bad organization annotations on the outline. If two users create two "good organization" Annotation types on different entities in the outline, one of them might end up with entities on an out-of-place outline. In such cases, the system has to be able to resolve these types of edit/annotate collisions and allow multiple users to annotate the text block or outline simultaneously.

The users have to be able to view a document and its outline simultaneously from different perspectives. Therefore, a user should be able to change the organization of the document, but these changes should be stored as information on the link and should not effect the original organization of the document. In such a case, users should not be able to change the text according to individual changing of the organization. Users also should be able to make comments of themselves and make a copy for some parts of the document. For example, if a reader wants to include a quotation from that document in his/her work, then he/she should be able to copy that quotation rather than retype it.

Users have to be able sort Annotation or Discussion nodes so that they could view one user's Annotation or Discussion nodes at a time, view Annotation or Discussion nodes by date, according to their types, by the hierarchy of the document, or view only the annotations that had not yet been incorporated or the issues that had not yet been resolved. Users such as authors have to be able to merge the Annotation or Discussion so that they could be viewed in the order in which they appear in the document, to incorporate suggested changes to the document, and to keep track of which annotations had been incorporated and which issues had been resolved. Furthermore, authors should be able to delete some Annotations or Discussions nodes, delete the reference to the Annotation and Discussions, but still save it somewhere.

When viewing these Annotation and Discussion types, a user might want to view each one of these types separately, or in conjunction with the document. Therefore, the method for creating and viewing Annotation and Discussion types had to be identical across all applications. The users, therefore, need to learn a set of features which are the same for the document, Annotation, or Discussion. The developers should also be able to write new applications which can be applied to document, Discussion, or Annotation.

Manipulating the text and the heading separately will make this model more efficient than other models in some aspects and as efficient as others in some other aspects. If it is assumed that there are some hypertext systems and models that support or can support text and headings creation, then the comparison here will be between such models and this Discussannotation model. The set of functions in this model is not the same as the set of functions in other models. The primary operations for creating a text block in this model are: create node, create link, and create text block. On the other hand, in the other systems, the primary operations are:

create node and create link. Create node, however, includes the two attributes: heading attribute and text attribute. Create node of course, is dependent on the phase (such as create node in Annotation means create Annotation node, and so on).

For simplicity, the task can be divided into sub-tasks. The set of sub-tasks includes: creating new text blocks, adding new text blocks with headings which are the same as some of those already created, creating headings without text, creating Annotations and Discussion on headings, creating Annotations and Discussion on a specific term or a span in a text block, creating Annotations and Discussion on a text block, creating Annotations and Discussion on a group of text blocks, and generating an outline. Since the Discussion is also composed of headings and text, then all the sub-tasks for making Annotation on the different components of the text might need to be made on the Discussion as well.

For making a primary operation in this model, it can be assumed that a user needs two key-strokes for each operation, one to select the function and one to store the data. In the other systems, it can be assumed that a user needs three key-strokes to creating a node; one to select the function, one to store the heading attribute value, and one to store the text attribute value, even if no text is entered. To create a link, two key-strokes are needed which is the same as in this model.

To do the task by using this model, the following key-strokes are needed: 6*(d+A+D) key-strokes are needed for creating new text blocks and new headings. Where 6 is the number of key-strokes which are needed to create one text block and 'd', 'A', and 'D' are the number of new text blocks which need to be created in a document (such as a paper, a book, or a report), Annotation, and Discussion respectively. 4*(d'+A'+D') key-strokes are needed for adding new text blocks with reuse of some of the target headings which have already been created, where 4 is the

number of key-strokes which are needed to add one text block, and d', A', and D' are the number of text blocks which need to be added to the structure in a document, Annotation, and Discussion respectively. To illustrate this point, it can be assumed the two headings 'DOC1' and 'DOC2' were created. Also, the heading 'Abstract' was created and connected with the heading 'DOC1'. Now, if the abstract needs to be created for 'DOC2', then a link needs to be created between 'Abstract' and 'DOC2' (2 key-strokes), and 2 key-strokes for the text.

4*(d"+A"+D") key-strokes are needed to create new headings without text (such as Chapter headings), where 4 is the number of key-strokes which are needed to create a new heading, and d", A", and D" are the number of headings which need to be added to the structure in a document, Annotation, and Discussion respectively. 2*(d'"+A'"+D'") key-strokes are needed for reuse of headings which already exist by linking these headings with different nodes (where 2 is the number of key-strokes which are needed to link a heading with another heading, and d'", A'", and D'" are the number of headings which need to be linked in a document, Annotation, and Discussion respectively).

To do the task using other models the following key-strokes are needed: 5*(d+A+D) key-strokes are needed for creating new text blocks. 5*(d'+A'+D') key-strokes are needed for adding new text blocks with headings which are the same as some of those already created. 5*(d"+A"+D") key-strokes are needed for creating headings without text. Headings can not be re-used and new nodes need to be created. So, 5*(d'"+A'"+D'") key-strokes are needed for creating headings even though they have been created before, because the text is different.

So to do the task by using this model, the total number of key-strokes is:

6*(d+A+D)+4*(d'+A'+D'+d"+A"+D")+2*(d'"+A'"+D'") key-strokes.

The total number of key-strokes which is needed to do the task by using the other models is:

$$5*(d+A+D+d'+A'+D'+d''+A''+D''+d'''+A'''+D''') \text{ key-strokes.}$$

One of the differences between the two models is the number of key-strokes which are needed for each individual sub-task. The number of key-strokes for each sub-task is unknown. So, it cannot be decided whether this model is more efficient than the others or vice versa in performing the whole task. On the other hand, a prediction can be made as to whether this model is more efficient or not in a specific sub-task or a group of sub-tasks according to the number of key-strokes which are needed for each function. For example, this model is less efficient than the other models in creating a set of new text blocks and new headings ($6*(d+A+D)$ vs. $5*(d+A+D)$), while this model is more efficient when the the number of Annotation and Discussion nodes which need to be created are greater than the number of document nodes, such as for novice users.

In Discussions and Annotations, most of the headings can be reused such as 'Bad-Content' node Annotation which can be linked with all Sections that have bad content, whatever the number is. In the Annotation also, sometimes both the heading and the text can be reused. Furthermore, in the case of working on more than one document at a time where the headings of the documents are similar (for example, almost all papers have title, authors, abstract, introduction, literature review, experiments, Discussion, conclusion, and references), such headings can be re-used. In such cases, this model is more efficient than the other models. However, there are many factors, suggesting that Discussion and Annotation are very important and are needed for document creation. Examples of this are the different existing hypertext systems which have been created especially for creating Discussion, for making

Annotations, for supporting both Discussion and Annotation, and the experience of this work as well. In Section 3.5.2, the number of Annotation and Discussion headings which are created on a document are much more than the number of the document headings itself. This indicates that this model is more efficient than the other models. However, in the case of creating a document with few Annotations and Discussions, such as in the case of writing a document by experts, the other models are more efficient than this model.

6.3. Practical Issues

The proposed Discussannotation hypertext model is a computer-based tool that supports both Discussion and Annotation in the creation of documents by one or several users. Discussion is a set of link objects. Each link object has a node and link. A link may be attached to it a text block. Each node has a label such as *Issue, Position*, and *Argument*. If the node in a link object is an *Issue*, then the text block that is attached to that link is text in form of a question. If the label is a *Position*, then the text block that is attached to that link is some text written in a way to answer an *Issue*. If the label is an *Argument*, then the block of text that is attached to that link is some text written in a way to support or to object to a *Position* or another *Argument*.

Annotation is a set of link objects that are connected to the Triple. Each node in an annotation link object has a label. A text block that is attached to an annotation link object is created for many purposes such as:

1) to question, clarify, critique, or to suggest changes to a text block, a particular term in a text block, a group of text blocks, or a heading in the document and/or in the discussion;

2) to suggest changes to the system; or

3) to comment on the administrative matters.

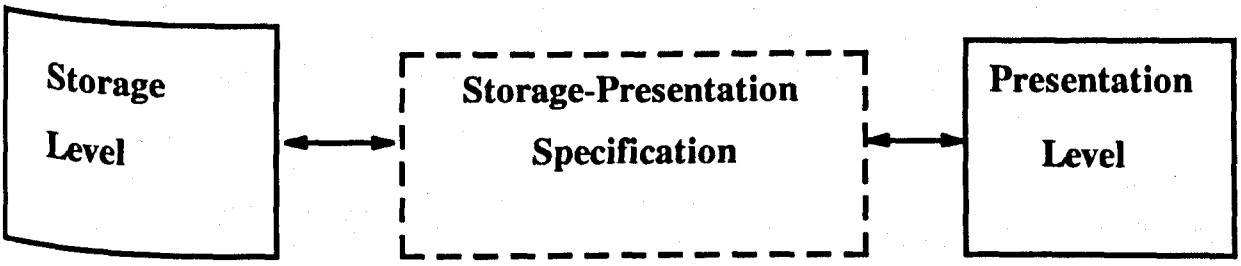


Figure 36 "Discussannotation Levels": Levels of the Discussannotation Model

The Discussannotation hypertext model divides a system into two major levels, the *Storage* level and the *Presentation* level, as illustrated in Figure 36. The storage level is concerned with the text of documents, the text of annotations, and the text of Discussions. The *Presentation* level is concerned with how the Triple in the storage should be presented to a user on the screen. The interface between the *Storage* and the *Presentation* levels is accomplished by the *Storage-Presentation Specification.*

The storage level describes a database that is composed of a hierarchy and non-hierarchy of nodes which are interconnected by relational links. The term node corresponds here to various alternative terms used in existing hypertext systems and models: *frame* in KMS, *node* in gIBIS and IBIS, *card* in NoteCards and HyperCards, *component* in Dexter, *document* in Augment and Intermedia, and *article* in Hyperties. In most of the hypertext systems and models, nodes contain chunks of text, graphics, images, and animations, and constitute the basic content of the hypertext network.Halasz1990 This is one of the main differences between this model and the other models, in that the chunks of text is attached to the link, not to the node. Thus, in the Discussannotation model, the heading and the text can be created separately. While in the others, each time a node is created, the heading and the text attributes should be created together.

The storage level focuses on the mechanisms by which the nodes and links are glued together to form hypertext networks for the Triple. The relationship between these different activities is illustrated in Figure 37. A user might start creating a document or a Discussion to determine the goals or make plans. After creating a part of a document, annotations can be created on the different parts of the document. Such annotations might be developed for a Discussion.

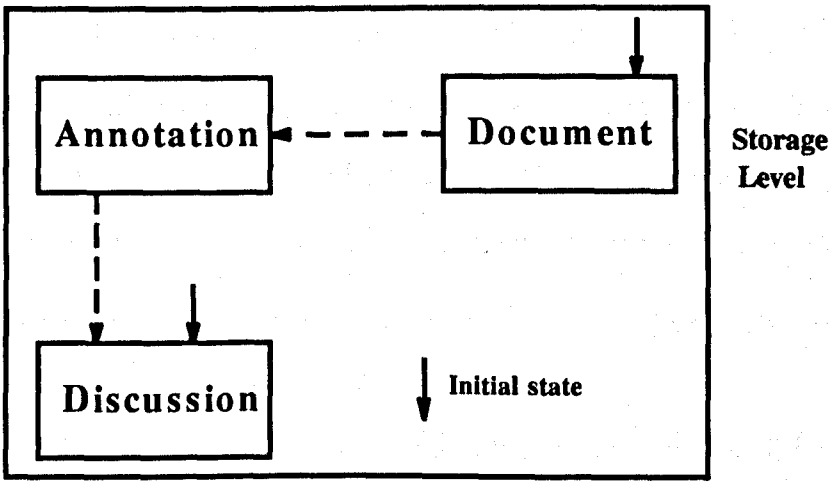


Figure 37 "Activities Relationships": The relationship among the different activities in the storage level can be started by creating a document, make Annotation, then make a Discussion. Another method is to start creating a Discussion for making a plan.

The Triple which is stored in the database hierarchically and non-hierarchically, needs to be presented on the screen according to the user's need. A user might want to view a subset of document, Discussion, and Annotation non-hierarchically or hierarchically in different ways according to different perspectives, or in terms of a specific attribute such as author, date, or version. In order to specify in which way the hypertext network should be presented to a user, the presentation level needs to access the information in the storage-presentation specification.

The storage-presentation specification focuses on providing information for:

1) making Annotations and Discussions on a group of text blocks and on a specific

region or term within a text block; and

2) viewing the Triple in the database hierarchically on the screen.

The Annotations and Discussions on a group of text blocks can be implemented by a method called *Composition*, while Annotations and Discussions on a specific term can be implemented by a method called *anchoring*. These two methods are different from *composition* and *anchoring* as they described in Dexter model. A description for these two methods will be given in Section 6.3.3.

Both the presentation level and the storage-presentation specification are very important for different reasons. One of these major reasons is that the documents that are stored hierarchically and non-hierarchically, are dynamic and unspecified (i.e the Sections, Chapters, and the starting nodes are unknown). Thus, in order to produce a linear document (report, paper, book), viewing such documents hierarchically is necessary for making Annotations and Discussions on the different components of a document. If a user wants to make an annotation on a Section or Chapter, then the boundaries of that Section or Chapter should be both determined and known.

Since such boundaries are unknown in the hypertext systems storage, there should then exist some methods to determine the boundaries and present them on the screen to the users. The Triple can be viewed hierarchically on the screen as the storage-presentation specification provides the necessary information about the boundaries of the different components of the document. Then, Annotation and Discussion can be made easily on a specific group of text blocks.

6.3.1. The Storage Issue

The storage level is at the bottom of the different levels of the model and it deals with all the traditional issues of information storage. It is necessary to store large amount of information on various computer storage devices like hard disks or optical disks. Furthermore, the database level should handle other traditional database issues, like multi-user access to the information and various security considerations.

The storage describes the structure of a hypertext as a semantic net that is composed of a finite set of link objects where each link object's functions are:

1) to point to a node or a set of nodes that contain source or target terms,

2) to point to a text block or a group of text blocks, and

3) to point to an individual term, word, or region in a text block.

Furthermore, each link object has several attribute-value pairs, including attributes for author, date of creation, link-type, and phase. In this Section, some issues have been addressed including: How can a document text block be created? How can a Discussion text block can be created? How can an Annotation text block can be created? How a Discussion and Annotation can be made on headings?

The storage focuses on creating a text of a *document, Annotation* and *Discussion* and their headings. A small set of operations has been defined in order to create a text block or heading. For creating Annotations and Discussions on a group of text blocks, headings, or on a specific term or region within a text block, the boundaries for such a group and some necessary information, which cannot be determined in the storage only, need to be specified. Such information and boundaries will be discussed in the next Sections.

The set of operations can be classified into two groups: 1) operations within the activities (document, Annotation, and Discussion) such as creating a document text block and 2) operations among the activities such as making Discussion on an annotation or making annotation on the document. As mentioned earlier, in Section 4.4, the Annotation node types regarding the documents were classified into two groups: Annotation node types about the content of the document and Annotation node types about the organization of the document. Any Discussion node connected to an annotation node will be considered as an initiation for a Discussion. In such a case, such a Discussion node type should be an issue.

The set of operations within the activities are the following:

1) Creating a text block for a document: to create a text block, a node should be created first (such as 'abstract' in Figure 38. Then, that node should be connected to the existing semantic net by creating a link from a source node (such as 'Annotation-and-Discussion') to the destination node (such as 'abstract'). A text block can be created after specifying the source node, the destination node, and the link between both of them. In MUCH, this mechanism is referred to as "link object". One of the advantages of attaching the text to the link, as illustrated in Figure 38, is that a node (such as 'abstract') that is related to more than one node will be created only once. Of course, information such as date, time, author name, and access privileges can be taken automatically by the computer, and users don not have to enter such information. In the document, a node has a name but it has no type. A link has a label, and a direction (determined by specifying the source and the destination) but no type.

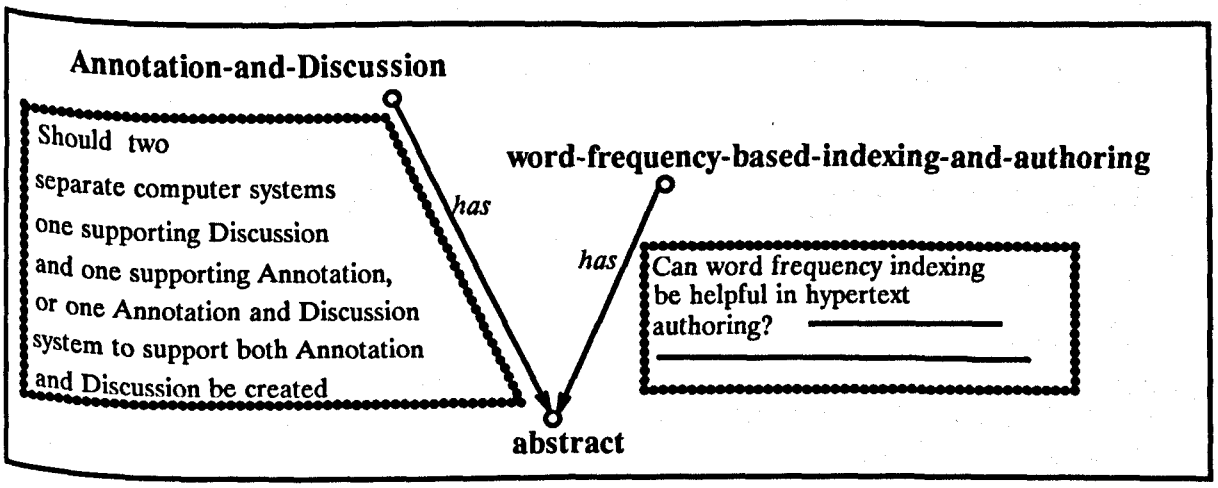


Figure 38 "Document Text Block": illustration of creating a text block.

2) Creating a text block for Discussion: to create a text block for a Discussion, the 'link object' mechanism is used in a way similar to the way it is used in creating a text block for a document. The nodes in the Discussion have a type and a name. A node name may be used as a name, a heading, and a unique identifier. A link has a type that depends on the type of the node. For example, if the link type is *generates*, the destination node is an *issue*. The Discussion types (see Section 4.5.2) should be specified by the user.

3) Creating a text block for Annotation: to create a text block for Annotation, the 'link object' mechanism will be used in the same way as it is in the document and Discussion. A node in the Annotation is similar to a node in the Discussion in that it has a type and a unique identifier. The node types in the Annotation are different from the node types in the Discussion. A link has a type. Most of the link types can be used for both Discussion and Annotation (such as *suggests* and *responds* for more detail regard the link types see Section 2.5). The Annotation types (see Section 4.5.2) should be specified by the users.

The set of operations among the different activities are the following:

1) Making Discussion and Annotation on a heading: the only operation that can be made among the different activities without any need for specifying boundaries is making Annotation and Discussion on a heading. For making an Annotation on a heading, the 'link object' mechanism will be used, in the same way as it is in the previous operations (i.e within activities operations). The node type for Annotation, however, should be one of the content Annotation node types. Then a Discussion can be created by creating an *Issue* (see Figure 39).

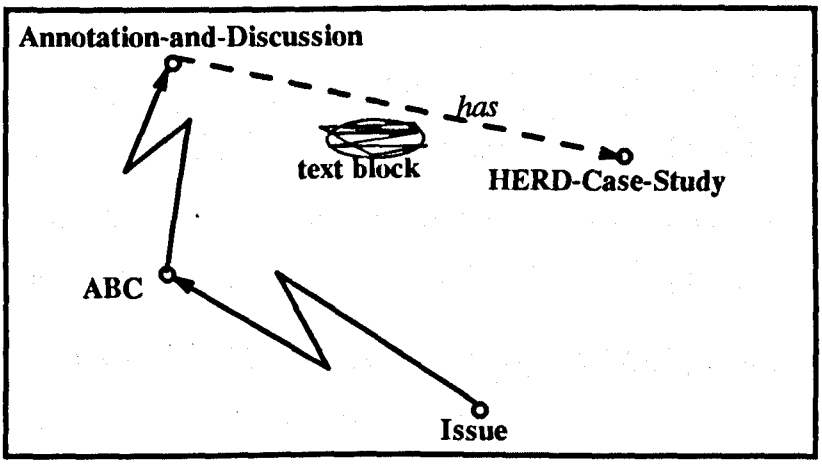


Figure 39 "Discussion and Annotation on heading": illustration of making Annotation and Discussion on a heading. ABC means an Annotation node about content with the type 'Bad-Organization'.

2) Making an Annotation and a Discussion on one or more text blocks: In order to make Annotations and Discussions on a group of text blocks, the boundaries for that group should be defined (see Figure 41). Then the *composition* method should be applied (see Figure 42). The methods of making Annotations and Discussion on the content or on the organization of a group of text blocks are similar, but in the first, the Annotation node types should be about the text itself (such as Section 'A' should be re-written).

3) Making Annotation and Discussion on a region or on a specific term within a text block: creating Annotation and Discussion on a region or on a specific

term within a text block can be made by using a method called *anchoring* (described in Section 6.3.3). The text block should be specified and presented on the screen. A text block can be specified by determining the link to which that text block is attached. A link can be specified by determining the source and target nodes, as illustrated in Figure 39. One further step is needed which is specifying the term or the region after displaying the text block on the screen.

6.3.2. The Presentation Issue

The Triple is stored in the database hierarchically and non-hierarchically. The Triple needs to be presented on the screen according to the users' needs for the review and revision processes. The users could then make a response by reviewing the Triple, making Discussion, making Annotations, or adding some text to the document. The Triple should be presented on the screen in a way that facilitates the reading process and satisfies the users' needs.

The users may want to view the Triple hierarchically or non-hierarchically. For example, in order to make a decision about a specific matter, users might need to view the Discussion about that issue hierarchically. They might also need to view the relationship between the different nodes of the Discussion (i.e non-hierarchy), so that they can understand the resolution better. This might affect their decision-making. The users want to view the Triple hierarchically according to a specific factor or a combination of different factors such as:

1) Type: users want to view the Discussions/Annotations which are created on the document, the medium, or administrative matters separately or as a combination of these,

2) Logic: users want to view the Triple according to their logic. For example, a

user wants to view the Discussions/Annotations made, first on the whole document, then Chapters, then on Sections (i.e start from the most important to the least important),

3) Version: users want to view Discussions/Annotations on a specific version of the document during its life cycle development,

4) Date and user importance: a user wants to view the most recent Discussions/Annotations which have been created by the most important user, in order to make a response. The most important user also wants to evaluate who made a contribution and what contribution over a specific period of time, and

5) Perspective: a user wants to view the Triple according to his/her own perspective which is different from others' perspectives. Also, a user wants to view the Triple according to the most important user perspective.

The question might be asked: How could the Triple activities in the storage which are stored hierarchically and non-hierarchically be presented on the screen according to the user perspective and desire? The answer for such a question will be found in the next procedure. This procedure will show: 1) how a graph will be converted to a hierarchy of outline and 2) how the hierarchy of outline can be presented on the screen according to the user's desire and perspective.

1) **Converting a graph to a hierarchy of outline:** *depth-first* and *breadth-first* are two main traversal techniques for converting a graph to an outline.[Ghaoui1991] The algorithms can be described as follows and Figure 40 shows an example traversal of a simple graph. The first algorithm searches through the graph (via the nodes) in a depth-first manner. Once a node has been arrived at, it will be printed in the output. Then the search will continue to the next node in a depth-first manner. The

number of levels will be increased by one which can be represented by indenting the output to the right few spaces. From Figure 40, the search would start at the node **A**. Node **A** will be printed, then the depth-first takes over and moves to node **B**. Node **B** will be printed in the output, intended a few spaces, and the process will continue until the last child, which is **E** in the graph. This will be followed by back-tracking and further searching visits of all of other nodes in the graph.

The second algorithm searches through the graph in breadth-first manner. Hence, in Figure 40, the search starts at **A** and it will be printed in the output. Then the search will go to the node **B** and backtracking to the next sibling until all children will be printed out. After printing all the siblings, the algorithm will move in a depth-first manner which is node **E** in the Figure. Then it backtracks again for all the siblings.

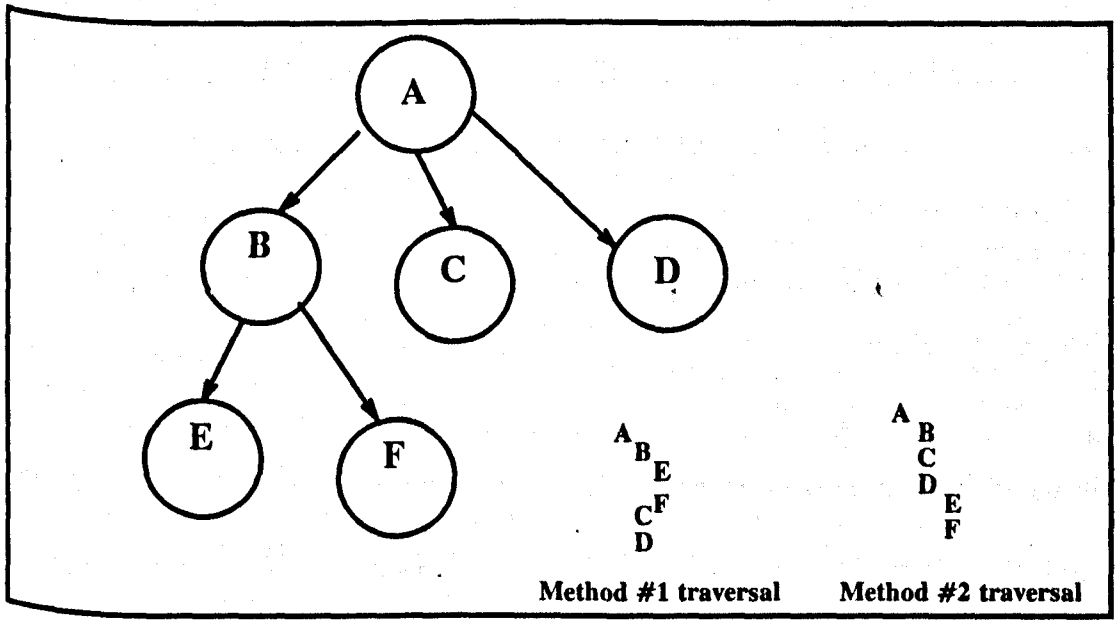


Figure 40 "Traversal": shows how a graph can be converted to an outline by using depth-first (method #1 traversal) or breadth-first (method #2 traversal) techniques.

Looking to the Figure 40, we see that each one of these outlines is a traversal for the whole graph. In our case, the whole graph contains data for the document

and for the various types of Discussion and Annotation. The question might be asked: Is it possible to view part of such a graph? Such a question leeds to the second step in this procedure. Furthermore, in the Figure two different outlines which are traversed from one graph can be seen. Assuming that this graph is a part of a bigger graph, a user might then want to view this part of the graph as in the outline which is traversed by the first technique. Another user might want to view the graph as in the outline which is traversed by the second technique. Others might want to view the graph in a way which is different from both outlines. Is it possible to satisfy all users? This question leeds to the third step in this procedure.

2) **Hierarchy of outline and user's desire:** In the previous step, an outline can be traversed for the whole graph which might consist of different data such as data for a document and various types of Discussions and Annotations. One user may want to view the document, the Discussion, or the Annotation only. Other users may want to view the Discussions made regarding the medium or the Discussions which are developed from Annotations only. In order to satisfy such desires, the algorithm which traverses the graph should access the information associated with the links and the nodes before printing out the nodes in the output. If the information which is associated with that node satisfies the user desire, then that node will be printed in the output, otherwise it will be skipped. Thus, the result at the end of this step is an outline according to the user's desire. However none of them might satisfy the user's perspective (i.e the organization of the outline should be modified).

3) **Hierarchy of outline and user's perspective:** In the previous two steps, a graph can be traversed into an outline by one of the described techniques and according to the user's desire. In order to satisfy the user's perspective the outline should be re-organized. By using the "Annotation-Organization-Node-Types" algorithm which is described in Section 4.5.1, a user will be given an option to move an entity in the

outline from one place to another. Applying that algorithm to the outline produced from the previous two steps, causes that outline to be in accord with the user's perspective.

6.3.3. The Storage and Presentation Specification

In document creation, Annotation and Discussion on a group of text blocks are needed. A group of text blocks might be a sub-Section, Section, Chapter, document, a group of these groups (such as a group of Chapters), or the whole documents in the storage. This Section focuses on specifying the boundaries of a group of text blocks and determining the necessary information for making Annotation and Discussion on a group of text blocks and on a region or a term within a text block. Annotation or Discussion may be concerned with the content or the organization. In this Section, some issues have been addressed, such as: How can a Discussion and Annotation be made on the organization of a group of text blocks? How can a Discussion and Annotation be made on the text of a group of text blocks?

The boundaries of a group of text blocks are variable and undefined in the non-hierarchical structures or in both the hierarchy and non-hierarchical structures. As illustrated in Figure 41 part A, a user might consider Node1 as a starting node. Another user might consider Node3 as a starting node. A user might want to view the document in the storage (part A) hierarchically as in the outline (part B). Another user might want to view the document as in the outline (part C). The two groups of text blocks which are bounded by the nodes 4 and 8 are different from each other in the outlines (B) and (C). So, the same graph can be viewed in different ways and the boundaries can be different from one view to another.

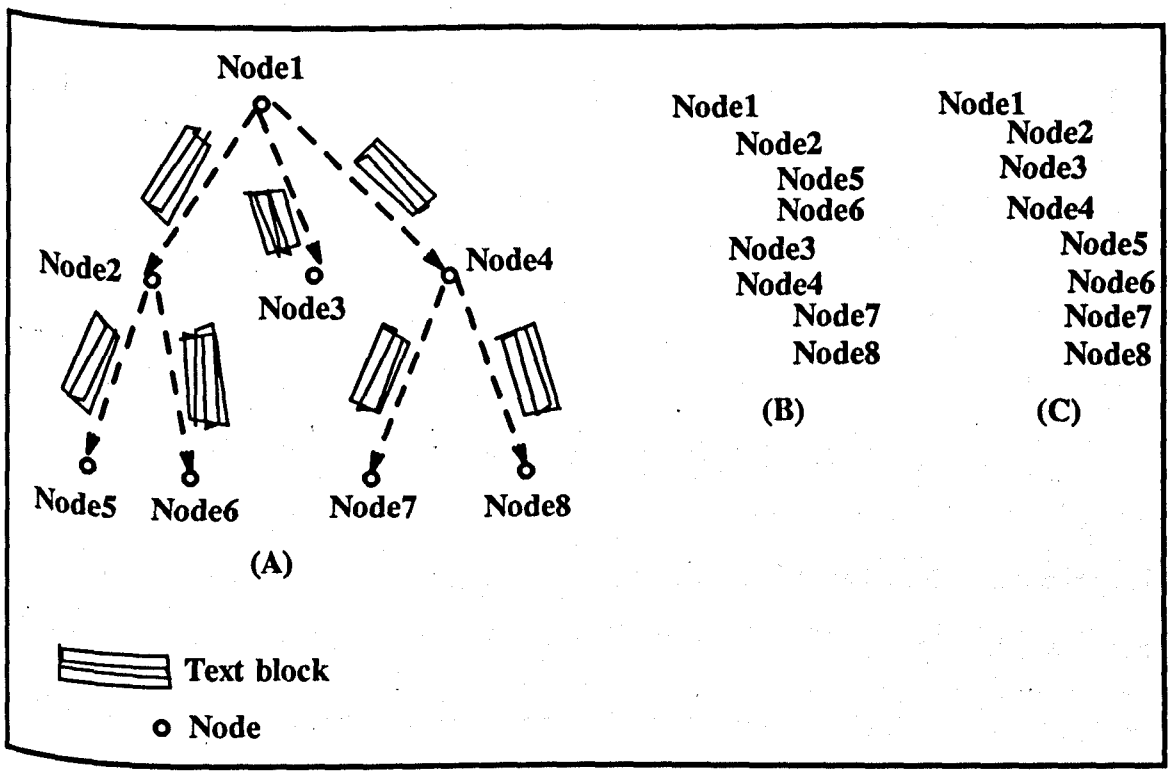


Figure 41 "Different views and undefined boundaries": illustration of different views for the same graph as in (B) and (C). The group of text blocks which is bounded by nodes Node4 and Node8 might include Node7 or the nodes 5,6, and 7. It is difficult to determine these boundaries from the graph. In contrast they can be determined easily from the outline. The nodes here are headings and the text is attached to the link, not to the nodes.

In order to make Annotation and Discussion on a group of text blocks, whether the annotation concerns the text itself (such as Section 'A' needs to be re-written) or the organization (such as Section 'A' should precede Section 'B'), the boundaries should be determined and known. So, the operations concerning a group of text blocks are dependent on both the presentation level and the storage level. The presentation level is needed to determine the boundaries of a group of text blocks, while the storage level is needed to store the data of the Annotation and Discussion. Creating Annotations and Discussions on a group of text blocks can be implemented by a method called *composition*, as illustrated in Figure 42. In this method, the boundaries of the group of text should be determined (i.e the two nodes which form

the two ends of a Section should be determined and called *composite nodes*). Then, the link object of Annotation will be linked to the composite nodes in the same procedure described in the "Storage Level" Section.

The *composition* method here is different from the one in the Dexter. Some of these differences including the fact that in this model, an outline should be generated first by using either depth-first or breadth first techniques. The boundaries can then be determined by selecting two entities from the outline. Also, most of the operations will be made on the entities of the outline not on the actual text. By contrast, in the Dexter model, all the nodes which need to be included in the composition should be determined. Generating an outline is not necessary and the operations are made on the nodes which are contain the text.

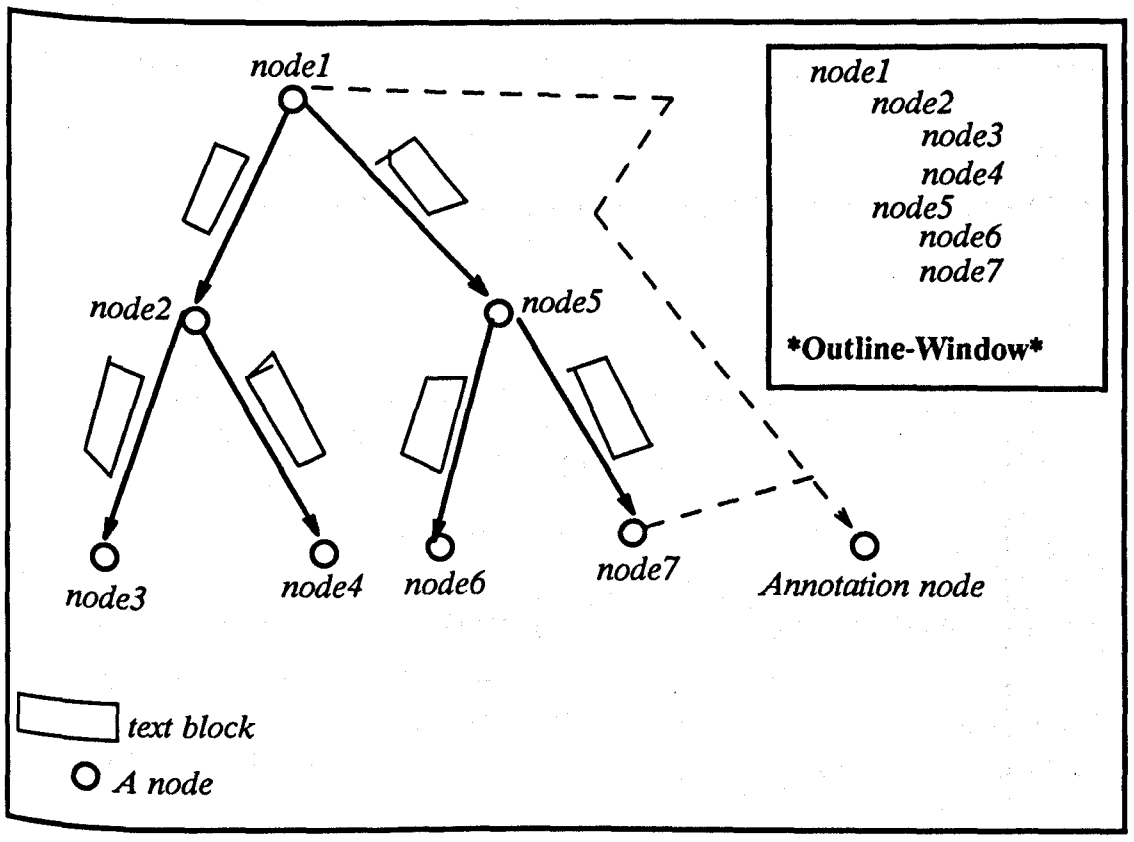


Figure 42 "Composition": illustrates how an annotation can be made on a group of text blocks by using the *composition* method. An outline should be generated first by using one of the traversal techniques such as depth-first. Assume that the nodes *node1* and *node7* are the *composite nodes*. From the graph, the annotation seems on the *nodes 1, 5,* and *7*, but from the outline, it is clear on the whole graph.

The question might be asked: How could the presentation level determine the boundaries of a group of text blocks? In order to answer such a question, there should be at least one technique to perform the traversal of the hypertext network and produce an outline (table of contents) such as parts 'B' and 'C' in Figure 41. However, there are many traversal techniques to produce a linear text from a hypertext network such as *depth-first* and *breadth-first*. After generating an outline for a hypertext network, determining the boundaries of a group of text block is straight forward.

To make Annotation or Discussion on a specific term or region in a text block,

the *anchoring* method should be used. In the *anchoring* method, the text block should be displayed first on the screen. Then the term or a region should be specified. The link to which the text block is attached and the two nodes on both sides of the link (i.e the *source* and the *target*) should also be determined as illustrated in Figure 43.

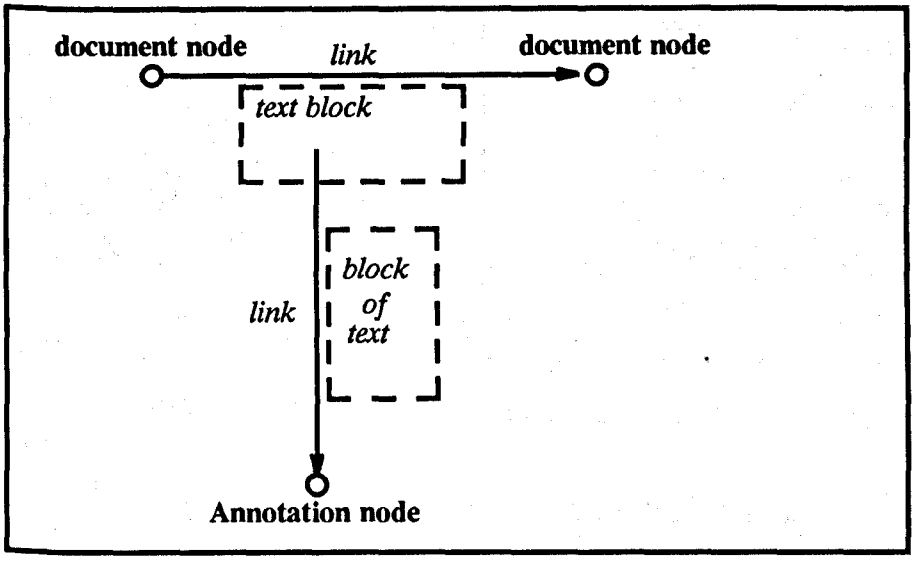


Figure 43 "Anchoring": illustration of how an annotation can be made on a specific term in a document text block by using the *anchoring* method. To determine the text block of the document, the two document nodes should be determined first. The target might be a term or a region.

Figure 44 illustrates the importance of the storage-presentation specification mechanism. In this Figure, there is a hypertext network (A) stored in the database. This network can be accessed by two different users, user 'A' and user 'B'. When user 'A' wants to view the network, the network should be brought up as the outline in (B). By contrast, when user 'B' wants to view the same network, the network should be brought up as the outline in (C). In order to separate these two cases and similar cases, the presentation level needs to access storage-presentation information encoded into the links in the network. This can be applied to the Annotations and Discussions made on a specific outline. For instance, when the outline (B) is

displayed on the screen, all the related Annotations and Discussions concerning this outline can be displayed if they are needed. The outline needs to be involved in this method, but this is not necessary in the Dexter Model.

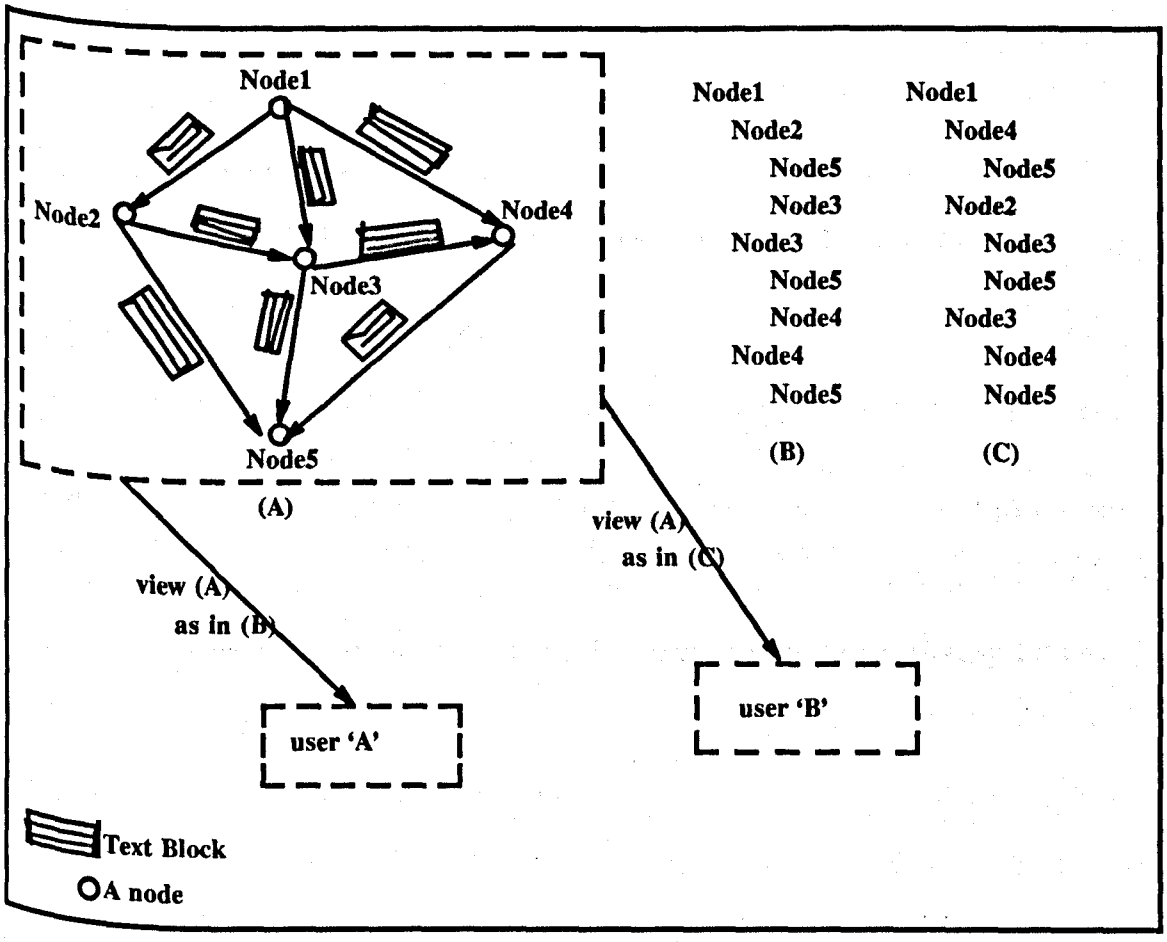


Figure 44 "storage-presentation importance": illustration of the need for presentation specifications on the access path.

6.3.4. The User-Model Interaction

In an automaton model of the system, the user creates a text block or goes from one block to another with a single command. The automaton model to be used here includes:

K, a finite set of states,

Σ, user actions,

$S \subseteq K$, the set of initial states, and

δ, a transition function from $K \times \Sigma$ to K.

The rules according to which the automaton M picks the next state are encoded into the transition function. Thus if M is in state q and the symbol entered by the user is σ, then $\delta(q, \sigma) \subset K$ is the uniquely determined state to which M passes. More specifically in the model M:

K = *{Document, Annotation, Issue, Position, Argument}*

S = *{Document, Issue}*

Σ = *{edit, view}*

and δ can be inferred from Figure 45.

The automaton model describes a language. Users can be asked to engage in Discussion and Annotation. In a system in which Discussion and Annotation are not connected, the user has to jump from the document to the Discussion system to discuss an *issue* and go back again to the document to proceed with the Annotation process. This could cause disorientation.

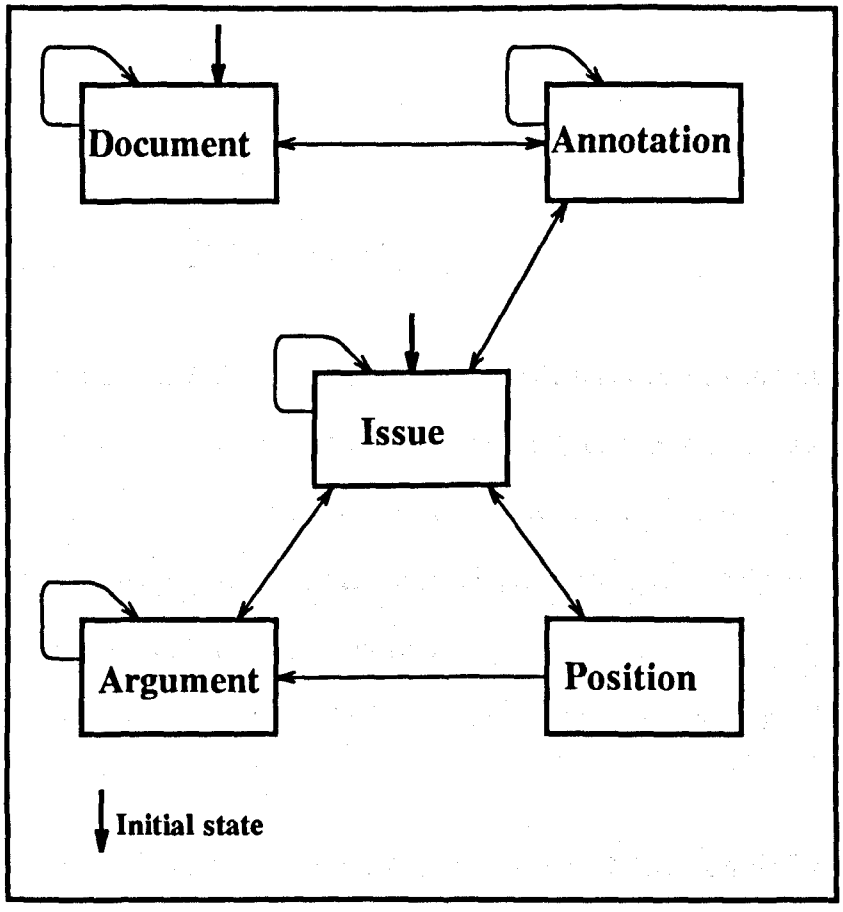


Figure 45 "Discussannotation Model": In going from one state to the other state the user may choose to either *view* or *edit*. Such commands are dependent on the context, such as *view* a Discussion or a document text block. The Annotation node types do not appear on the model because most of the relationships between the Annotation and the others, not among the Annotation nodes themselves. The *issue* could be of type Triple-issue or non-Triple-issue (medium or administrative matters). The Annotation also could be Triple Annotation or non-Triple Annotation. Such types need to be determined by users.

Users' behavior can be compared to that predicted by the model. Based on this user data, modifications could be made to the model. The model could also guide the monitoring of the users' actions. For instance, a computer program based on the model might prevent a user from creating an *argument* or a *position* while the user is in an *Annotation* state.

CHAPTER 7

Discussion, Conclusions, and Future Work

7.1. General Discussion

The aim of the research described in this thesis has been to investigate the role of hierarchy in Discussion and Annotation. The starting point for this work was a recognition of the significant differences between the different hypertext systems which support Discussion or Annotation. These differences, which relate to the nature of the data model and the sets of functionality, impose an important set of objectives upon the design of a single Discussannotation hypertext model which supports both Annotation and Discussion.

In exploring the fundamental differences between Discussion and Annotation:

1) case studies and experiments were performed;

2) different software algorithms were examined; and

3) different systems were explored.

Discussion and Annotation were tested in different levels. They were tested in the interface style, structure, and creating and viewing them. It was noted that:

1) Annotation cannot be independent. It is completely dependent on other activities, and thus the annotations should point to other activities even when they are intended to change something else, such as the system. by contrast, Discussion can be independent of or dependent on other activities such as Annotation; and

2) Annotation is a natural path from document to Discussion, and the Discussion node and link types could be considered as a complement to the Annotation

node and link types.

Additionally, the test went further down, to the word level. The algorithms indexing, STATBUILDER, and RELATION were applied to the document text, Discussion text, and Annotation text (Section 5.3 in Chapter 5). It was found that:

1) the relatedness between text of the Annotation's nodes is much less than it is between the text of the Discussion's nodes, and

2) the relationships between the Discussion's text blocks were much stronger than they were between the Annotation's text blocks.

The considerable number of similarities between Discussion and Annotation (see Section 5.4) and the small number of differences, lead to the suggestion of a Discussannotation hypertext model which supports both functions. The model developed in this research is different from the existing hypertext models in many aspects such as:

1) it supports making Discussion and Annotation on the different components of the document. It also supports making Discussion and Annotation regarding the medium and administrative matters;

2) it supports generating different outlines such as generating outline for the document, Annotation, or Discussion separately or in conjunction with each other;

3) users can view the document in different ways according to their perspectives and according to different attributes such as author and date;

4) it supports converting a network into an outline by using different techniques such as depth-first or breadth-first. However, the output of these techniques separately or in conjunction with each other could not generate an outline which would be in accord with the user's desire and perspective, individually or altogether. Such an outline can be generated by applying the Annotation

organization node types technique to the output of any current existing traversal techniques; and

5) it can manipulate the regular text and the headings separately. This aspect is for the Triple. This manipulation is one factor that makes the model more efficient than other models in some aspects. This manipulation also facilitates generating different outlines for the Triple. Such outlines facilitate making the *composition* mechanism. Such a mechanism resolves many problems in Annotation and Discussion.

The next Section discusses the various issues relating to the implementation of the systems and algorithms and discusses the general strategy adopted. The conclusions to be drawn from this work are given in Section 7.3. Finally, Section 7.4 makes suggestions for a discussion of possible future work.

7.2. Discussion of Results

This Section discusses the strengths and the weaknesses of the use of the systems and discusses the results of the experiments and case studies. Section 7.2.1 discusses the results of the experiments and case studies that were performed in Chapter 3. This Section also discusses the strengths, weaknesses, and uses of HERD. Section 7.2.2 discusses the results of the experiments and case studies that were performed to test the various issues in the Annotation hypertext systems, while Section 7.2.3 discusses the results of the experiments that were performed to test some issues in browsing systems. The power and efficiency of the Discussannotation model will be discussed in Section 7.2.4.

7.2.1. Discussion Hypertext Systems

There are many variations in the current Discussion systems (see Table 1). In order to test such variations and to explore some issues regarding Discussion, the HERD system was created and three case studies and one experiment were performed. The HERD system enables its users to target several abilities. Some of these abilities are: (1) the ability to use the computer from the project's beginning, and (2) the ability to critique and complete some work requirements. Having a Discussion on the computer using the HERD system allows the user to read Discussion text in their own time and to prepare thoughtful responses. This type of discipline appears to encourage a more meaningful dialogue. Unlike the situation of face-to-face Discussions, it was clear which issue a participant was addressing and there was a written report at the end of the Discussion. On the other hand, after seven meetings of face-to-face Discussion (more than 15 hours) there was no report, nor an agreement about a guideline for software requirements specification.

In Section 3.3.1, the semantics of hierarchy and non-hierarchy were tested. The results suggest that the semantics hierarchical structure is preferable to the non-hierarchical structure but the latter is important and sometimes needed. Selecting the hierarchical structure to represent the Discussion facilitates the reading process and diminishes the disorientation problem. Adding the decomposition to the hierarchical structure not only facilitates the reading process, but also eliminates the disorientation problem, diminishes the need for the non-hierarchical links, and leads to a conclusion faster than other approaches such as using non-decomposition. Since there was just one group of subjects participating in this exercise, it could not be conclusively ascertained that the decomposition is preferable to non-decomposition. In order to test this, another case study (Section 3.3.2) was performed. The results

of this case study suggested that the decomposition is preferable to the non-decomposition (61 percent decomposition versus 39 percent dialog and non-decomposition).

Section 3.3.1 reveals how very difficult it is to produce the requirements specification documents report at the end of the Discussion without using specific node types for resolution. It also reveals how difficult it is to know whether or not the Discussion was completed and how difficult it is to display the resolution without supporting decision-making. Furthermore, using just three types of links in the exercise causes many problems. For example, one user's response to an issue may not be clear to other users, who may be unclear whether the user who made the response was offering some criticism or was responding in support of an issue. In addition to these problems, some of the operations were difficult to implement in the absence of some necessary link types. In spite of the usefulness of using a small set of node and link types, it is not a good typing strategy. Other strategies might be useful such as the system supports a big set of node and link types, or the system does not support typing and this is to be left as an open option to the user.

In order to test whether the syntax of hierarchy is preferable to non-hierarchy and whether or not the hierarchical structure will satisfy all user needs, an experiment (Section 3.4) was performed. This experiment suggests that the syntax of the hierarchical structure is more preferable than the non-hierarchical structure but as seen in Section 3.3.1, the syntax of non-hierarchical structure is important. As a result of both the exercise and this experiment, the Discussion systems should support both hierarchical and nonhierarchical structures for both the representation on the display and in the storage. Therefore, the Discussion systems should have the ability to convert the hypertext network (stored in the database) into a hierarchy for presentation on the screen, and thus techniques such as the depth-first and breadth-

first techniques should be supported by Discussion systems. One more thing revealed by both Sections 3.3.1 and 3.4 was that suggesting a set of node and link types to the users and giving them an option to add more link and node types is a good strategy. It cannot be certain, however, whether it is the best strategy. Some other options had not been tested at this time, such as the open user option (i.e typing to be left open for the user).

In terms of exploring issues about Discussion (Section 3.3.1) based on the analysis made on the titles and the text of the nodes, the data from a Discussion was found to be similar to the regular text, in that the data from the Discussion has headings and content. Therefore, the data from the Discussion should be manipulated like normal text in this aspect. So in the data model, the Discussion heading should be distinguished from the content. Also, viewing the heading of the text is more important than viewing the text itself (the heading file was viewed 84 times, while the text file was viewed 74 times).

Furthermore, a test was carried out, of users' responses to the created Discussion nodes and to how they viewed them, in terms of seniority. Thus a node created by a Professor has a higher priority than one created by an undergraduate. This test was done by registering the date and the time for each node. Also, by registering when and who created which node. It was found that the nodes which are created by the most important subject were responded to before the others. Additionally, viewing when and who created a node is more important than viewing the heading and the text (116, 84, and 74 times) respectively.

In exploring the fundamental differences between Discussion and Annotation, a case study (Section 3.3.3) was performed. The initial hypothesis was that the processes of Annotation and Discussion are sufficiently similar, for a Discussion system such as HERD to serve equally well in the support of Annotation node and link

types. In the case study (Section 3.3.1), HERD was used in the discussing of requirements document outlines. In the case study (Section 3.3.3), the same Discussion system was used to support the Annotation of approximately 600 paragraphs of an online book. The hypothesis that the Discussion system would serve equally well as an Annotation system was refuted.

The requirement that each node and link be classified into a small set of candidates did not seem difficult to follow for those in the 'Discussion' case study Section 3.3.1. Labeling nodes as *issues, positions*, and *arguments*, and labeling links as *generates, responds-to*, or *other* were natural for those in the Discussion. On the other hand, when using HERD for Annotation, such labeling of nodes and links was not easy to apply. As an annotation was a comment on a Chapter it was not always natural to classify it as an *issue, position*, or *argument* nor to classify the link types as *generates, responds-to*, or *other.*

Data from the two case studies Sections 3.3.1 and 3.3.1, were comparatively evaluated as follows: There were similar numbers of nodes in total (59 vs. 55). Of these, the same number (11) were in both cases used administratively, i.e. to organize the Discussion or the Annotation. Of the rest, 10 percent (6 nodes) were part of a secondary discourse in the Discussion case study. In the Annotation, four nodes were not classified as either *issue, position* or *argument.*

The features of the HERD system requiring users to type classify their node into an *issue, position*, or *argument* suited the two uses differently. A common finding seemed to be that users had greater problems distinguishing between the use of *positions* and *arguments* than between *issues* and either of those two. For Discussion, users expressed little dissatisfaction with this feature of the HERD system, whereas there were signs that the Annotation group had more problems. When the structures of the two node trees resulting from the case studies were examined, it

was found that the Discussion would progress from one or a few initial *issues*, with several *positions* and *arguments* to each *issue*.

A Discussion is similar to annotations on annotations, except that there does not have to be a document which is at the root of the annotations. Furthermore, one may want to define Annotations such that they only point directly to a document. In this case, a comment on an annotation immediately becomes an instance of a Discussion. In the Annotation case study, four Discussions were made and each one of them started with an issue which was actually a comment and thus Annotation could lead to a Discussion.

In the Discussion the one root issue led to several sub-issues which in turn led to other sub-issues. In the Annotation the top level issues were essentially prompting for comments on a Chapter, and there was one such issue for each Chapter. Comments on a Chapter tended to point directly to the top-level issue of 'What do you think of the Chapter?'. This structuring of the annotations facilitates the generation of high-level comments on the Chapters of the book but does not exploit the organization of the book into Sections within a Chapter. To comment on a Section an annotator would need to somehow specify the Section as issue.

Many observations had been noticed in the case studies, experiment, and on the use of HERD. Some of the observations are:

The usefulness of hierarchical structures and decomposition.

The same HERD users who supported and recommended this system didn't use it to discuss the requirements specification documents for another system called MUCH. It seems there is a contradiction, but actually there is not. The subjects attended more than seven face-to-face meetings to discuss a guideline for software requirements specification before doing the case study Section 3.3.1. In this case

study, examining the first 32 nodes, the non-decomposition method was used and more than 70% of the links were hierarchical. Confusion, restructuring of the information, and the adding of unnecessary nodes were also happened in this part of case study. By contrast, in the examination of the next 27 nodes, there was just 4 percent of non-hierarchical links, no restructuring information, and all the nodes were related. Thus attending 7 more face-to-face meetings and using non-decomposition strategy in the first half of the case study helped the subjects in understanding the problem and how to decompose it. It might be conclude that a pre condition of using hierarchy and decomposition is understanding the problem and how it can be decomposed.

The usefulness of using 'proto-node'.

A Discussion using HERD forces the users to structure their ideas and organize their thoughts in order to incorporate the new information within the existing structure. However the early phases of the Discussion are fragile and critical, and an issue, is sometimes vague and the response is unclear. To override such problems, a proto-node might be used (eighty percent of the subjects used this option, Section 3.3.1) as a private space for recording notes, segmenting the mucks, identifying their types, and linking them to the current structure.

Sometimes, while users are reading literature to prepare a response for an issue, some useful information might be found for another issue intended to be created. Sometimes also, while a user is typing a response, some thoughts and ideas might come to his/her mind. In the non-hierarchical structures, shifting to another place and linking such thoughts and ideas to the most appropriate place might be easy. On the other hand, the hierarchical structure can be seen as one body, and such thoughts and ideas should be placed in the right place. If in the hierarchical structure the right place is not yet ready, then the proto-node is the right place for

such thoughts and ideas to be recorded.

Capturing the Resolution.

When people participate in a long Discussion, they discuss many things and create many nodes. Some of these nodes are organizing nodes, others might not be relevant to the Discussion itself, and the others might be comments. As a result, the actual nodes which discuss the problem are hard to see, and the users may experience confusion as to their current whereabouts in the Discussion. One of the major problems confronting HERD users is that of capturing the resolution. Also, there is no efficient method in the IBIS systems to represent the resolution for a specific issue, nor for the whole problem. When HERD users become confused about the latest result of a Discussion of a specific issue, they restructure the information from different nodes and add a new node. They then carry on the Discussion starting from that node.

The literature did not show how resolution can be extracted from the different nodes of a Discussion. The following is a possible solution for capturing the resolution problem. Assuming that a problem can be divided to a number of issues, and that each issue can be divided to a number of sub-issues. The resolution of the whole problem consists of the resolution of all the sub-issues. The resolution of each issue is the resolution of that issue, besides being the resolution of the sub-issues linked to that issue. To capture the resolution, at least three types of link are needed: responses, objects, and supports. The following algorithm can be applied to a problem, issue, or sub-issue in order to capture the resolution.

1) The contents of all Positions attached to an Issue should be included in the resolution.

2) The content of all Arguments attached to a Position with the link "supports"

should be added to the contents of that Position. Also, all arguments linked transitively via a sequence of "supports" links should be added to the content of that Position.

3) All or part of a Position or Argument should be removed from the resolution if it has been rejected by an Argument with the link "objects".

4) For a series of objections the following rules can be applied:

   a) Odd (objection) is an objection.

   b) Even (objection) is a support (i.e The Argument or Position will be accepted). For example (see Figure 46) let A = Argument, P = Position, and I = Issue. Then, If both P1 and P2 respond to I1, A1 and A2 supports P1, A3 objects to P2, A4 objects to A1, and A5 objects to A4, in that case P2 will not be included in the resolution for I1. The resolution will be the contents of P1, A1, and A2.

The usefulness of decision-making.

This observation shows how a decision might be made on resolving an issue. One of the problems encountered in HERD is that users may have finished discussing a specific issue, but most of the users will be unsure as to whether the Discussion is finished or not. Conversely, some users might think that a specific issue was resolved, but actually it is not. Using decision-making might solve such a problem.

Thus, a decision should be assigned for each *issue*; in particular, those issues which will be included in the final resolution of the problem.

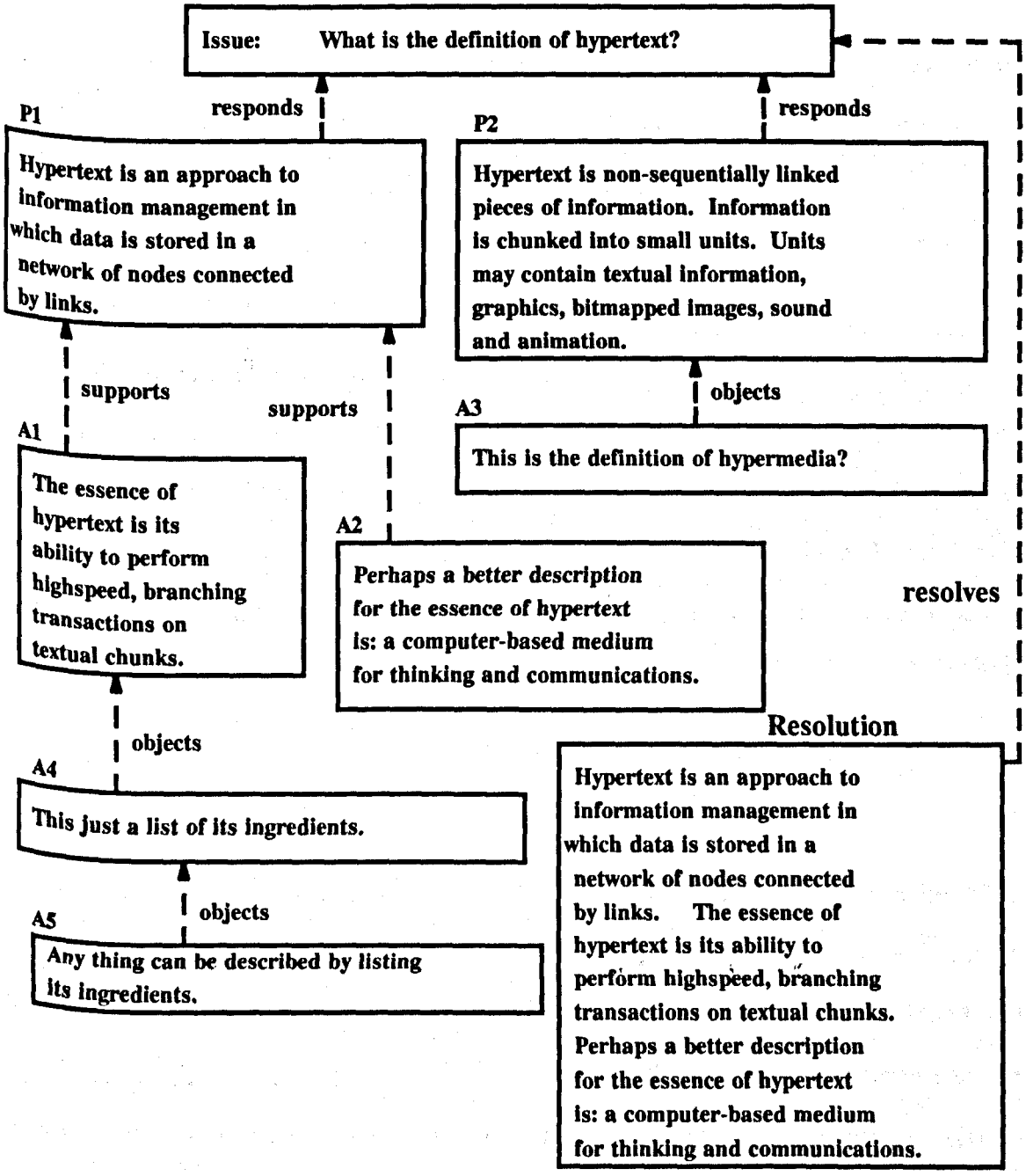


Figure 46 "Capturing resolution": An example explains capturing resolution.

One of the possible solutions to this problem is the *Consultation* decision method. There are two cases in this method:

1) single decision making and

2) multiple decision making.

In the first case, the decision-maker has the authority to assign any type of decision--positive or negative, to any type of node– *Issue, Position, Argument*, or *Surrogate*, at any time. This should be done after the necessary consultation has been made with the participants. Unfortunately, the system cannot force the decision-maker to consult the other participants, or to take their opinions into account.

In the second case, the decision will be assigned to a specific node by the Discussion leader, if a consensus has been made to that node. Otherwise, one of the voting schemes should be executed, and the decision type will be assigned by the system to the selected node. Of course, the discoursers should be given the option of changing the decision attached to a node, and the system should be able to make such a change.

7.2.2. Annotation Hypertext Systems

One of the major problems, may be, is the distinction between the headings and the text. In Section 3.3.1, it was easier to respond to the text than the heading, because the heading is embedded in the text. Another problem exists in HERD and that is the distinction between the structure on the display and the structure in the database. In order to arrive at a solution for such problems, a hypertext system called MUCH has been created (Section 4.2).

The ability to make annotation on the headings and the text of a document in MUCH was tested. Results suggested that it was easy to annotate the headings of a document. On the other hand, it was difficult to do the same to the text of a document. However, assuming that the classification of the hypertext systems divides into two groups (creating document systems and browsing systems), then annotations

to the heading and to the contents of a document in the first group of hypertext (document creation) might be needed. However, Annotation to only the headings in the second group of hypertexts (browsing systems) might be needed. So, attaching the text to the node does not satisfy the document creation hypertext systems, nor the browsing systems. On the other hand, attaching the text to the link does satisfy the browsing hypertext systems (assuming the content is perfect).

A set of node types has been suggested for Annotation (Section 4.4). In order to test these node types and to explore some issues about Annotation, an experiment (Section 4.5.1) was performed. This experiment was concerned with the organization of the document. The hypothesis that restructuring the entities in the outline by using the Organization Annotation node types would give the author an indication about the quality of that document was fully supported. It confirmed the usefulness of using the organization of Annotation types. However, the method suggests that moving an entity from one place to another might work automatically only at the same level of outline, but moving an entity from one level to another needs to be done manually.

In hypertext systems, there are many issues with regard to viewing published documents (such as papers or books) stored in the database (see Section 6.3.2). These issues concerning the viewing of a hypertext might be satisfied by registering, in the re-organization process; the history of what a user selects to view next, which node was selected to be the first in the outline, and the necessary information about each reader which reflects his importance. This information can be registered for each version of a hypertext. Thus, the Annotation organization node types might facilitate viewing one hypertext from different perspectives. Decisions about the removal and retention of such annotations will be dependent on their purpose. For example, if published documents are stored in database on a form of hypertext and

readers want to view them from different perspectives, then such annotations might continue as long as the documents exist.

A hypertext can be viewed in different ways (i.e different outlines could represent one hypertext). If someone wants to view a hypertext, then this raises the following question: What are the precedence factors for presenting an outline which represents a version of hypertext to a reader? In answering such a question, the readers who would like to view a hypertext, could be classified into two groups, depending on whether they shared in the re-organization process or not. The first option to a user who shared in the re-organization process might be the outline which that user selected. The selected outline can be discovered through the registered history for each user in the re-organization process. On the other hand, the first option for a reader who did not share in the re-organization process might be the outline which was agreed on by the users at the end of the re-organization process. One or more additional options could be offered to both types of readers (those who shared or didn't share in the re-organization process). These options are dependent on different attributes. The importance of these attributes might be different from one reader to another or from one group to another. Some of these attributes are:

1) Date: the hypertext might be modified from one date to another. A user might need to view a specific version of the hypertext on a specific date. Even with the same date, the hypertext might be modified by different users at the same time. In such a case, a user would need to specify both the date and the author.

2) Author importance: the hypertext might be modified by different authors. So, a user might need to view the version which was modified by the most important author or the less important one. The author importance can be determined by

different characteristics. Some of these characteristics are: the status of the author (such as Professor, Doctor), and the number of publications (papers, books) that he/she has produced.

3) Demand: one version of a hypertext might be viewed by different users more than another version. Also, one specific outline which represents a version of hypertext might be viewed more than another outline. Furthermore, an outline starting with a specific node might be selected more than another. In such a case, a reader might determine which version or outline (starting with or without a specific node) he/she wants to view, based on the number of times that version or outline has been selected.

Thus, the hierarchy could help the readers in making a decision regarding the version of the document, its outline, and from which perspective the readers want to view it. Also, the hierarchy could help the author of the document in deciding how far he/she is from reaching the final version, by looking at the organization of the most important parts of the document.

The results of the experiment (Section 3.3.1) might open new vistas to the applications of Annotation in document creation and browsing. One can hypothesize that the Annotation "content" and "organization" node types could give authors an indication regarding the quality of a document. When most of the text blocks are attached by both "good-content" and "good-organization", this will give the authors an indication how far they are from reaching the final version. This will also direct the authors to the parts of the documents which need more work than the others. Furthermore, different documents might be generated based on these two groups of Annotation node types. This is can be done by following the "good-contents" and "good-organization" Annotation node types.

The experiment (Section 4.5.2) was concerned with the content of the document and a task of making annotations. One of the hypotheses was that using the computer system for making annotations on a document will diminish the need for a face-to-face Discussion. This hypothesis was refuted. In this experiment, Annotations were made to this dissertation by using three different approachs: paper, flat ascii file, and MUCH, respectively. Each has its own problems, most of which differ from the problems of the other two approachs. When using the paper approach:

1) different strategies for making annotations are employed;

2) more than one page can be seen at the same time, tables figures are clear and they are in the final form, so the reading process is easy;

3) annotators may make many short comments and connect them to the document with pointers that can isolate arbitrary portions of a page; and

4) the face-to-face Discussion is concerned with the understanding of the annotations rather focusing on the document.

The productivity (the feed back written for the document), in the ascii file approach, is greater than in the paper approach. The Annotation process, however, is more difficult. When using the ascii file:

1) the annotator can write more than in the paper approach, so understanding the annotations will be easier;

2) the face-to-face Discussion is concerned with finding the annotations and their response; and

3) there is a difficulty of pursuance of the Annotation history.

In the MUCH system approach, some of the problems which had been found in the previous two approaches were resolved, but at the same time new problems were discovered. Using this approach, makes it is easier to find who created which

annotation, and also, to see when an annotation was created and which annotation has been responded to. All the annotations can be seen through the outline which can be generated for the Triple activities (document, Annotation, and Discussion). The pursuance of the annotations can be traced more easily with MUCH than with the other two approaches. The feedback for the document when using MUCH was better than others.

On the other hand, some problems were discovered. When using the MUCH system, many of the comments are of an administrative or system-specific type and do not relate to the content of the document. The user is constrained in the way that parts of the document can be connected to the Annotation. Additionally, the MUCH system facilitates comment that is intimately linked to the semantic net or outline of the document. The advantage is that the major conceptual issues of the document are the focus of attention. The disadvantage is that comments which someone might want to make are either:

1) not about the document content or

2) on a level of organization of the document distinct from its outline or semantic net

and thus they are difficult to make.

This experiment shows how Annotation leed to Discussion; in 8 out of 24 annotations, each one led to a different Discussion. Some of these Discussions were registered in the computer and some others were made face-to-face. The face-to-face Discussion was seen as an easier way of facilitating Discussion than using the system, for two different reasons:

1) The goal of the Discussion was not to produce a written text, but to understand an issue or to exchange viewpoints, and

2) the system is slow and it uses "typing" rather than pointing. The Discussions that used the computer occurred in this form because the subjects were not working in the same place at the same time. Finally, the Annotation node types which were suggested with regard the content of the document were they never used. They were, however, used with different words which have the same meaning. It appears that this task was not suitable for use with suggested words that were the same.

An important observation of these experiments concerns the large extent to which people may get side-tracked by a new technology system and want to enter information into the system which is not about the task at hand, but is about the system itself. Nevertheless in the case of the Discussion with the HERD system, the text was usually specific to the central goal of that Discussion. Thus in some circumstances the computer support may have a focusing effect.

In exploring the fundamental differences between Annotation and Discussion, a case study (Section 4.3) was performed. One can conclude from this case study that most of the link types for Discussion are useful for Annotation, while the Discussion node types might be considered as a complement to the Annotation node types. Furthermore, assigning a name for a procedure which does not exist or is unknown to the users, is not useful. The opposite is also not useful (i.e using an action without assigning a name for it). For example, using the name 'Issue' without knowing what the issue means, is not useful. Also, using an action such as a text block written in a form of question or concern which needs a resolution, without assigning a name for such a procedure is not useful. The useful and the natural is to find the action first, then a name should be assigned to it.

In this case study, it was difficult to specify the text block on which the annotation was made, without mentioning the link object in the text by the subjects. This

was due to a fault, namely that a node can be linked to more than one node. A node might appear more than once in the outline, but when a subject hits 'Return' on the same node name in different places in the outline, different texts will be retrieved. This was due to a fault, namely that the text is attached to the link not to the node. For example, in Figure 47, hitting "Return" on the node name "motivation" in the "*Frames Window*" (or in the "*Text Outline*" window), causes the text attached to the link between "motivation" and "preface" to appear in the "*Paragraph Text*" window. In Figure 48, hitting "Return" on the same node name "motivation" causes a different text to appear in the "*Paragraph Text*" window. This is because the previous text is attached to a different link (the link between "motivation" and "writing_tool"). Thus if a subject wants to make an annotation (or a Discussion) on the text which is attached to a link, then the subject should mention the link object in the text of annotation. Otherwise, it is difficult to identify the text on which the annotation is made.

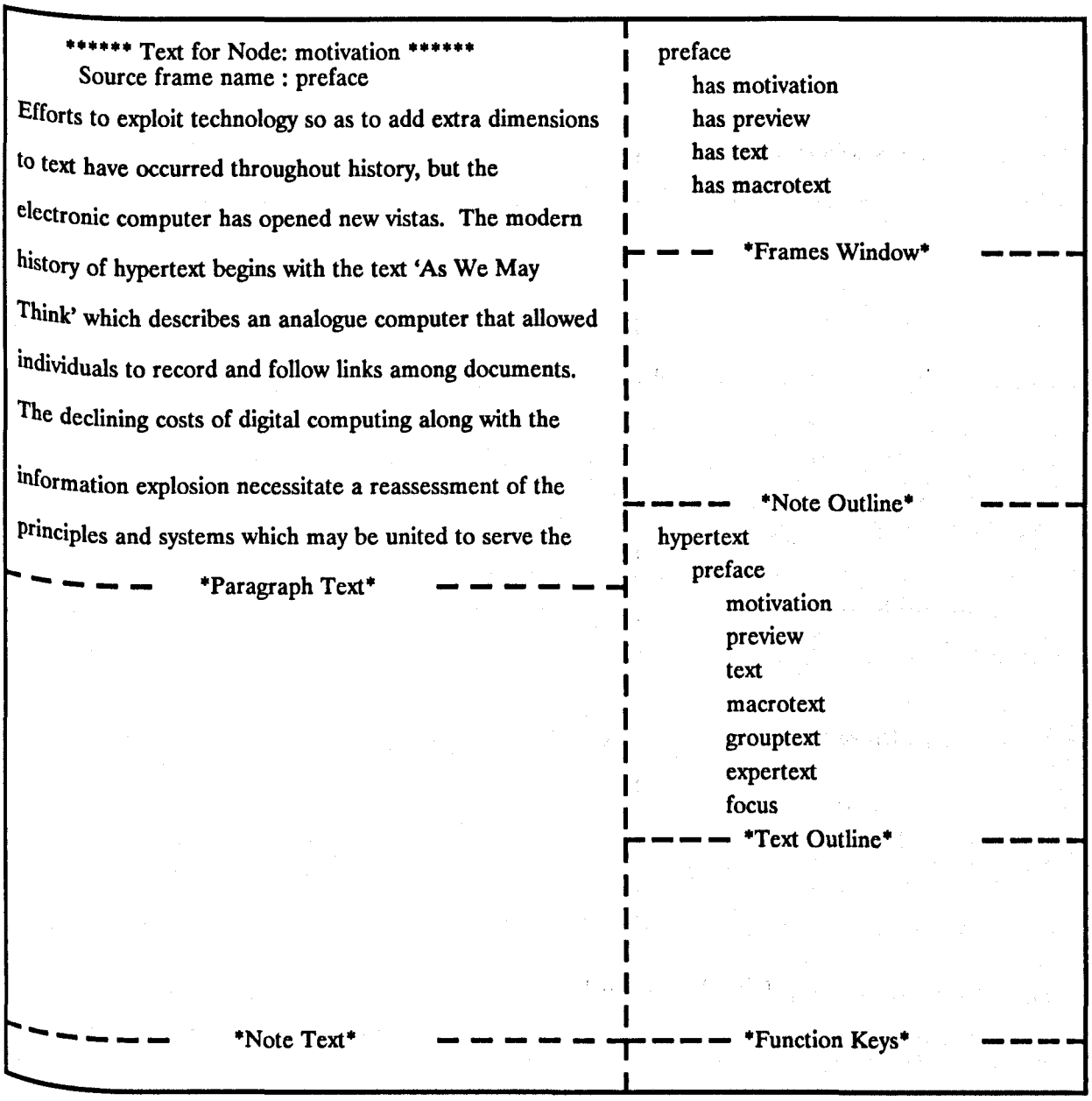


Figure 47 "Text and link object (A)": This figure and the next one show how 'clicking' on the same entity in different places in the outline, causes different text to appear on the emacs window. In this figure clicking on 'motivation' entity causes the text attached to the link object 'preface has motivation' to appear.

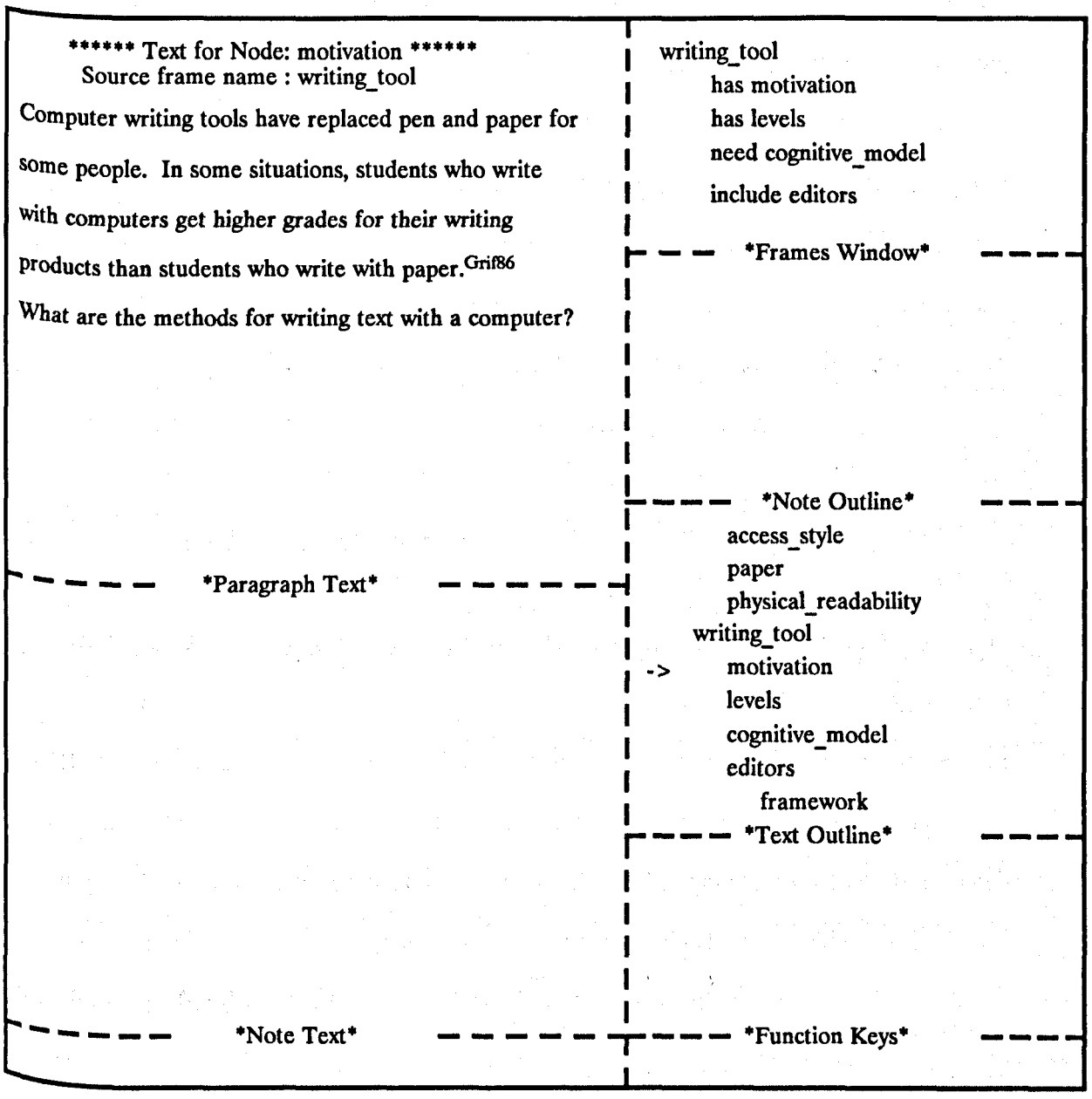


Figure 48 "Text and link object (B)": In this figure clicking on 'motivation' entity causes the text attached to the link object 'writing_tool has motivation' to appear on the emacs window.

The role of hierarchy in Annotation was both direct and indirect. The direct contribution of hierarchy in Annotation was very limited. The indirect contribution of hierarchy in the Annotation through the hierarchy of the document was considerable. The hierarchy supports decision making in both creating and reading annota-

tions. The decision can be made as to which annotation needs to be read first and responded to; which part of the document first needs the creation of an annotation, or from which perspective a hypertext of the document needs to be viewed, based on the different attributes such as reader importance, the document components, or the importance of the information. Finally, by making annotations on the hierarchy of outline, readers could view the same hypertext of the document from different perspectives. This aspect is expected to solve most of the problems which are related to converting the non-hierarchical structures to a hierarchy of outline.

7.2.3. Browsing Hypertext Systems

When users move around a large information space as much as they do in hypertext, there is a real risk that they may become disoriented or having trouble finding the information they need. Giving the users some guidance about one or more related nodes which they have the most relevant information might diminish such a problem. An attempt was made to explore the extent to which the word-frequency indexing may be augmented with Mili's method to provide both a relational indexing for documents and a semantic net for browsing.

Word-frequency indexing was used to analyze the contents of 600 paragraphs, which were part of a textbook entitled *Hypertext*. Mili's algorithm– called STATBUILDER, was applied to the resulting indexing vocabulary to build a hierarchy. The accuracy of the index terms and the derived semantic net were tested first (Section 5.3.1). Mili predicted that the relationships generated by his algorithm are "Broader-Term"-like. Given two text blocks TB_1 indexed by t_1 and t_2 and TB_2 indexed by t_3 so that STATBUILDER identified a relationship between these index terms, it was of interest to ascertain whether TB_1 did, indeed, describe a relationship

between t_1 and t_2, and whether there was a relationship between TB_1 and TB_2. The results showed that 75 percent of the text blocks whose set of index terms included the end nodes of a link generated by STATBUILDER described a relation between those nodes. They also showed that 65 percent of different Sections were completely recognized and the rest partly recognized. Thus, it is reasonably sure that whenever a pair of related index terms appear in the index set of a text block, the text block describes the relationship between the pair. Also, whenever two different index terms appear in two different text blocks so that a relationship is identified between the index terms, then a relationship is also identified between the text blocks.

The ability to rank paragraphs according to their similarity to a specific paragraph using the word-frequency based indexing was tested (Section 5.3.2.1). The results illustrated that it was feasible to use this method of indexing to measure the similarity between paragraphs. Word-frequency based indexing and the derived semantic net were used to attempt to find a paragraph related to another paragraph (Section 5.3.2.2). This experiment indicated that the relationships in the derived semantic net were sufficiently accurate to support browsing and thus enable the user to consult a related text block.

In exploring the similarities between Discussion and Annotation, two experiments were performed (Section 5.3.3) using three algorithms- indexing, STATBUILDER, and RELATION. In the first experiment, the similarity of the text of Discussion and Annotation was tested. It appears that since the text blocks of Annotation were not related (53 percent), but the text blocks of Discussion were related (89 percent), Annotation and Discussion were not similar. This confirms that within the limits of this study, the tree structure for annotation is more conceptually disjointed than the same structure for a Discussion.

In the second experiment, the similarity between the relationships of the Discussion and Annotation was tested. The results showed that the relationships between the Discussion text blocks are much stronger than the relationships between the Annotation text blocks. This supports the results of the previous experiment in that most annotations are completely dependent on the document, while the Discussion nodes can be independent from the document, and they address one topic.

7.2.4. Discussannotation Hypertext Model

The Discussannotation hypertext model, as described in Section 6.3, is as efficient in many aspects as other models, more effective in some aspects, and provides more functions with respect to document creation than any existing hypertext model. This model is the only one supporting creating Discussion and Annotation on the various components of the document. This can be made by the virtue of the *anchoring* and the *composition* methods. These methods are difficult to implement without having a hierarchy of outline.

In order to make Annotations or Discussions on a group of text blocks, determining the boundaries for a group of text blocks is essential. Since the hierarchy is dynamic and not a data storage, then determining the boundaries for a group of text blocks by using the graph only is a difficult or sometimes impossible task. Such a task becomes easy by the virtue of converting a graph into a hierarchy of outline.

This model manipulates the text and the headings separately. The users can create headings or text blocks separately. This manipulation is one of the features that make the model more efficient than any other existing model for some tasks such as generating different outlines. An example of this method could be the finding that the document, Discussion, and Annotation consists of headings and text.

Since they are similar in this regard, it is possible to use one method to create them all, minimizing the number of modes. This finding is important from both users' and designers' perspectives. The number of skills which the user should learn will be decreased. Similarly, the number of application programs which need to be made by the designers will be reduced.

The usefulness of using typing was also proved in the literature and in this work. This model supports presenting a set of Annotation and Discussion nodes and link types to the users. The strategy is to present the existing node and link types to the users with the option to modify these types and to add new types as they arose. The way in which the node and link types are presented to the users is important. The presentation of such types to the users should be dynamic, and guide the user in accomplishing the task.

Finally, the efficiency of this model ranged from negative to positive. This model is less efficient than other models when creating one document only. By contrast, this model is more efficient when more than one document needs to be created and Annotations and Discussions need to be made. The increase of this efficiency will depend on the manner in which the users cooperate. All these issues will be discussed in some detail in the next Sections.

7.2.4.1. Hierarchy and Efficiency of the Model

The assumption is made that the Discussannotation hypertext model (Section 6.3), if well designed, is more efficient than any existing hypertext system in some specific tasks and for some specific users. Assuming that there are some hypertext system models which provide their users with the same functions as the Discussannotation hypertext model does, then an attempt is made to prove that this model is

as efficient as the other hypertext system models in many aspects and more efficient in some other aspects. A hypertext system 'A' is more efficient than a hypertext system 'B', if the users, by using system 'A' can perform a task 'T' in less time than using system 'B'.

The hierarchy of outline for the Triple needs to be viewed as a whole, or part of it, according to the user's desires and perspectives. The cost of generating an outline in this model will be comparing with the cost for other models. How the cost will be affected by the number of users and their way of working (collaboratively or not) will also be demonstrated. Which model is more efficient than the other, and when this is the case, will be predicted.

What is the cost for generating an outline? There are two cases:

1) the headings are separated from the text as in this model and

2) when the headings are embedded in the text as in the other existing models.

It is assumed that a draft of the document and some Annotations and Discussion nodes have been created. The task is to view the Triple in the storage and to create more Annotation, Discussion, and to bring the document forward toward the final version. In order to perform the task, the outline needs to be generated.

Assuming there are N nodes and each node has J paragraphs which were created by K users. The cost for generating the outline once in the first case (headings separated) is N time units where N is the number of headings. In the second case (headings embedded in text), the cost is N * L, where L is the number of time units which are needed to retrieve the text block which is attached to the heading. This cost will be further increased depending on 1) how many times each user wants to generate the outline; 2) the number of users; and 3) how the users want to work.

In many cases, a user might want to generate the outline for the Triple. Some

of these cases are:

1) at the beginning of the work;

2) switching from one part of the outline to a different part (such as switching from the Annotation outline to the Discussion outline);

3) switching from one desire to another (such as switching from outline by author to an outline by date);

4) switching from one starting node to another;

5) adding one or more text blocks or headings;

6) deleting one or more text blocks or headings; and

7) modifying the organization of the document.

Thus, it is assumed that each user wants to generate the outline G times. In such a case, the cost for the first case will be increased from N to K*G*N. By contrast, in the second case, the cost will be increased from N*L to K*G*N*L time units.

How the users cooperate with each other will affect on the cost of generating the outline. For example, if the users are working at different times, in different places, and without any communication, then the cost in the first case will be K*G*N, while in the second case, the cost will be K*G*N*L time units. If the users work cooperatively and divide the work equally between them, the cost will be decreased in the first case to K*(G/K)*N (i.e G*N) time units. In the second case, the cost will also be decreased to K*(G/K)*N*L (i.e G*N*L) time units. This assumes that they are working cooperatively in one room, that they divide the work equally between them, and that they have communication between them so that one of them will generate the outline once, after all the users finish their parts. In such a case, the cost in the first case will be N time units. In the second case, the cost will be N*L time units. Thus, this model is always more efficient than the other

models in viewing the hierarchy outline of the document. The cost of generating an outline will also be decreased when the users are working cooperatively and they are able to communicate with each other.

7.2.4.2. The Power of the Model

One system will be considered more powerful than another if that system provides its users with more functions than the others. Power can be defined as a number of functions that a system provides for its users for them to be able to perform some tasks, assuming that supporting such functions is necessary. Some functions facilitate the task and others are necessary for the task to be performed. One example of the first group is the use of e-mail messages. Using e-mail messages facilitates the Annotation and Discussion activities in different ways. For example, in a case where there is no support for e-mail messages, if a user wants to know whether his issue has been responded to, or a specific Section has been annotated, then the user should set up the system and find out whether his demand has been satisfied or not. In the case, for instance, where setting up the system needs K time units, if that user tried N times, then K*N time units are needed. By contrast, in the case of supporting e-mail messages, and assuming the user needs to check the e-mail messages anyhow, the user will set up the system as soon as he/she has received an e-mail message regarding the response for his demand (i.e K units of time are needed).

An example of the second group is when the users are provided with a function for creating an annotation. Such a function is necessary for document creation. With regard to the power of the system, the functions of the second group are of more interest. Based on the experiments and the case studies which were per-

formed in this research, some of the functions which are needed in document creation are as follows:

1) Making Annotations on the different components of a document, Discussion, administrative matters, or medium.

2) Creating Discussions about the documents, administrative matters, medium, and Discussions developed from Annotations as well as creating an independent Discussion,

3) Creating Annotations and Discussions on one or more headings, individual terms or regions in a text block, or on one or more text blocks, and

4) Viewing the outline of the Triple or part of it (i.e viewing document, Discussion, or Annotation separately), according to the user's desire (such as by author or by date) and perspective.

Some of the above functions are supported by some systems, but not all of them. All of these functions are supported by this model. However, some systems support some functions which are not supported by this model. Such functions might be facilitation functions and their existence is not essential. However, considering this model overall, it can be seen that it supports most of the necessary functions for document creation.

7.3. General Conclusions

The conclusions to be drawn from this research are:

1) Making Discussion or/and Annotation on one or more groups of text blocks (Sections, Chapters, documents, ... etc) is needed. A prerequisite for such a task is to determine the boundaries for such groups of text blocks. It is very difficult, if not impossible, for such a determination to be made without using

hierarchical structures (see Chapter 6 Section 6.3).

2) In document creation, including problem exploration and authoring, both Annotation and Discussion activities are needed. Separating Annotation and Discussion from each other, or partly supporting Annotation and/or Discussion (see Table 1), does not satisfy the goals of a hypertext system. Combining both Annotation and Discussion in one single computer system is possible (see Chapter 6) and hypertext systems should support these two activities.

3) A hypertext system supporting one kind of structure only, either hierarchical or non-hierarchical, (see Table 1) does not satisfy all users' needs. Hypertext systems should support both kinds of structure separately or in conjunction with each other. Therefore, some techniques to convert non-hierarchical to hierarchical structures is necessary.

4) In terms of system support for typing, it is not sufficient to only support a fixed set of types, including Annotation types, Discussion types, and node and link types. Nor it is sufficient to only support user-defined types. These design options do not satisfy user needs. The hypertext systems should support a specific set of types with the option to modify (add and/or delete) the existing types.

5) It should be possible for users to view documents stored in the system from different perspectives, including their own. Many traversal techniques were used to convert a graph into the hierarchy of an outline, but these techniques, separately or in conjunction, failed to provide for all users' perspectives. However, applying Annotations to the output of any traversal technique can satisfy most users' perspectives when viewing the document.

7.4. Future Work

The final objective is to propose future work which is explained in this Section. There are a number of ways in which the current research described in this research could be continued. Future work can be classified into two kinds of plans: long term plan and short term plan.

With regard to the long term plan, a number of extensions can be made to this research. First of all, investigation needs to be made into the knowledge structuring strategies. Exploring such strategies will reduce the need to develop a new system for each new task. For instance, assume there are two tasks T_1 and T_2, and T_1 needs hierarchical syntax structure, while T_2 needs non-hierarchical syntax structure. Having the strategy that systems should support both hierarchical and non-hierarchical syntax structure, and providing some methods to convert a non-hierarchy to a hierarchy, will facilitate developing one system that satisfies both tasks. These strategies may apply to all phases of collaborative hypertext activity.

Sometimes the text of the Triple Activities is too large for a user to explicitly represent without the help of a data manager. The "Argument Representation Language" (ARL) has been developed to characterize the semantics of textual constituents of a discussion or argument[Bernstein1989] and can be applied to the texts of the Triple Activities. Using ARL can help users to choose building blocks for the arguments. The ARL statement corresponding to "Claim C_1 supports Claim C_2" uses the formal term *claims* and relates two such terms to the formal predicate *supports*. Presenting the different relationships in the text to the users may have important ramifications with regard to extracting the resolution from the text, and in the accuracy of decision-making. It is difficult to make an accurate decision without developing a shared understanding of the issues among the participants, creating a

sense of common purpose, and gaining a commitment to action. These factors might be facilitated by using ARL.

The decision-making might be facilitated by different factors; the hierarchy of people might be one of these factors. It is difficult, sometimes, to arrive at a decision when a decision-maker needs to consult each one of a large number of participants. One solution in such a case might involve the hierarchy of people. In this approach, participants will be divided into a number of groups, and in turn each group might be divided into sub-groups until further division is no longer needed. These divisions might be based on different factors such as user importance or experience. Dividing the groups into sub-groups allows the decision-maker to consult the group leaders rather than consulting every user.

Looking at the hierarchy of people opens up a further area of work. The role of the other hierarchies, such as the hierarchical structure of systems, can be examined. In an approach such as this, the system can be divided into levels of abstraction. The number of levels then depends on the system and on the task. The decomposition of the system might start from its most global description and progressively obtain a more and more detailed description in the hierarchy. The role of this type of hierarchy, and others, can be explored to see how they might facilitate using such a system.

Using the computer, however easy it is to use, will not eliminate the need for face-to-face discussion, though not necessarily in the physical sense, eg. via video, audio, or the telephone. Therefore, integrating some media such as video and audio needs to be considered. Such consideration might solve many problems, but might also cause new problems.

With regard to the short term plan, software can be developed for the different algorithms that have arisen in this research:

1) the capturing resolution algorithm that is described in Section 6.2.1.1. The algorithm can be tested, and then the strengths and the weaknesses of the algorithm can be explored. In another extension to this work, this algorithm and ARL might be integrated;

2) the organization algorithm that is described in Section 4.5.1. It has been seen that the re-organization process needs to be done manually, whereas generating the outline is done automatically. After developing the software, this algorithm needs to be tested and evaluated. It has also been explained that side-effects such as confusion might arise, so imposing some control is necessary;

3) the RELATION algorithm that is described in Section 5.2.3 This algorithm was used as a testing tool, but it has not tested for browsing and searching. STATBUILDER and the produced index terms were tested for browsing and searching at the concept or term level. RELATION can be tested at the text block level; and

4) the index terms that are produced manually gave more accurate results than those index terms that are produced by Salton's algorithm. To improve the quality of the index terms, so that they become meaningful, simple natural language processing tools might be employed. Since grammars of natural language are now well established and lexicons are widely available, these might be exploited to improve the ability to detect meaningful index terms for paragraphs and to suggest relationships between those terms. Causing the index terms to be meaningful might improve the relationships in the semantic net that can be generated using Mili's algorithm. Such a semantic net can be used as a knowledge base.

Finally, after performing the short term plan and performing as much as possible of the long term plan, the Discussannotation model that is suggested in Chapter 7

needs to be implemented. Such implementation needs to consider the algorithms described, the work done in the long term plan, and some other useful mechanisms arising at that time from the literature. The implementation needs to be done on a fast and high resolution computer. It is then hoped that the system will be easy to learn, easy to use, and that it will satisfy most users' needs.

REFERENCES

Acar1990.
Levent Acar and Umit Ozguner, "Design of Knowledge-Rich Hierarchical Controllers for Large Functional Systems," *IEEE Transactions on Systems, Man, and Cybernetics, 20, 4*, pp. 791-803, 1990.

Addison1991.
M. A. Addison, "Generalised Hierarchical Operators and their Implementation in Spatial Data Processing and Other Applications," Ph.D. Thesis, Computer Science Department, University of Liverpool, Liverpool L69 3BX, UK, September, 1991.

Akscyn1988.
Robert Akscyn, Donald McCracken, and Elise Yoder, "KMS: A Distributed Hypermedia System for Managing Knowledge in Organizations," *Communications of the Association of Computing Machinery, 31, 7*, pp. 820-835, 1988.

Alvo1985.
Mayer Alvo and Paul Cabilio, "Average Rank Correlation Statistics in the Presence of Ties," *Communications in Statistics Theory and Methods*, pp. 2095-2108, 1985.

Begoray1990.
John A. Begoray, "An introduction to hypermedia issues, systems and application areas," *International Journak of Man-Machine Studies, 33, 2*, pp. 121-148, 1990.

Bernstein1989.
Bernard Bernstein, Paul Smolensky, and Brigham Bell, "Design of a Constraint-Based Hypertext System to Augment Human Reasoning," *Proceedings Fourth Annual Rocky Moutain Conference on Artificial Intelligence*, pp. 21-30, Rocky Moutain Society for Artificial Intelligence, Denver, Colorado, June 8-9, 1989.

Bui1984.
Tung Bui and Jarke Matthias, "A DSS for cooperative multiple criteria group decision making," *Working Paper Series*, New York Univ., New York, 1984.

Bullen1990.
Christine V. Bullen and John L. Bennett, "Learning from User Experience with Groupware," *In Proceedings of the Conference on Computer-Supported Cooperative Work*, pp. 291-301, Los Angeles, CA., October 1990.

Bush1945.
Vannevar Bush, "As We May Think," *The Atlantic Monthly, 176, 1*, pp. 101-108, July 1945.

Campbell1988.
Brad Campbell and Joseph M Goodman, "HAM: A General-Purpose Hypertext Abstract Machine," *Communications of the Association of Computing Machinery, 31, 7*, pp. 856-861, 1988.

Carbonell1984.
Jaime Carbonell and R. Frederking, "Natural Language Interfaces to Knowledge-Based Systems," in *The Factory of the Future*, ed. D Reddy, Digital Press, 1984.

Carlson1990.
Patricia Ann Carlson, "The rhetoric of hypertext," *Hypermedia, 2, 2*, pp. 109-132, 1990.

Catlin1989.
T. Catlin, P. Bush, and N. Yankelovich, "InterNote: Extending a Hypermedia Framework to Support Annotative Collaboration," *Proceedings Hypertext '89*, pp. 365-378, Association of Computing Machinery, New York, 1989.

Charney1987.
Davida Charney, "COMPREHENDING NON-LINEAR TEXT: The Role of Discourse Cues and Reading Strategies," *Hypertext '87*, pp. 109-120, University of North Carolina, Chapel Hill, North Carolina, November 13-15, 1987.

Conklin1987a.
Jeff Conklin and Michael Begeman, "gIBIS: A Hypertext Tool for Team Design Deliberation," *Hypertext '87*, pp. 247-252, University of North Carolina, Chapel Hill, North Carolina, November 13-15, 1987.

Conklin1987b.
Jeff Conklin, "Hypertext: An Introduction and Survey," *Computer, 20, 9*, pp. 17-41, September 1987.

Conklin1988.
Michael L. Begeman , "gIBIS: A hypertext tool for exploratory policy discussion," *ACM Trans. Office Information Systems 6, 4*, pp. 303-331, October 1988.

Conklin1989.
Jeff Conklin and Michael Begeman , "gIBIS: A Tool for All Reasons," *Journal of the American Society of Information Science, 40, 3*, pp. 200-213, 1989.

Cooper1989.
M. C. Cooper, "Formal Hierarchical Object Models for Fast Template Matching," *The Computer Journal, 32, 4*, pp. 351-361, 1989.

Czuchry1988.
Andrew J Czuchry, Jr and David R Harris, "KBRA: A New Paradigm for Requirements Engineering," *IEEE Expert*, pp. 21-35, Winter 1988.

Dam1988.
Andries van Dam, "Hypertext '87 Keynote Address," *Communications of the Association of Computing Machinery, 31, 7*, pp. 887-895, July 1988.

Delisle1986.
Norman Delisle and Mayer Schwartz, "Neptune: a Hypertext System for CAD Applications," *Proceedings of ACM SIGMOD '86: International Conference on Management of Data*, pp. 132-139, Association of Computing Machinery, New York, 1986.

Desanctis1987.
Gerardine Desanctis and R. Brent Gallupe, "A Foundation for the Study of Group Decision Support Systems," *Management Science, 33, 5*, pp. 589-606, May 1987.

Duchastel1990.
Philippe C. Duchastel, "Examining Cognitive Processing in hypermedia usage," *Hypermedia, 2, 3*, pp. 221-234, 1990.

Dumais1988.
S. T. Dumais, G. W. Furnas, T. K. Landauer, S. Deerwester, and R. Harshman, "Using Latent Semantic Analysis to Improve Access to Textual Information,"

CHI'88 Conference Proceedings, pp. 281-285, ACM Press, New York, 1988.

Faloutsos1990.
Christos Faloutsos, Raymond Lee, Catherine Plaisant, and Ben Shneiderman, "Incorporating string research in a hypertext system: user interface and signature file design issues," *Hypermedia, 2, 3*, pp. 183-200, 1990.

Fischer1989.
Gerhard Fischer, Raymond McCall, and Anders Morch, "JANUS: Integrating Hypertext with a Knowledge-Based Design Environment," *Proceedings Hypertext '89*, pp. 105-118, Association of Computing Machinery, New York, 1989.

Fish1988.
Robert S. Fish, Robert E. Kraut, Mary D. P. Leland, and Michael Cohen, "Quilt: a Collaborative Tool for Cooperative Writing," *Proceedings Conference on Office Information Systems*, pp. 30-37, Association Computing Machinery, New York, 1988.

Frederiksen1981.
Carl H. Frederiksen and Joseph F. Dominic, "Introduction: Perspectives on the Activity of Writing," in *Writing: the Nature, Development, and Teaching of Written Communication, Volume 2*, ed. J F Dominic, pp. 1-20, Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1981.

Frisse1988.
Mark E. Frisse, "Searching for Information in a Hypertext Medical Handbook," *Communications of the Association for Computing Machinery, 31, 7*, pp. 880-886, July 1988.

Furuta1989.
Richard Furuta and P. David Stotts, "Programmable Browsing Semantics in Trellis," *Hypertext '89 Proceedings*, pp. 27-42, Pittsburgh, Pennsylvania, November 1989.

Gale1990.
Stephen Gale, "Human Aspects of Interactive Multimedia Communication," *Interacting with Computers, 2, 2*, pp. 175-189, 1990.

Ghaoui1991.
Claude Ghaoui, Steven M. George, Roy Rada, Martin Beer, and Januz Getta , "Text to Hypertext and Back Again," *Computers and Writing III*, Edinburgh, Scotland, April, 6-7, 1990.

Hahn1989.
Udo Hahn, Matthias Jarke, Klaus Kreplin, Marisa Farusi, and Francesco Pimpinelli, "CoAUTHOR: A Hypermedia Group Authoring Environment," *Proceedings of First European Conference on Computer Supported Cooperative Work*, pp. 226-244, 1989.

Halasz1987.
Frank G. Halasz, T. P. Moran, and Randall H Trigg, "NoteCards in a Nutshell," *Proceedings of the ACM CHI+GI Conference*, pp. 45-52, Toronto, Ontario, April 1987.

Halasz1988.
Frank G. Halasz, "Reflections on Notecards: Seven Issues for the Next Generation of Hypermedia Systems," *Communications of the Association of Computing Machinery, 31, 7*, pp. 836-855, 1988.

Halaz1990.
Frank Halaz and Mayer Schwartz, "The Dexter Hypertext Reference Model," *Proceedings of the Hypertext Standardization Workshop*, pp. 95-133, National Institute of Standards and Technology Special Publication 500-178, U. S. Government Printing Office, Washington, DC, 1990.

Hashim1990.
Safaa H. Hashim, "Exploring Hypertext Programming," in *Writing Knowledge Representation and Problem-Solving Programs*, ed. N Sharkey, Windcrest Books, U.S.A, 1990.

Hayes-Roth1979.
B. Hayes-Roth and F. Hayes-Roth, "A cognitive model of planning," *Cognitive Science, 3, 4*, pp. 275-310, 1979.

Hershey1985.
William Hershey, "Software Review: Idea Processors," *BYTE*, pp. 337-351, June 1985.

Horton1989.
William K. Horton, in *Designing and Writing online Documentation: Help Files to Hypertext*, John Wiley & Sons, Inc., U.S.A, 1989.

Jones1980.
Karen Sparck Jones, "A Statistical Interpretation of Term Specificity and its Application in Retrieval," in *Key Papers in Information Science*, ed. Belver C Griffith, pp. 305-315, Knowledge Industry Publications, White Plains, New York, 1980. also appeared 1972 Journal of Documentation, 28, 1

Jordan1989.
Daniel S. Jordan, Daniel M. Russell, Anne-Marie S. Jensen, and Russell A. Rogers, "Facilitating the development of representations in hypertext with IDE," *Hypertext '89 Proceedings*, pp. 93-104, Pittsburgh, Pennsylvania, November 1989.

Kaplan1990.
Simon M. Kaplan and Yoelle S. Maarek, *Incremental Maintenance of Semantic Links in Dynamically Changing Hypertext Systems,* University of Illinois at Urbana-Champaign, 1990.

Kelogg1987.
R. T. Kelogg, "Effects of topic knowledge on the allocation of processing time and cognitive effort to writing processes," *Memory and Cognition, 15, 3*, pp. 256-266, 1987.

Kintsch1988.
Walter Kintsch, "The Role of Knowledge in Discourse Comprehension: A Construction-Integration Model," *Psychological Review, 95, 2*, pp. 163-132, 1988.

Knuth1973.
Donald Ervin Knuth, *The Art of Computer Programming,* Vol. 1, Addison-Wesly, Reading (Mass.), 1973.

Knuth1990.
Randy A. Knuth and Thomas A. Brush, "Results of the Hypertext '89 Design Survey," *Hypermedia, 2, 2*, pp. 91-108, 1990.

Kraemer1988.
Kenneth L. Kraemer and John Leslie King, "Computer-Based Systems for Cooperative Work and Group Decision Making," *ACM Computing Surveys, Vol. 20, No. 2*, pp. 115-146, June 1988.

Krzywiec1983.
Roberk K. Krzywiec, "Multi ... Multi-Information," *Proceedings of the 14th Annual International Modeling and Simulation Conference*, University of Pittsburg, Pittsburg, Pennsylvania, 1983.

Kunz1970.
Werner Kunz and Horst Rittel, "Issues as elements of information systems," *Working paper #131*, Institute fur Grundlagen der planung I.A. University of Stuttgart, 1970.

Lebowitz1986.
Michael Lebowitz, "Integrated Learning: Controlling Explanation," *Cognitive Science, 10*, pp. 219-240, 1986.

Leland1988.
Mary D. P. Leland, Robert S. Fish, and Robert E. Kraut, "Collaborative Document Production Using Quilt," *Second Conference on Computer-Supported Cooperative Work: CSCW '88*, pp. 206-215, 1988.

Lesk1969.
M. E. Lesk, "Word-Word Associations in Document Retrieval Systems," *American Documentation, 20, 1*, pp. 27-38, January 1969.

Liou1989.
Yihwa Irene Liou and Jay F. Nunamaker, "A Computer Supported Cooperative Approach to Knowledge Acquisition from Multiple Experts," *Augmenting Human Intellect by Computer Conference Proceedings*, Denver, Colorado, June 8 - 9, 1989.

Lowe1985.
David G. Lowe, "Cooperative Structuring of Information: The Representation of Reasoning and Debate," *Int'l. J. of Man-Machine Studies, Vol 23*, pp. 97-111, 1985.

Luhn1958.
H. P. Luhn, "The Automatic Creation of Literature Abstracts," *IBM Jr Res Devel, 2, 2*, pp. 159-165, Apr 1958.

Marshall1987.
Catherine C. Marshall, "Exploring representation problems using hypertext," *In: Proceedings of the Hypertext '87 Workshop, Chapel Hill*, pp. 253-268, North Carolina, November 1987.

Maurer1990.
Hermann Maurer and Ivan Tomek, "Broadening the scope of hypermedia principles," *Hypermedia, 2, 3*, pp. 201-220, 1990.

McCall1987.
R. McCall, "PHIBIS: Procedurally Hierarchical Issue-Based Information Systems," *Proceedings of the Conference on Architecture at the International Congress on Planning and Design Theory*, American Society of Mechanical Engineers,, New York, 1987.

McCracken1984.
Donald McCracken and Robert Akscyn, "Experience with the ZOG Human-Computer Interface System," *International Journal of Man-Machine Studies, 21*, pp. 293-310, 1984.

McKnight1991.
Cliff McKnight, Andrew Dillon, and John Richardson, in *Hypertext in Context*, pp. 65-87, The Cambridge Series on Electronic Publishing, November 1991.

Mhashi1990.
Mahmoud Mhashi, Roy Rada, Hafedh Mili, Geeng-Neng You, Akmal Zeb, and Antonis Michailidis , "Word Frequency Based-Indexing and Authoring," *Computers and Writing III*, Edinburgh, Scotland, April, 6-7, 1990.

Mhashi1991.
Mahmoud M. Mhashi, Roy Rada, Eevi E. Beck, Akmal Zeb, and Antonios Michailidis, "Computer-Supported Discussion and Annotation," *Technical Report*, Department of Computer Science, University of Liverpool, Liverpool, U.K, May 1991.

Mili1985.
Hafedh Mili and Roy Rada, "A Statistically Built Knowledge Base," *Proceedings Expert Systems in Government Conference*, pp. 457-463, IEEE Computer Society Press, October 1985.

Mili1987.
Hafedh Mili and Roy Rada, "Building a Knowledge-Base for Information Retrieval," *Proceedings Third Annual Expert Systems in Government Conference*, pp. 12-18, Computer Society of the IEEE, Washington, D.C., 1987.

Nelson1980.
T.H. Nelson, "Replacing the Printed Word: A Complete Literary System," *IFIP Proc.*, pp. 1013-1023, October 1980.

Neuwirth1990.
Christine M. Neuwirth, David S. Kaufer, Ravinder Chandhok, and James H. Morris, "Issues in the Design of Computer Support for Co-authoring and Commenting," *Proceedings of the Conference on Computer-Supported Cooperative Work*, pp. 183-195, Los Angeles, CA, October 7-10, 1990.

Rada1987.
Roy Rada and Brian Martin, "Augmenting Thesauri for Information Systems," *ACM Transactions on Office Information Systems, 5, 4*, pp. 378-392, 1987.

Rada1989a.
Roy Rada and Ellen Bicknell, "Ranking Documents with a Thesaurus," *Journal of the American Society for Information Science, 40, 5*, pp. 304-310, September 1989.

Rada1989b.
Roy Rada, Barbara Keith, Marc Burgoine, Steven George, and David Reid, "Collaborative Writing of Text and Hypertext," *Hypermedia, 1, 2*, pp. 93-110, 1989.

Rada1990.
Roy Rada, Mahmoud Mhashi, and Judith Barlow, "Hierarchical Semantic Nets Support Retrieving and Generating Hypertext," *Information and Decision Technologies, 16*, pp. pp 117 - 135, North-Holland, 1990.

Rada1991a.
Roy Rada, *Hypertext from Text to Expertext*, , McGraw-Hill, London, 1991.

Rada1991b.
Roy F. Rada, Akmal Zeb, Geeng-Neng You, Antonios Michailidis, and Mahmoud M. Mhashi, "Collaborative hypertext and the MUCH system," *Journal of Information Science, 17*, pp. 191-196, 1991.

Raghavan1979.
Vijay Raghavan and C T Yu, "Experiments on the Determination of the Relationships between Terms," *ACM Transactions on Database Systems, 4, 2*, pp.

240-260, 1979.

Rasmussen1985.
Jens Rasmussen, "The Role of Hierarchical Knowledge Representation in Decisionmaking and System Management," *IEEE Transactions on Systems, Man, and Cybernetics, 15, 2*, pp. 234-243, April 1985.

Remde1987.
Joel R. Remde, Louis M. Gomez, and Thomas K Landauer, "SuperBook: An Automatic Tool for Information Exploration--Hypertext?," *Hypertext '87*, pp. 175-188, University of North Carolina, Chapel Hill, North Carolina, November 13-15, 1987.

Rijsbergen1979.
C J Van Rijsbergen, *Information Retrieval*, pp. 18-21, Butterworths, London, 1979.

Ritchie1989.
Ian Ritchie, "HYPERTEXT--Moving Towards Large Volume," *The Computer Journal, 23, 6*, pp. 516-523, December 1989.

Rizk1990.
A. Rizk, N. Streitz, and J. Andre, "A Model for Hypertext-Based Information Retrieval," in *Hypertext: Concepts, Systems and Applications*, pp. 81-94, The Cambridge Series on Electronic Publishing, November 1990.

Salton1983.
Gerard Salton and Michael McGill, *Introduction to Modern Information Retrieval*, McGraw-Hill, New York, 1983.

Smith1987.
John B. Smith, Stephen F. Weiss, and Gordon F. Ferguson, "A Hypertext Writing Environment and its Cognitive Basis," *Hypertext '87*, University of North Carolina, Chapel Hill, North Carolina, November 13-15, 1987.

Soergel1974.
Dagobert Soergel, *Indexing Languages and Thesauri: Construction and Maintenance*, Wiley, New York, 1974.

Stallman1981.
Richard Stallman, "EMACS Manual for TWENEX Users," *AI Memo 555*, MIT AI Lab, Cambridge, Massachusetts, October 1981.

Stefik1987.
Mark Stefik, Gregg Foster, Daniel Bobrow, Kenneth Kahn, Stan Lanning, and Lucy Suchman, "Beyond the Chalkboard: Computer Support for Collaboration and Problem Solving in Meetings," *Communications of the ACM, 30, 1*, pp. 32-47, 1987.

Steward1987.
Donald V. Steward, *Software Engineering: Approach to Systems Analysis and Design*, Wadsworth Publishing, Delmont, California, 1987.

Streitz1989.
Norbert A. Streitz, Jorg Hannemann, and Manfred Thuring, "From Ideas and Arguments to Hyperdocuments: Traveling through Activity Spaces," *Hypertext '89 Proceedings*, pp. 343-364, Pittsburgh, Pennsylvania, November 1989.

Trigg1986a.
Randall Trigg, Lucy Suchman, and Frank Halasz, "Supporting Collaboration in NoteCards," *Proceedings of the Conference on Computer Supported Cooperative*

Work, pp. 1-10, December 3-5, 1986.

Trigg1986b.
Randall H. Trigg and Mark Weiser, "TEXTNET: A Network-Based Approach to Text Handling," *ACM Transactions on Office Information Systems, 4, 1*, pp. 1-23, 1986.

Walker1988.
Janet Walker, "Supporting Document Development with Concordia," *Computer, 21, 1*, pp. 48-60, January 1988.

Yankelovich1985.
Nicole Yankelovich, Norman Meyrowitz, and Andries van Dam, "Reading and Writing the Electronic Book," *Computer*, pp. 15-30, October 1985.

Yankelovich1988.
Nicole Yankelovich, Karen E. Smith, L. Nancy Garrett, and Norman Meyrowitz, "Issues in Design a Hypermedia Document System," in *Interactive Multimedia*, ed. Kristina Hooper, pp. 33-86, Harper & Row, Washington, 1988.

Yeh1980.
R. T. Yeh and P. Zave, "Specifying Software Requirements," *Proceedings IEEE, 68 (9)*, pp. 1077-85, 1980.

Zeb1991.
Akmal Zeb, Antonios Michailidis, Geeng-Neng You, Mahmoud Mhashi, and Roy Rada, "MUCH: A UNIX-based Hypertext System," *UKUUG: Interfacing Unix to the User Conference Proceedings*, Greenbank Halls, University of Liverpool, July 15-17, 1991.

Zellweger1989.
Polle T. Zellweger, "Scripted documents: a hypermedia path mechanism," *Hypertext '89 Proceedings*, pp. 1-14, Pittsburgh, Pennsylvania, November 1989.

Appendix 1

List of Figures

Figure 1 : Viewing nodes and links 24
Figure 2 : Emacs INFO 27
Figure 3 : Hyperties Example 31
Figure 4 : Internote Screen 34
Figure 5 : Push or Pull 35
Figure 6 : KMS Screen 37
Figure 7 : Dexter Model 43
Figure 8 : Software life cycle 46
Figure 9 : HERD's nodes and links 50
Figure 10: HERD's Fill-form screen 53
Figure 11: HERD's Title-Information 55
Figure 12: HERD's Author-Date-Information 56
Figure 13: HERD's Discourse representation structure ... 57
Figure 14: HERD's text screen 58
Figure 15: Diagram of Discussion 61
Figure 16: Predominant Links 61
Figure 17: Annotation outline using HERD 66
Figure 18: Suggested links 69
Figure 19: First MUCH screen 73
Figure 20: MUCH entity relationship 76
Figure 21: Annotation mode 81
Figure 22: Experiment Annotation outline 86
Figure 23: Experiment Annotation tree 87
Figure 24: Re-organization example 91
Figure 25: Organization experiment outline (A) 98
Figure 26: Organization experiment outline (B) 99
Figure 27: Organization experiment outline (C) 102
Figure 28: Sections correspondent text blocks 117
Figure 29: Document hierarchy 118
Figure 30: Document relationships 119
Figure 31: Discussion hierarchy 124
Figure 32: Discussion relationships 125
Figure 33: Annotation tree 126
Figure 34: Annotation hierarchy 127
Figure 35: Annotation relationships 128
Figure 36: Discussannotation Levels 147
Figure 37: Activities Relationships 148
Figure 38: Document Text Block 152
Figure 39: Discussion and Annotation on heading 153
Figure 40: Traversal 156
Figure 41: Different views and undefined boundaries 159
Figure 42: Composition 161
Figure 43: Anchoring 162
Figure 44: storage-presentation importance 163
Figure 45: Discussannotation Model 165

Figure 46: Capturing resolution 177
Figure 47: Text and link object (A) 186
Figure 48: Text and link object (B) 187

Appendix 2

Glossary

Anchoring

The *anchoring* method defines the elements between which a link is established. The source of a link can be either a term, a span, or a region of text in a text block. While the destination of a link can be either as the source, a node, or a *composition*.

Annotation

After the first draft of a document is prepared, annotations are collected to guide revision of the document. Annotating concerns reading the evolving text, evaluating the text and plans, and correcting errors. Annotation hypertext systems focus on recording ideas dynamically generated while reading text, including critique, explaining difficult passages, sorting user-produced mnemonic aids, and communicating with the library manager and/or other users.

Argument

Argument is a node type. The node's content, in IBIS method, is one or more text block constructed in support or object the different *Positions* until the *Issue* is resolved.

Browsing

Browsing is often thought of as a relatively free-flowing meandering through the information in a hyperdocument collection. It can be characterized as going from *where to what*; presumably the readers might know where they are in the database and they want to know what is there.

Decision-Making

If a conflict happened between users, the decision-making is the best solution for such a conflict. Decision-making involves two or more persons elaborating on the nature of a problem, generating and evaluating potential solutions, and formulating strategies for their implementation. Each of these persons is characterized by his or her own perceptions, attitudes, motivations, and personality.

Discussannotation

The Discussannotation hypertext model is a computer-based tool that supports facilitating Discussion and making Annotation in the creation of documents by one or several users. Different views and generating different outlines are supported by this model. The model is different from most existing hypertext systems in that the text is attached to the links rather than be attached to the nodes.

Discussion

A Discussion might be described as a search through hierarchy of knowledge states, with the ultimate goal being to provide a solution to the problem. A knowledge state is the information researchers know or postulate when they are at a particular stage in their search for a solution. Furthermore, Discussions might be described as issue-based information. They might also include models for planning and problem-solving.

Issue

An *Issue* is a node type. The node's content is written on a form of question, such as "What kind of computers our department should by?". Any problem or concern can be an *Issue*, and may require Discussion.

Planning

The planning model would support the determination of document requirements and general goals, the setting up of an agenda, and the co-ordination of the whole authoring activities.

Position

Within the context of the discussion systems, *Position* is a node type. The node's content is a statement or assertion which resolves an *issue*.

Problem-Solving

The Problem-solving model might serve as the medium for exploring the capture of design history (the decisions, rejected options, and tradeoff analysis). it might also serve as a conversation among the stakeholders (designers, authors, or discussants), in which they bring their respective expertise and viewpoints to the resolution of design problems.

Searching

People usually think of information search in the relatively strict sense of starting with a specific question in mind and interacting with the information base to find the answer in a fraily straightforward manner. Searching can be characterized as going from *what to where*; presumably the users know what they want and they wish to find where in the database it is.

Triple

The discussannotation model supports three types of entities, each of which is associated with an activity. To facilitate reference to these three entities, the term *Triple* is used and refers in this context to the three entities: document, Discussion, and Annotation.

Appendix 3
HERD Algorithms

Main Program

```
;; THE MAIN PROGRAM FOR CREATING AND LINKING A NODE


;;  Get a region of text, where a1 is the starting point and a2 is
;;  the end point.

(defun get-sub-string (a1 a2)
    (buffer-substring a1 a2))

;;  This procedure will be invoked at the beginning of the Discussion.
;;  The root should be an issue.

(defun create-root ()
    (setq node-name "I[1]")
    (setq where-text-to-be-inserted "!")
)

;; This subroutine validates the link type connected to the root

( defun Is-this-is-the-first-node (This-is-the-first-node)
     (if (equal This-is-the-first-node "FALSE") (find-file "fill_form")

;;      Otherwise display the following error message:

        (find-file "error_message_file")
        (kill-line)
        (insert "There is an error in fill_form file")
        (save-buffer)
        (shell-command "cp display_the_mistake_original display_the_mistake")
        (find-file "display_the_mistake")
        (kill-line 2)
        (search-forward  "ERROR DURING VALIDATION PROCESS" (point-max) t)
        (newline 2)
        (insert "Since this the first node, the content of Edge-type should be")
        (newline)
        (insert "G   or  Generated-by  ")
        (newline)
        (save-buffer)
        (interrupt-process)
     )
)

;;  This validates the link type connected to an issue destination.

(defun Is-node-parent-an-Issue(edge-type-contents This-is-the-first-node)
    (Is-this-is-the-first-node  This-is-the-first-node)
    (find-file "type_of_node")
    (goto-char (point-min))
    (search-forward Node-parent-content (point-max) t)
    (setq a1 (point)) (forward-char 1)
    (setq a2 (point))
    (setq edge-type-from-file (get-sub-string a1 a2))
    (if (equal edge-type-from-file "I") (find-file "fill_form")
;;      Otherwise display the following error message.
        (find-file "error_message_file")
        (kill-line)
        (insert "There is an error in fill_form file")
        (save-buffer)
        (shell-command "cp display_the_mistake_original display_the_mistake")
        (find-file "display_the_mistake")
        (kill-line 2)
        (search-forward  "ERROR DURING VALIDATION PROCESS" (point-max) t)
        (newline 2)
        (insert "Since the content of Edge-type is " edge-type-contents " then,")
        (newline)
```

```
        (insert "the Node-parent should be an Issue ") (newline)
        (save-buffer)
        (interrupt-process)
    )
)

;; The project name should be identified before creating any node.
;; The node parent should be the word "Root" in the case of creating
;; the first node.

(defun No-any-node-created-yet()
    (find-file "error_message_file")
    (kill-line)
    (insert "There is an error in fill_form file")
    (save-buffer)
    (shell-command "cp display_the_mistake_original display_the_mistake")
    (find-file "display_the_mistake")
    (search-forward "ERROR DURING VALIDATION PROCESS" (point-max) t) (newline 2)
    (insert "There is no any node created yet. Node-parent should be the word Root")
    (save-buffer)
    (interrupt-process)
)
```

Security Function

```
;; The user should be known to the system.

(defun Author-error-message()
    (find-file "error_message_file")
    (kill-line)
    (insert "There is an error in fill_form file")
    (save-buffer)
    (shell-command "cp display_the_mistake_original display_the_mistake")
    (find-file "display_the_mistake")
    (kill-line 2)
    (search-forward "ERROR DURING VALIDATION PROCESS" (point-max) t)
    (newline 2)
    (insert "The content of the field Author should be one of the following:")
    (newline 2)
    (insert "Akmal Zeb  Kenny  Mahmoud Mhashi  Prof. Roy Rada  Judith Barlow")
    (newline 2)
    (insert "If you would like to make any modification on your name, visit the file")
    (newline)
    (insert "called cn23, then search to your name, and make the modification you want.")
    (newline)
    (save-buffer)
    (interrupt-process)
)

(find-file "Link_flag")
(setq a1 (point)) (forward-word 2)
(setq a2 (point))
(setq link-flag (get-sub-string a1 a2))

;; If someone already linking a new node, then lock Link-Node option

(if (equal link-flag "Busy") (find-file "display_link_busy_message")
   (beginning-of-line) (kill-line) (insert "Busy")
   (save-buffer)
   (find-file "fill_form")
```

Validation Function

```
(search-forward "Node-parent: " (point-max) t)
(beginning-of-line)
(setq a1 (point)) (search-forward ":" (point-max) t)
(setq a2 (point))

;; This validates the "Node-parent:" field.

(setq Node-parent (get-sub-string a1 a2))
    (if (equal Node-parent "Node-parent:")  (setq a1 (point))
        (find-file "error_message_file")
        (kill-line)
        (insert "There is an error in fill_form file")
```

```
        (save-buffer)
        (shell-command "cp display_the_mistake_original display_the_mistake")
        (find-file "display_the_mistake")
        (search-forward "ERROR DURING VALIDATION PROCESS" (point-max) t)
        (newline 2)
        (insert "The first field in fill-form file should be Node-parent: ")
        (save-buffer)
        (interrupt-process)
    )

;; This validates the "Node-parent:" field content.

(forward-word 1) (setq a2 (point))
(setq Node-parent-content (concat (get-sub-string a1 a2) " "))
(if (equal Node-parent-content " Root ") (find-file "type_of_node")
  (find-file "type_of_node")
  (if (eobp) (No-any-node-created-yet)
    (goto-char (point-min))
    (search-forward Node-parent-content  (point-max) t)
    (beginning-of-line)
    (setq a1 (point)) (forward-word 1)
    (forward-char 1) (setq a2 (point))
    (setq Node-parent-from-file (get-sub-string a1 a2))
    (if (equal Node-parent-content  Node-parent-from-file) (find-file "type_of_node")
        (if (eobp) (shell-command "cp display_the_mistake_original display_the_mistake")
            (find-file "error_message_file")
            (kill-line)
            (insert "There is an error in fill_form file")
            (save-buffer)
            (shell-command "cp display_the_mistake_original display_the_mistake")
            (find-file "display_the_mistake")
            (search-forward "ERROR DURING VALIDATION PROCESS" (point-max) t)
            (newline 2)
            (insert "The Node-parent" Node-parent-content "doesn't exist in the graph")
            (save-buffer)
            (interrupt-process)
        )
        (find-file "error_message_file")
        (kill-line)
        (insert "There is an error in fill_form file")
        (save-buffer)
        (find-file "display_the_mistake")
        (search-forward "ERROR DURING VALIDATION PROCESS" (point-max) t)
        (newline 2)
        (insert "Since this is the first node, the content of field Node-parent")
        (newline)
        (insert "in fill-form file should be the word Root")
        (save-buffer)
        (interrupt-process)
    )
  )
)

(goto-char (point-min))
(if (equal Node-parent-content " Root ")
    (if (eobp) (setq This-is-the-first-node "TRUE")
        (find-file "error_message_file")
        (kill-line)
        (insert "There is an error in fill_form file")
        (save-buffer)
        (shell-command "cp display_the_mistake_original display_the_mistake")
        (find-file "display_the_mistake")
        (search-forward "ERROR DURING VALIDATION PROCESS" (point-max) t)
        (newline 2)
        (insert "Since this is not  the first node, the content of field Node-parent")
        (newline)
        (insert "in fill-form file should be one of the numbers appearing on the graph")
        (save-buffer)
        (interrupt-process)
    )
    (setq This-is-the-first-node "FALSE")
)

;;      This validates the "Edge-type:" field.
```

```
(find-file "fill_form")
(search-forward "Edge-type: " (point-max) t)
(beginning-of-line)
(setq a1 (point))
(search-forward ":" (point-max) t)
(setq a2 (point))
(setq Edge-type (get-sub-string a1 a2))
(if (equal Edge-type "Edge-type:") (setq a1 (point))
   (find-file "error_message_file")
   (kill-line)
   (insert "There is an error in fill_form file")
   (save-buffer)
   (shell-command "cp display_the_mistake_original display_the_mistake")
   (find-file "display_the_mistake")
   (search-forward "ERROR DURING VALIDATION PROCESS" (point-max) t)
   (newline 2)
   (insert "The second field in fill-form file should be Edge-type: ")
   (save-buffer)
   (interrupt-process)
)

;; This validates the "Edge-type:" field content.

(end-of-line)
(backward-word 1) (forward-word 1)
(setq a2 (point))
(setq edge-type-contents (get-sub-string a1 a2))
(if (equal edge-type-contents " R")
   (Is-node-parent-an-Issue edge-type-contents This-is-the-first-node)
   (if (equal edge-type-contents " Responds-to")
      (Is-node-parent-an-Issue edge-type-contents This-is-the-first-node)
      (if (equal edge-type-contents " G")
         (find-file "fill_form")
         (if (equal edge-type-contents " Generated-by")
            (find-file "fill_form")
            (if (equal edge-type-contents " Supports-to")
               (Is-this-is-the-first-node This-is-the-first-node)
               (if (equal edge-type-contents " other")
                  (Is-this-is-the-first-node This-is-the-first-node)
                  (find-file "error_message_file")
                  (kill-line)
                  (insert "There is an error in fill_form file")
                  (save-buffer)
                  (shell-command "cp display_the_mistake_original display_the_mistake")
                  (find-file "display_the_mistake")
                  (kill-line 2)
                  (search-forward "ERROR DURING VALIDATION PROCESS" (point-max) t)
                  (newline 2)
                  (insert "The content of Edge-type " edge-type-contents " doesn't exist.")
                  (newline)
                  (insert "Edge type should be one of the following: ")
                  (newline 2)
                  (insert "R  Responds-to  G  Generated-by  Supports-to other ")
                  (newline 2)
                  (insert "For more information read the some memo written")
                  (newline)
                  (insert "at the beginning of the file called fill_form.")
                  (newline)
                  (save-buffer)
                  (interrupt-process)
               )
            )
         )
      )
   )
)

;; This validates the "Author:" field.

(search-forward "Author: " (point-max) t)
(beginning-of-line)
(setq a1 (point))
(search-forward ":" (point-max) t)
(setq a2 (point))
(setq Author (get-sub-string a1 a2))
```

```
(if (equal Author "Author:") (setq a1 (point))
    (find-file "error_message_file")
    (kill-line)
    (insert "There is an error in fill_form file")
    (save-buffer)
    (shell-command "cp display_the_mistake_original display_the_mistake")
    (find-file "display_the_mistake")
    (search-forward "ERROR DURING VALIDATION PROCESS" (point-max) t)
    (newline 2)
    (insert "The third field in fill-form file should be Author: ")
    (save-buffer)
    (interrupt-process)
)

;; This validates the "Author:" field content.

(end-of-line)
(backward-word 1) (forward-word 1)
(setq a2 (point))
(setq Author-content (get-sub-string a1 a2))
(if (equal Author-content " Akmal Zeb") (find-file "fill_form")
    (if (equal Author-content " Kenny") (find-file "fill_form")
        (if (equal Author-content " Mahmoud Mhashi") (find-file "fill_form")
            (if (equal Author-content " Prof. Roy Rada") (find-file "fill_form")
                (if (equal Author-content " Judith Barlow")
                    (find-file "fill_form")
                    (Author-error-message)
                )
            )
        )
    )
)

;; This validates the "Title:" field.

(search-forward "Title: " (point-max) t)
(beginning-of-line)
(setq a1 (point))
(search-forward ":" (point-max) t) (setq a2 (point))
(setq Title (get-sub-string a1 a2))
(if (equal Title "Title:")
    (setq a1 (point))
    (find-file "error_message_file") (kill-line)
    (insert "There is an error in fill_form file")
    (save-buffer)
    (shell-command "cp display_the_mistake_original display_the_mistake")
    (find-file "display_the_mistake")
    (search-forward "ERROR DURING VALIDATION PROCESS" (point-max) t)
    (newline 2)
    (insert "The forth field in fill-form file should be Title: ")
    (save-buffer) (interrupt-process)
)

;; This validates the "Title:" field content.

(search-forward "Text: " (point-max) t)
(beginning-of-line) (setq a1 (point))
(search-forward ":" (point-max) t)
(setq a2 (point))
(setq Text (get-sub-string a1 a2))
(if (equal Text "Text:")
    (setq a1 (point)) (find-file "error_message_file")
    (kill-line)
    (insert "There is an error in fill_form file")
    (save-buffer)
    (shell-command "cp display_the_mistake_original display_the_mistake")
    (find-file "display_the_mistake")
    (search-forward "ERROR DURING VALIDATION PROCESS" (point-max) t)
    (newline 2)
    (insert "The fifth field in fill-form file should be Text: ")
    (save-buffer) (interrupt-process)
)
```

Linking Function

```
;; Get date from the machine

(delete-file "Date_buf")
(shell-command "date > > Date_buf")
(find-file "Date_buf")
(copy-to-buffer "Date-buf" (point-min) (point-max))
(find-file "beginning_and_end_of_node")
(setq a1 (point)) (forward-char 10) (setq a2 (point))
(setq  stars (get-sub-string a1  a2)) (forward-line)

;; Get text

(copy-to-buffer "Text-buf" (point)  (point-max))
(goto-char (point-min))
(setq a1 (point)) (forward-line)
(append-to-buffer "Text-buf" a1 (point))
(find-file "fill_form")
(goto-char (point-min))
(append-to-buffer "Text-buf" (point-min) (point-max))

;; Get some information for later use

(find-file "fill_form")
(search-forward "Node-parent:")
(setq a1 (point)) (forward-word 1) (setq a2 (point))
(setq  parent (get-sub-string a1  a2))
(setq root (get-sub-string  a1 a2))

(search-forward "Edge-type:")
(setq beg (point)) (end-of-line)
(backward-word 1) (forward-word 1)
(copy-to-buffer "edge-buf" beg (point))
(if (equal root " Root" ) (create-root)
   (delete-file "temp_parent")
   (find-file "temp_parent")
   (insert  parent " ")
   (end-of-line)
   (insert-buffer "edge-buf" )
   (save-buffer)


;; Call labeling function

   (shell-command "Cal")

;; Distribute some information on different files for later use

   (find-file "temp_child")
   (goto-char (point-min))
   (setq a1 (point))
   (forward-list) (setq a2 (point))
   (setq node-name (get-sub-string a1 a2))
   (setq a1 (point))
   (forward-word 2) (setq a2 (point))
   (setq old-parent-child (get-sub-string a1 a2))
   (setq a1 (point))
   (forward-word 2) (setq a2 (point))
   (setq new-parent-child (get-sub-string a1 a2))
   (forward-line 1)
   (setq a1 (point))
   (forward-word 1) (forward-char 1)
   (setq a2 (point))
   (setq node-number-where-text-to-be-inserted (get-sub-string a1 a2))
   (backward-char 1)
   (setq a1 (point)) (forward-word 1)
   (forward-char 1) (setq a2 (point))
   (setq current-node-number (get-sub-string a1 a2))
   (backward-word 1)
   (setq a1 (point))
   (forward-list) (setq a2 (point))
   (setq node-type (get-sub-string a1 a2))
   (find-file "Parent_child")
   (search-forward old-parent-child (point-max) t)
```

```
    (if (bolp) (goto-char (point-max))
        (beginning-of-line)
        (kill-line)
    )
    (insert new-parent-child )
    (newline) (save-buffer)
)
(shell-command "sort Parent_child -o Parent_child_temp")
(find-file "move_num_of_lines_to_insert_title")
(kill-line) (insert node-number-where-text-to-be-inserted)
(save-buffer "move_num_of_lines_to_insert_title")
(shell-command "cp Parent_child_temp Parent_child")
(shell-command "Cal_move")
(find-file "where_title_to_be_inserted_file") (beginning-of-line)
(setq a1 (point)) (forward-word 1) (setq a2 (point))
(setq a3 (get-sub-string a1 a2))
(setq where-title-to-be-inserted (string-to-int a3))
(find-file "type_of_node") (goto-char (point-min)) (newline)
(forward-line -1)
(if (equal root " Root" ) (insert " 1 " node-name)
    (insert " " node-type)
    (save-buffer)
    (shell-command "sort type_of_node -o temp_type_of_node")
    (find-file "temp_type_of_node")
    (search-forward current-node-number (point-max) t)
    (forward-line -1)
    (beginning-of-line)
    (forward-word 2) (backward-word 1)
    (setq a1 (point)) (forward-list)
    (setq a2 (point))
    (setq where-text-to-be-inserted (get-sub-string a1 a2))
)
(save-buffer)

;; Get Author and Title information

(find-file "fill_form")
(goto-char (point-min))
(search-forward "Author:")
(setq beg (point))
(end-of-line)
(copy-to-buffer "Author-buf" beg (point))

(goto-char (point-min))
(search-forward "Title:")
(setq beg (point)) (end-of-line)
(copy-to-buffer "title-buf" beg (point))

;; Insert the text in the TEXT file

(shell-command "chmod 600 text")
(find-file "text")
(forward-line 1)

(if (equal root " Root" ) (insert-buffer "Text-buf")
    (search-forward where-text-to-be-inserted (point-max) t)
    (end-of-line)
    (search-forward "**********" (point-max) t )
    (end-of-line) (newline)
    (insert-buffer "Text-buf")
)
(end-of-line)
(insert " " node-name "   " stars)
(goto-char (point-min))
(search-forward "@@@@@@@@@" )
(beginning-of-line)
(setq beg (point))
(search-forward "Text:")
(beginning-of-line)
(kill-region beg (point)) (newline)
(beginning-of-line)
(insert "Node-name: " node-name) (newline 1)
(search-forward "**********") (end-of-line)
(insert " " node-name "   " stars)
(save-buffer)
```

```
(shell-command "chmod 444 text")

;; Insert Author information in Author_Date_Inf file

(shell-command "chmod 600 Author_Date_Inf")
(find-file "Author_Date_Inf")
(search-forward where-text-to-be-inserted )
(end-of-line)
(newline)
(insert node-name "            ")
(move-to-column 14)
(insert-buffer "Author-buf")
(end-of-line) (insert "        ")
(move-to-column 32)
(insert-buffer "Date-buf")
(save-buffer)
(shell-command "chmod 444 Author_Date_Inf")

;; Insert Title information in Title-Inf file

(shell-command "chmod 600 Title_Inf")
(find-file "Title_Inf")
(search-forward where-text-to-be-inserted )
(end-of-line) (newline)
(insert node-name "            ")
(move-to-column 14)
(insert-buffer "title-buf")
(save-buffer)
(shell-command "chmod 444 Title_Inf")

;; Save title information to be used later for generating a graph

(find-file "title_file")
(if (equal root " Root")
    (beginning-of-line)
    (goto-line  where-title-to-be-inserted)
    (end-of-line)(newline)
)
(insert node-name " ")
(end-of-line)
(insert-buffer "edge-buf")
(end-of-line)
(insert  " ") (end-of-line)
(insert-buffer "title-buf")
(save-buffer)
(shell-command "cp Parent_child_temp temp_Parent_child")

;; Prepare data for generating a graph

(shell-command "C_n_l")
(shell-command "cp fill_form fill_form_temp")
(delete-file "fill_form")
(shell-command "cp fill_form_original fill_form")
(shell-command "cp temp_type_of_node type_of_node")
(shell-command "chmod 600 Create_flag")
(find-file "Create_flag") (kill-line)
(insert "Not busy") (save-buffer)
(shell-command "chmod 400 Create_flag")

) ;; END-IF "Link-Flag = Busy"

(find-file "Link_flag")
(beginning-of-line) (kill-line)
(insert "Not busy")
(save-buffer)
```

Storage Function

```
/* C n 114.c Program*/
/* THIS PROGRAM STORES ALL THE NECESSARY INFORMATION IN ARRAYS*/

#include <stdio.h>

main()

{
   int nparents[1000], Parent, childn, i, j, k, num_of_parents, next_parent, page;
   int node_index[1000], num_of_nodes, current_node, nor_more_unnor, temp_childn;

   int page_limit[1000];
 /* k1 is the number of the first child as a parent */
   int k1, k2, last_num_of_childn, is_left_any_in_P_c, new_parent_location;
   float x1, y1, x2, y2, deltax, deltay, temp, xlocation[1000], ylocation[1000];
   char c[100];
   FILE *fopen(), *num_of_childn, *Is_parent, *ptr_final_data, *ptr_max_page;

   num_of_childn = fopen("Parent_child", "r");
   Is_parent = fopen("temp_Parent_child", "r");
   ptr_final_data = fopen("Final_data", "w");
   ptr_max_page = fopen("max_page", "w");
   num_of_parents = 0;

/*  Store all parents in an array to be used later for determing
   whether a specific child is also a parent or not.
*/
   while( (fscanf(Is_parent, "%d%d", &Parent, &childn) != EOF)) {
   nparents[num_of_parents] = Parent;
   num_of_parents += 1;                                    }
   nor_more_unnor = 0; page_limit[1]=0;
   page = 1; x1 = 300.0; y1 = 595.0; x2 = 300.0; y2 = 540.0; num_of_nodes = 0;
   node_index[num_of_nodes] = 1; xlocation[num_of_nodes] = x2, ylocation[num_of_nodes] = y2;
   fprintf(ptr_final_data, "%d %d %f %f %f  %f 0,nor_more_unnor, page, x1, y1, x2, y2 );
   is_left_any_in_P_c = 0;
   while ( (fscanf(num_of_childn, "%d%d", &Parent, &childn) != EOF)) {
   is_left_any_in_P_c += 1;
/*  Find the first child.    */
   printf("*****  The next parent will be = [%d] *********0, Parent);
   itoa(Parent,c); first_child(c);
   next_parent  = atoi(c); current_node = next_parent;
   for(i=0; i<=25; i++) { c[i] = ' '; }
/*  Check to see which the first child as a parent.  */
   j=0;
   while( (j += 1) <= childn ) {
            next_parent += 1;
            i = 0;
            while(i < num_of_parents) {
                  if (nparents[i] == next_parent) {  break;}
               i += 1;                }
               if (i < num_of_parents){
               printf("j = %d , i = %d , next_parent = [%d] 0, j, i, next_parent);
                break;              }
            }
/* Calculate the locations and store it in the Final_data
  If the parent is the first child, then do the following
  find the location of the new parent */

          i = 0;
          while(i <= num_of_nodes) {
                if(node_index[i] == Parent)  break;
                i += 1;              }
          k2 = childn/5;
          temp_childn = childn; k1 = 0;
          for ( k=0; k < temp_childn; k += 5 ) {
              if( (temp_childn - k) >= 5) childn = 5;
          if ( k >= 5 ) {
          nor_more_unnor = 1;
             if( (temp_childn - k) < 5) childn = temp_childn - k;
         x2 = 90.0;
         if(y2 - 55.2 - 10.0 < page_limit[page]){
               y2 = page_limit[page];
```

```
            page += 1;
            page_limit[page] = page_limit[page - 1] - 600;
                                    }
        else
            y2 -= 55.2;
            fprintf(ptr_final_data, "%d %d %f %f %f  %f 0,
                  nor_more_unnor, page, x1, y1, x2, y2 );
            y1 = y2; x1 = x2;
                                }

       if (childn > 1)
       deltax = 420.0/(childn - 1);
       if (childn == 1)  deltax = 0.0;
       if (nor_more_unnor == 0) {
       x1 = xlocation[i]; y1 = ylocation[i];
                                }
          if (nor_more_unnor == 2) {
         x1 = xlocation[i]; y1 = ylocation[i];
              x2 = 510.0;

/*     Check the page limit here                */
      if(last_num_of_childn < 4){
      if(y2 - 45.0 - 10.0 < page_limit[page]) {
            y2 = page_limit[page];
            page += 1;
            page_limit[page] = page_limit[page - 1] - 600;
                                    }
                        }

      if(last_num_of_childn > 3){
        if(y2 - 69.0 - 10.0 < page_limit[page]) {
            y2 = page_limit[page];
            page += 1;
            page_limit[page] = page_limit[page - 1] - 600;
                                    }
                        }
      fprintf(ptr_final_data, "%d %d %f %f %f  %f 0,
            nor_more_unnor, page, x1, y1, x2, y2 );
            y1 = y2; x1 = x2;
                                }
            if (childn < 4){
            deltay = 36.0/childn;
              x2 = 90.0  - deltax;
              y2 = y1 - 78.0  - deltay;
                        }

          if (childn > 3){
            deltay = 60.0/childn;
              x2 = 90.0  - deltax;
              y2 = y1 - 102.0  - deltay;
                        }

/*     Check the page limit here                */
       if(y2 < page_limit[page]) {
            y2 = page_limit[page] - 100.0;
            page += 1;
            page_limit[page] = page_limit[page - 1] - 600;
                                    }
            nor_more_unnor = 0; last_num_of_childn = childn;

          for (i = 1; i <= childn; i += 1) {
          k1 += 1;
          x2 += deltax;
          y2 += deltay;
          if( j > 1  && j <= temp_childn && i == 1 ) { temp = y2; y2 += deltay;
                                    }
          if( j > 1  && j <= temp_childn && k1 == j ) y2 = temp;
          current_node += 1;
          num_of_nodes += 1; node_index[num_of_nodes] = current_node;
          xlocation[num_of_nodes] = x2; ylocation[num_of_nodes] = y2;
          fprintf(ptr_final_data, "%d %d %f %f %f  %f 0,
                nor_more_unnor, page, x1, y1, x2, y2 );
                                    }
                                }
          if ( j > temp_childn ){
```

```
                printf("j=%d childn=%d No any child is a parent
                    for parent [%d] 0, j, childn, Parent);
                        }

    if( (j > temp_childn && (num_of_parents - is_left_any_in_P_c)
            != 0) || (j <= (5 * k2) && temp_childn > 5) ) {
            nor_more_unnor = 2;
                                                    }
    fprintf(ptr_max_page, "%d", page);
    fclose(ptr_max_page);
    fclose(Is_parent);
    fclose(ptr_final_data);
    fclose(num_of_childn);
}


first_child(s)
char s[100];
{
      int i;
      i = 0;
      while ( s[i] != ' ' )  ++i;
      s[i] = '0';
}


atoi(s)
char s[100];
{
  int i, n;
  n = 0;
  for (i = 0; s[i] >= '0' && s[i] <= '9'; ++i)
  n = 10 * n + s[i] - '0';
  return(n);
}

itoa(n, s)
char s[100];
int  n;
{
   char s1[];
  int i, sign;
 copy(s1,s);
  if (( sign = n) < 0) n = -n;
  i = 0;
  do {
     s[i++] = n % 10 + '0';
     }
  while ((n /= 10) > 0);
  if (sign < 0)
        s[i++] = '-';
        s[i] = '';
        reverse(s);
}


reverse(s)
char s[100];
{
    int c, i, j;
    for (i = 0, j = strlen(s) - 1; i < j; i++, j--) {
        c = s[i];
        s[i] = s[j];
        s[j] = c;
    }
}

copy(s1,s2)
char s1[] , s2[];
{
   int i;
   i = 0;
   while(( s2[i] = s1[i]) != ' ')
            ++i;
}
```

```
* Put the titles in the right place
* Cal_move_num_of_lines_to_insert_title.c                    *


#include <stdio.h>

main()

{

    int old_Parent, Parent, childn, num_of_lines;
    FILE *fopen(), *ptr_num_of_childn, *ptr_num_of_lines, *ptr_num;

    ptr_num_of_childn = fopen("Parent_child", "r");
    ptr_num_of_lines = fopen("move_num_of_lines_to_insert_title", "r");
    ptr_num = fopen("where_title_to_be_inserted_file", "w");
    num_of_lines = 0;
    fscanf(ptr_num_of_lines, "%d", &old_Parent);
    while( (fscanf(ptr_num_of_childn, "%d %d", &Parent, &childn) != EOF)) {
       num_of_lines += childn;
       if(old_Parent == Parent) break;
       }
    fprintf(ptr_num, "%d", num_of_lines);
    fclose(ptr_num_of_childn);
    fclose(ptr_num_of_lines);
    fclose(ptr_num);

}


* Cal_node_name.c
* THIS PROGRAM GIVES A UNIQUE IDENTIFIER TO A NODE


#include <stdio.h>
#define   FIRST_CHILD      1

main()

{
    int n_p, Parent, childn;
    char edgetype[20], G[20], Generate[20], R[20], Respond[20], Supports[20], other[20];
    FILE *fopen(), *node_parent, *node_Parent_child, *num_of_childn, *edge;

    node_parent = fopen("temp_parent", "r");
    edge = fopen("edge_type", "r");
    node_Parent_child = fopen("Parent_child", "r");
    num_of_childn = fopen("temp_child", "w");

    Parent = -1;

    fscanf(edge, "%s %s %s %s %s %s", G, Generate, R, Respond, Supports, other);
    fscanf(node_parent, "%d %s", &n_p , edgetype);

    while( (fscanf(node_Parent_child, "%d%d", &Parent, &childn) != EOF)) {
        if(n_p == Parent ) {
   childn += 1;
   if( equal(edgetype,Generate) == 1 || equal(edgetype,G) == 1 ) {
fprintf(num_of_childn, "I[%d%d] %d %d %d %d 0, Parent, childn, Parent, childn - 1, Parent, childn );
fprintf(num_of_childn, " %d%d %d%d I[%d%d] 0, Parent, childn - 1, Parent, childn, Parent, childn ); }

   if( equal(edgetype,R) == 1 || equal(edgetype,Respond) == 1 ) {
fprintf(num_of_childn, "P[%d%d] %d %d %d %d 0, Parent, childn, Parent, childn - 1, Parent, childn );
fprintf(num_of_childn, " %d%d %d%d P[%d%d] 0, Parent, childn - 1, Parent, childn, Parent, childn ); }

   if( equal(edgetype,other) == 1 || equal(edgetype,Supports) == 1 ){
fprintf(num_of_childn, "A[%d%d] %d %d %d %d 0, Parent, childn, Parent, childn - 1, Parent, childn );
fprintf(num_of_childn, " %d%d %d%d A[%d%d] 0, Parent, childn - 1, Parent, childn, Parent, childn ); }
        break;
                    }
                                                }
     if(n_p != Parent) {

   if( equal(edgetype,Generate) == 1 || equal(edgetype,G) == 1 ){
fprintf(num_of_childn,"I[%d%d]                %d                %d                %d                %d
```

```
0,n_p,FIRST_CHILD,n_p,FIRST_CHILD,n_p,FIRST_CHILD );
 fprintf(num_of_childn," %d %d%d I[%d%d] 0,n_p,n_p,FIRST_CHILD,n_p,FIRST_CHILD );
                                                }

    if( equal(edgetype,R) == 1 || equal(edgetype,Respond) == 1 ) {
 fprintf(num_of_childn,"P[%d%d]                %d              %d              %d              %d
0,n_p,FIRST_CHILD,n_p,FIRST_CHILD,n_p,FIRST_CHILD );
 fprintf(num_of_childn," %d %d%d P[%d%d] 0,n_p,n_p,FIRST_CHILD,n_p,FIRST_CHILD );
                                                }

    if( equal(edgetype,other) == 1 || equal(edgetype,Supports) == 1 ) {
 fprintf(num_of_childn,"A[%d%d]                %d              %d              %d              %d
0,n_p,FIRST_CHILD,n_p,FIRST_CHILD,n_p,FIRST_CHILD );
 fprintf(num_of_childn," %d %d%d A[%d%d] 0,n_p,n_p,FIRST_CHILD,n_p,FIRST_CHILD );
                                                }
                    }

    fclose(node_parent);
    fclose(node_Parent_child);
    fclose(num_of_childn);
}

equal(s1, s2)
char s1[], s2[];
{
      int i;
      i = 0;
      while ( s1[i] == s2[i] ) ++i;
      if( strlen(s1) < i ) return(1);
      else
         return(0);
}
```

Interface Function

```
;; w38.c

#include <stdio.h>
#include <starbase.c.h>
#include <sys/types.h>
#define EndOfLine   FALSE
#define   MAXLINE        1000
#define   MAX            40

float sb_xmax,sb_ymax;
float sb_xmin;

int sb_num_pens;

char *getenv();

main(argc,argv)
int argc; char *argv[];
{
        int fildes;
        int j,q;
        int n=2,loops=2;
        float xm, ym;
        float x,y,inc=40.0;
        float ri,imax;
        float pts[20];
        float pad=0.0;
        float *ptr;
        int flags;
        float p[2][3],p1[3],p2[3],resolution[3];

    float x1, y1, x2, y2;
    char        line[MAXLINE], nname[MAX], edge[MAX];
    FILE *fopen(), *ptr_to_location, *ptr_to_title,
        *ptr_Final_data, *ptr_final_data, *ptr_old_page_f,
        *ptr_current_page_f, *ptr_to_Title, *ptr_max_page;
    int  p, nor_more_unnor, num_of_far_parents, page,
        current_page, old_page, max_page;

/*   The following part just to determine the current page   */
```

```
    ptr_old_page_f = fopen("old_page_f", "r");
    ptr_max_page = fopen("max_page", "r");
     fscanf(ptr_old_page_f, "%d", &old_page);
     fscanf(ptr_max_page , "%d", &max_page);
    if(old_page <  max_page) current_page = old_page + 1;
      else
          current_page = max_page;
    fclose(ptr_old_page_f);
    ptr_old_page_f = fopen("old_page_f", "w");
     fprintf(ptr_old_page_f, "%d",current_page);
    fclose(ptr_old_page_f);
    fclose(ptr_max_page);

/*    The following part to find out the data for the current page   */

    ptr_to_title = fopen("title_file", "r");
    ptr_to_Title = fopen("Title_file", "w");
    ptr_Final_data = fopen("Final_data", "r");
    ptr_final_data = fopen("final_data_w", "w");
    while( (fscanf(ptr_Final_data,"%d%d%f%f%f%f%f",
          &nor_more_unnor, &page, &x1, &y1, &x2, &y2) != EOF)) {
    if(nor_more_unnor == 0) fgets(line, MAXLINE, ptr_to_title);
          if(current_page == page){
               if(current_page >= 2) {
          y1 += (600.0 * (current_page - 1));
          y2 += (600.0 * (current_page - 1));}
    fprintf(ptr_final_data, "%d %d %f %f %f  %f 0,
           nor_more_unnor, page, x1, y1, x2, y2 );
    if(nor_more_unnor == 0) fprintf(ptr_to_Title, "%s 0, line);

             }
          }
    fclose(ptr_Final_data);
    fclose(ptr_final_data);
    fclose(ptr_to_title);
    fclose(ptr_to_Title);

/* The following part is used to draw the graph
  which represents the reasoned discourse */

   if ((fildes = gopen(getenv("SB_OUTDEV"),
      OUTDEV, getenv("SB_OUTDRIVER"),INIT)) == -1)
      {
               printf ("error during gopen()0);
               exit (1);
      }
     num_of_far_parents = 0.0;
      inquire_sizes(fildes,pl,resolution,p1,p2,&sb_num_pens);
      sb_xmin = pl[0][0];
      sb_xmax = pl[1][0];
      if (pl[0][1] > pl[1][1])
               sb_ymax = pl[0][1];
      else     sb_ymax = pl[1][1];
      /* fildes = sbopen(argc,argv,OUTDEV,INIT); */
      xm=sb_xmax;
      ym=sb_ymax;
      imax = (xm > ym ? ym / 2 : xm / 2);

      /* SET UP VIEWING TO FULL DEVICE LIMITS */
      vdc_extent(fildes,0.0,0.0,0.0,xm/ym,1.0,1.0);
      view_window(fildes,0.0,0.0,xm,ym);

      /* TRY MOVING CURSORS (IN VDC'S) */
      echo_type(fildes,1,1,0.0,0.0,0.0);
      for (ri=0.0;  ri <= 1.0;  ri += 0.01 )
                        echo_update(fildes,1,ri,ri,ri);
      echo_type(fildes,1,0,0.0,0.0,0.0);

      background_color_index(fildes,2 % sb_num_pens);
      interior_style(fildes,INT_SOLID,TRUE);

      clear_view_surface(fildes);
      character_height(fildes,12.0);
      character_width(fildes,6.0);
     text_alignment(fildes, TA_CENTER, TA_CAP, 0.0 , 0.0);
```

```
        q = 0; ri=0.0;

        /* TRY TEXT AND RECTANGLES */
                rectangle(fildes,0.0, 0.0, 600.0, 600.0);
                text_color_index(fildes,(q++) % sb_num_pens);

     if(current_page == 1)
    text2d(fildes, 300.0 , 595.0 , "PESIS", WORLD_COORDINATE_TEXT, EndOfLine);

      ptr_to_location = fopen("final_data_w", "r");
    ptr_to_Title = fopen("Title_file", "r");

        line_color_index(fildes,(int)ri % sb_num_pens);

    while ( fscanf(ptr_to_location,"%d %d %f %f %f %f",
          &nor_more_unnor, &p, &x, &y, &x1, &y1) !=EOF) {
        if( nor_more_unnor == 1){
        move2d(fildes, x , y );
    draw2d(fildes, 12.0, y);
        draw2d(fildes, 12.0, y1);
        draw2d(fildes, x1, y1 );
                        }
    if( nor_more_unnor == 2){
    if(num_of_far_parents == 15)  num_of_far_parents = 0.0;
    num_of_far_parents +=1;
        move2d(fildes, x , y - 3.0);
    draw2d(fildes, 600.0 - (num_of_far_parents * 2.0), y - 3.0);
        draw2d(fildes,  600.0 - (num_of_far_parents * 2.0), y1);
        draw2d(fildes, x1, y1 );
                        }
    if( nor_more_unnor == 0){
    fscanf(ptr_to_Title, "%s %s", nname, edge);
    fgets(line, MAXLINE, ptr_to_Title);
        move2d(fildes, x , y );
    draw2d(fildes, x1, y);
        draw2d(fildes, x1, y1 + 39.0);
    text2d(fildes, x1 , y1+ 12.0 , line, WORLD_COORDINATE_TEXT, EndOfLine);
    text2d(fildes, x1 , y1+ 24.0 , edge, WORLD_COORDINATE_TEXT, EndOfLine);
    text2d(fildes, x1 , y1+ 36.0 , nname, WORLD_COORDINATE_TEXT, EndOfLine); }
                        }
        gawk (fildes,3000);
    fclose(ptr_to_location);
    fclose(ptr_to_Title);
        gclose(fildes);
}

gawk(fildes,sec)
int fildes,sec;
{
        make_picture_current(fildes);
        sleep(sec);
}

fgetss(s, n, iop)
 char *s;
int n;
register FILE *iop;
{

  register int c;
  register char *cs;

  cs = s;
  while (--n > 0 && (c = getc(iop)) != EOF)
    if (( *cs++ = c) == '0')
            break;
  *cs = ' ';
  return((c == EOF && cs == s) ? NULL : s);
}



fputs(s, iop)
register char *s;
register FILE *iop;
```

```
{
  register int c;

  while (c = *s++)
      putc(c, iop);
}
```

Browsing Function

```
## This program has the menu which controls most of the HERD features

WarpCursor
BorderWidth     2
TitleFont       "9x15"
MenuFont        "8x13"
IconFont        "8x13"

Color
{
    BorderColor "red"

    BorderTileForeground "yellow"
    BorderTileBackground "yellow"

    TitleForeground "black"
    TitleBackground "PaleGreen"

    MenuForeground "white"
    MenuBackground "CadetBlue"

    MenuTitleForeground "white"
    MenuTitleBackground "CornFlowerBlue"
    MenuShadowColor "black"

    IconForeground "white"
    IconBackground "cadetblue"
    IconBorderColor "green"
    IconManagerForeground "black"
    IconManagerBackground "thistle"
}

Monochrome
{
    BorderColor "black"
    BorderTileForeground "black"
    BorderTileBackground "white"
    TitleForeground "black"
    TitleBackground "white"
}

ResizeFont      "fixed"
NoTitleFocus
Zoom # 20
RandomPlacement

IconifyByUnMapping
DontIconifyByUnmapping
{
    "Xmh"
}

ShowIconManager
IconManagerGeometry     "=150x10+742+10"
IconManagerFont                 "*new century schoolbook-medium-r-normal--12*"

UnknownIcon     "woman"
IconDirectory   "/cs/crosby/mdb/usr/include/X11/bitmaps"
ForceIcons
Icons
{
    "xfig"          "xfig.icon"
    "xterm"         "xterm.icon"
}
```

```
NoTitle
{
  "TWM"
  "xpostit"
  "topBox"
  "xclock"
  "xbiff"
  "xload"
  "xckmail"
  "Digital Clock"
  "Analog Clock"
}

NoHighlight
{
  "xclock"
  "xckmail"
  "dclock"
}

AutoRaise
{
  "xterm"
}

DefaultFunction f.menu "default-menu"
#WindowFunction f.function "blob"

#Button = KEYS : CONTEXT : FUNCTION
#-------------------------------
Button1 =      : root   : f.menu "button1"
Button3 =      : root   : f.menu "button3"
Button2 =      : root   : f.menu "button2"
Button2 =      : iconmgr   : f.function "de-raise-n-focus"

Button1 =      : title  : f.raiselower
Button2 =      : title  : f.lower
Button3 =      : title  : f.move

Button1 =      : frame  : f.raise
Button2 =      : frame  : f.lower
Button3 =      : frame  : f.move

Button1 =      : icon   : f.iconify
Button2 =      : icon   : f.raiselower
Button3 =      : icon   : f.move
Button1 = m    : icon   : f.raise
Button2 = m    : icon   : f.lower
Button2 = c    : root   : f.function "beep-beep"
Button3 = m    : icon   : f.move

"Break" =      : window : f.iconify
"Break" =      : icon   : f.iconify

"KP_F1"          =      : "Xmh"    : f.iconify
"KP_F2"          =      : "hpfcra" : f.iconify
"KP_F3"          =      : "hpfcdq" : f.iconify

"F1"    =      : window : f.raise
"F1"    =      : title  : f.raise

"F2"    =      : window : f.lower
"F2"    =      : window : f.lower

"F7"    =      : window : f.function "Print-Screen"
"F7"    =      : icon   : f.function "Print-Screen"
"F7"    =      : root   : f.function "Print-Screen"
"F7"    =      : title  : f.function "Print-Screen"

"F8"    =      : window : f.function "Print-Window"
"F8"    =      : title  : f.function "Print-Window"

"Clear" =      : window : f.refresh
"Clear" =      : icon   : f.refresh
"Clear" =      : title  : f.refresh
```

```
"Clear" =       : root   : f.refresh

"ClearLine" =       : window : f.winrefresh
"ClearLine" =       : title  : f.winrefresh

"Cancel" = sc : window : f.quit
"Cancel" = sc : root   : f.quit
"Cancel" = sc : icon   : f.quit
"Cancel" = sc : title  : f.quit

Function "beep-beep"
{
   f.beep
   f.beep
   f.beep
   f.beep
   f.beep
}

Function "de-raise-n-focus"
{
   f.deiconify
   f.raise
   f.focus
}

Function "raise-n-focus"
{
   f.raise
   f.focus
}

Function "blob"
{
   f.deiconify
   f.raise
   f.focus
   f.lower
}


menu "button1"
{
"< < Useful Window Ops > >"      f.title
"  Print Window"                 f.function "Print-Window"
"  Print Screen"          f.function "Print-Screen"
"  New Xterm Window"             !"xterm -fn 8x13 -fb 8x13bold -n xterm-aroni -sk -sb -sl  &"
"  New HPterm Window"            !"hpterm -n hpterm-aroni -T hpterm-aroni -sb &"
"  New Emacs Window"             !"emacs -fn 8x13 -name Emacaroni -title Emacaroni newfile_aroni &"
"  Calendar"                     !"xterm -e month -A &"
"< < Calling on Friends > >"     f.title
"  Call Hp1"             !"xterm -e telnet hp1 &"
"  Call Chad1"           !"xterm -e telnet chad1 &"
"  Call Chad2"           !"xterm -e telnet chad2 &"
"  Call Crosby"          !"xterm -e telnet crosby &"
"  Call Dingle"          !"xterm -e telnet dingle &"
"  Call Everton"         !"xterm -e telnet everton &"
"  Call Formby"                  !"xterm -e telnet formby &"
"  Call Orion"           !"xterm -e telnet orion &"
"< < Other Useful Junk > >"      f.title
"  Run C shell"                  !"xterm -e csh &"
"  Run Bourne shell"     !"xterm -e sh &"

}

menu "button2"
{
"Window Ops"           f.title
"Show Icon Mgr"        f.showiconmgr
"Hide Icon Mgr"        f.hideiconmgr
"Refresh"              f.refresh
"Refresh Window"       f.winrefresh
"twm Version"          f.version
"Focus on Root"        f.unfocus
"Source .twmrc"        f.twmrc
```

```
"Cut File"                  f.cutfile
"(De)Iconify"               f.iconify
"DeIconify"                 f.deiconify
"Move Window"               f.move
"ForceMove Window"          f.forcemove
"Resize Window"             f.resize
"Raise Window"              f.raise
"Lower Window"              f.lower
"Raise or Lower" f.raiselower
"Focus on Window"           f.focus
"Raise-n-Focus"             f.function "raise-n-focus"
"Kill twm"                  f.quit
}

menu "button3"
{
"PESIS: A Hypertext for all Reasoned Discourse"      f.title
" Proto-Node "              f.function "proto-node"
" Create-Node"              f.function "create-node"
" Link-Node"                f.function "link-node"
" Display ==========================================>>>>"          f.menu "display"
" Browsing ==========================================>>>>"          f.menu "browsing"
" Search ==========================================>>>>"          f.menu "search"
" Print ==========================================>>>>"          f.menu "print"
" font windows"             f.menu "pulley"
" Destroy Window"           f.destroy
" Stop Xwindows"            f.quit
}

menu "display"
{
"   DISPLAY      "          f.title
" Content        "          f.function   "disp-content"
" Information ===>>"        f.menu       "disp-Author-or-Title"
" Graph          "          f.function   "disp-graph"
}

menu "disp-Author-or-Title"
{
" Author-Date-or-Title-Inf " f.title
"  Author-Date-Information  " f.function   "Display-Author-Date"
"  Title-Information        " f.function   "Display-Title"
}

menu "browsing"
{
" BROWSING   "              f.title
"  Page      "              f.function     "brows-graph"
}

menu "search"
{
" SEARCH  " f.title
"  Text   " f.function "search-text"
"  Information   " f.function "search-information"
}

menu "print"
{
" PRINT  " f.title
"  Report   " f.function "print-report"
"  Sub-report ===" f.menu "sub-report"
}

menu "sub-report"
{
" SUB-REPORT      "         f.title
"  Content        "         f.function   "sub-report-content"
" Information ===>>"        f.menu       "sub-report-information"
"  Graph          "         f.function   "sub-report-graph"
}

menu "sub-report-information"

{
```

```
" Sub-Report-Author-or-Title"   f.title
"  Print-Author-Date-Inf      "  f.function     "Print-Author"
"  Print-Title-Inf           "  f.function     "Print-Title"
}


menu "pulley"
{
"emacs window"          f.menu "efonts"
"xterm window"          f.menu "xfonts"
}


menu "efonts"
{
"small font"            !"emacs -name E-macaroni -fn 6x13 -T E-macaron new-file-aroni &"
"medium font"           !"emacs -name E-macaroni -T E-macaroni -fn 8x13 new-file-aroni &"
"large font"            !"emacs -fn 9x15 -name E-macaroni -T E-macaroni new-file-aroni &"
"bold font"             !"emacs -fn 8x13bold -name E-macaroni -T E-macaroni new-file-aroni &"
}

menu "xfonts"
{
"small font"            !"xterm -fn 6x13 -name eye-strain -T eye-strain -sk -sb -sl &"
"medium font"           !"xterm -fn 8x13 -name Xterm-aroni -T Xterm-aroni -sk -sb -sl &"
"large font"            !"xterm -fn 9x15 -name FATSO -T FATSO -sk -sb -sl &"
"bold font"             !"xterm -fn 8x13bold -name BRAZEN -T BRAZEN -sk -sb -sl &"
}


menu "pull-right"
{
"Window Ops"            f.title
"Refresh"               f.refresh
"Refresh Window"        f.winrefresh
"twm Version"           f.version
"Focus on Root"         f.unfocus
"Source .twmrc"         f.twmrc
"Cut File"              f.cutfile
"(De)Iconify"           f.iconify
"Move Window"           f.move
"Resize Window"         f.resize
"Raise Window"          f.raise
"Lower Window"          f.lower
"Focus on Window"       f.focus
"Destroy Window"        f.destroy
"Kill twm"              f.quit
}

menu "default-menu"
{
"Default Menu"          f.title
"Refresh"               f.refresh
"Refresh Window"        f.winrefresh
"twm Version"           f.version
"Focus on Root"         f.unfocus
"Source .twmrc"         f.twmrc
"Cut File"              f.cutfile
"(De)Iconify"           f.iconify
"DeIconify"             f.deiconify
"Move Window"           f.move
"ForceMove Window"      f.forcemove
"Resize Window"         f.resize
"Raise Window"          f.raise
"Lower Window"          f.lower
"Focus on Window"       f.focus
"Raise-n-Focus"         f.function "raise-n-focus"
"Destroy Window"        f.destroy
"Zoom Window"           f.zoom
"FullZoom Window"       f.fullzoom
"Kill twm"              f.quit
}

Function "Print-Window"
{
```

```
    !"xwd | xpr -device ljet -density 100 -rv |lpr -ddvijet & "
}

Function "Print-Screen"
{
    !"xwd -root | xpr -device ljet -density 100 -rv |lpr -ddvijet & "
}

Function "hpfcdq"
{
    !"xterm =80x48+150+50 -bg palegreen #+964+4 -e rlogin hpfcdq &"
}

Function "hpfcra"
{
    !"xterm =80x48+200+100 -bg lightblue #+903+4 -e rlogin hpfcra &"
}

Function "hpfccb"
{
    !"xterm =80x48+200+100 -bg palegreen #+903+4 -e rlogin hpfccb &"
}

Function "hpfcat"
{
    !"xterm =80x48+200+100 -bg thistle #+903+4 -e rlogin hpfcat &"
}

Function "disp-content"
{
     !"emacs text  -geometry 69x44+0+190 &"
}

Function "Display-Author-Date"
{
    !"emacs Author_Date_Inf  -geometry 69x41+0+224 &"
}

Function "Display-Title"
{
      !"emacs Title_Inf  -geometry 69x39+0+258 &"
}

Function "sub-report-content"
{
     !"lp -djet text &"
}

Function "Print-Author"
{
     !"lp -djet Author_Date_Inf &"
}

Function "Print-Title"
{
     !"lp -djet Title_Inf &"
}

Function "link-node"
{
     !"emacs -l cn16 -kill &"
}

Function "create-node"
{
     !"emacs fill_form -geometry 69x44+0+190 &"
}

Function "disp-graph"
{
     !"rw_graph &"
}

Function "brows-graph"
{
```

```
    !"rw_scroll &"
}

Function "proto-node"
{
   !"xterm -n Proto_Node -geometry 55x9+419+5 &"
}
```

Appendix 4
MUCH Algorithm

Annotation Function

```
;; Show Annotation Outline and Read Annotations


(defun show-annot-or-read-annot ()
 "select the current node to show annotations"
 (interactive)
 (setq book-browser-buffer-flag t)
 (setq outline-buffer-flag t)
 (if (string-equal (buffer-name) "*Frames Window*")
     (setq book-browser-buffer-flag nil))
 (if (string-equal (buffer-name) current-outline-buffer)
     (setq outline-buffer-flag nil))

(if (and book-browser-buffer-flag outline-buffer-flag)
   (disp-message)
   (setq buffer-read-only nil)
   (beginning-of-line)
   (if book-browser-buffer-flag (do-nothing)
   (forward-word 2) (backward-word 1)
   (setq start (point))
   (forward-word 1) (setq selected-node-name
   (buffer-substring start (point)))

   (find-window "*Annotation Text*")
   (find-file (concat user_name "/annot-outline-history"))
   (setq last-outline (buffer-string))
   (kill-line) (insert selected-node-name)
   (goto-char (point-min)) (save-buffer)
   (setq current-outline-buffer selected-node-name)
   (find-file (concat user_name "/emacs-ing"))
   (setq buffer-read-only nil)
   (kill-region (point-min) (point-max))
   (insert "show_annotations") (newline)
   (insert selected-node-name) (newline)
   (save-buffer)
   (switch-to-buffer "*Annotation Text*")
   (find-file (concat user_name "/ing-emacs1"))
   (setq buffer-read-only nil) (kill-region (point-min) (point-max))
   (save-buffer)

   (find-file "~/ing-emacs1")
   (insert-file "~/ing-emacs")
   (goto-char (point-max))
   (save-buffer)
   (while (bobp)
        (message "%s" "Waiting for output from database ....." )
        (sleep-for 1)
        (insert-file "~/ing-emacs") (goto-char (point-max))
   (save-buffer)
   )

   (switch-to-buffer "*Annotation Text*")
   (find-window last-outline )
   (find-file (concat user_name "/" selected-node-name))
   (setq buffer-read-only nil)
   (insert-file "~/ing-emacs1") (save-buffer)
   (setq buffer-read-only t)

;; The following shell command is used to make sure "ing-emacs" file is empty
;; and not to get every time the question which asked for old/new version.

   (shell-command "cp ~/t ~/ing-emacs")
)
```

```
(if outline-buffer-flag (do-nothing)
    (setq frame-n (buffer-name))
    (forward-word 1) (backward-word 1)
    (setq a (point)) (forward-word 1)    ;(end-of-line)
    (setq s-annotation  (buffer-substring a (point)))
    (forward-word 1) (backward-word 1)
    (setq a (point)) (forward-word 1)    ;(end-of-line)
    (setq link  (buffer-substring a (point)))
    (forward-word 1) (backward-word 1)
    (setq a (point)) (forward-word 1)    ;(end-of-line)
    (setq d-annotation  (buffer-substring a (point)))
    (find-window "*Annotation Text*")
    (find-file (concat user_name "/emacs-ing" ))
    (setq buffer-read-only nil) (kill-region (point-min) (point-max))
    (insert "read-annotation") (newline)
    (insert frame-n ) (newline)
    (insert s-annotation ) (newline)
    (save-buffer)
    (find-file (concat user_name "/ing-emacs1"))
    (setq buffer-read-only nil) (kill-region (point-min) (point-max))
    (save-buffer)

   (find-file s-annotation)
   (kill-region (point-min) (point-max))
   (insert-file "~/ing-emacs")
   (goto-char (point-max))
   (save-buffer)
   (while (bobp)
         (message "%s" "Waiting for output from database ....." )
         (sleep-for 1)
         (insert-file "~/ing-emacs") (goto-char (point-max))
   (save-buffer)
   )
   (copy-to-buffer "*Annotation Text*" (point-min) (point-max))
   (switch-to-buffer "*Annotation Text*")
   (shell-command "cp ~/t ~/ing-emacs")
  )
 )
)


(defun disp-message ()
    (message   "Wrong cursor position, try again ..." )
    (sleep-for 5)
)

;; Show Annotations ordered by author

(defun show-annot-by-author ()
 "find annotations were created by a specific author"
 (interactive)
 (setq author (read-from-minibuffer "Enter author name (user-login-name): "))
 (setq frame (read-from-minibuffer "For a specific frame enter frame name. Otherwise press return: "))
 (find-window "*Paragraph Text*")   (other-window 1)
 (find-file (concat user_name "/emacs-ing"))
 (setq buffer-read-only nil)
 (kill-region (point-min) (point-max))
 (insert "show-annotations-by-author") (newline)
 (insert author ) (newline)
 (if (string-equal frame "") (do-nothing)
 (insert  frame ) (newline) (save-buffer)
 )
   (find-file (concat user_name "/ing-emacs1"))
   (setq buffer-read-only nil) (kill-region (point-min) (point-max))
   (save-buffer)

   (find-file "~/ing-emacs1")
   (kill-region (point-min) (point-max))
   (insert-file "~/ing-emacs")
   (goto-char (point-max))
   (save-buffer)
   (while (bobp)
         (message "%s" "Waiting for output from database ....." )
         (sleep-for 1)
```

```
        (insert-file "~/ing-emacs") (goto-char (point-max))
   (save-buffer)
   )

 (goto-char (point-min))
 (insert "These are the annotations were created by the author "author)
 (newline)  (goto-char (point-min)) (save-buffer)
 (copy-to-buffer "*Annotation Text*" (point-min) (point-max))
 (switch-to-buffer "*Annotation Text*")
 (shell-command "cp ~/t ~/ing-emacs")
)


;; Show Annotations ordered by date

(defun show-annot-by-date ()
 "find annotations were created at, after, or before a pecific date"
 (interactive)
 (setq b-o-a (read-from-minibuffer "Enter (b for before) (o for on) (a for after) date: "))
 (setq dates (read-from-minibuffer "Enter date on format (day/mon/year) : "))
 (setq frame (read-from-minibuffer "For a specific frame enter frame name. Otherwise press return: "))
 (find-window "*Annotation Text*")
 (find-file (concat user_name "/emacs-ing"))
 (setq buffer-read-only nil)
 (kill-region (point-min) (point-max))
 (insert "show-annotations-by-date") (newline)
 (if (string-equal b-o-a "b") (insert "before"))
 (if (string-equal b-o-a "o") (insert "on"))
 (if (string-equal b-o-a "a") (insert "after"))  (newline)
 (insert dates) (newline)
 (if (string-equal frame "") (do-nothing)
 (insert frame ) (newline)
 )
 (save-buffer)
 (find-file (concat user_name "/ing-emacs1"))
 (setq buffer-read-only nil) (kill-region (point-min) (point-max))
 (save-buffer)
 (find-file "~/ing-emacs1")
 (kill-region (point-min) (point-max))
 (insert-file "~/ing-emacs")
 (goto-char (point-max))
 (save-buffer)
 (while (bobp)
       (message "%s" "Waiting for output from database ....." )
       (sleep-for 1)
       (insert-file "~/ing-emacs") (goto-char (point-max))
       (save-buffer)
 )

 (goto-char (point-min))
 (insert "These are the annotations were created ")
 (if (string-equal b-o-a "b") (insert "before"))
 (if (string-equal b-o-a "o") (insert "on"))
 (if (string-equal b-o-a "a") (insert "after"))
 (insert  " " dates) (newline)
 (goto-char (point-min)) (save-buffer)
 (copy-to-buffer "*Annotation Text*" (point-min) (point-max))
 (switch-to-buffer "*Annotation Text*")
 (shell-command "cp ~/t ~/ing-emacs")
)

;; Store Annotations in Database

(defun store-in-database ()
  (interactive)
  (find-window "*Annotation Text*")
  (goto-char (point-min))
  (while (search-forward "  :  " (point-max) t)
        (setq a (point))
        (beginning-of-line)
        (kill-region (point) a)
  )
 (setq temp (buffer-substring (point-min) (point-max)))
 (find-file "~/emacs-ing")
 (kill-region (point-min) (point-max)) (insert temp) (save-buffer)
```

```
(kill-buffer (current-buffer))
(kill-region (point-min) (point-max))
)

;; Creating Annotation Function

(defun create-annotation ()
 "select the current node for annotation"
 (interactive)
 ;; book-browserbuffer-flag   represents the *Book_Browser*
 ;; outline-buffer-flag       represents the *Outline* window
 ;; paragraph-buffer-flag     represents the *Paragraph_Text* window
 (setq book-browser-buffer-flag t) (setq outline-buffer-flag t)
 (setq paragraph-buffer-flag t)
 (if (string-equal (buffer-name) "*Frames Window*")
 (setq book-browser-buffer-flag nil))
 (if (string-equal (buffer-name) current-outline-buffer)
 (setq outline-buffer-flag nil))
 ;;(if (string-equal (buffer-name) "*Paragraph Text*")
 ;;(setq paragraph-buffer-flag nil))
 (setq buffer-read-only nil)
 (beginning-of-line)
 (if book-browser-buffer-flag (do-nothing)          (forward-word 2) (backward-word 1)
     (setq start (point))          (forward-word 1)
     (setq selected-node-name (buffer-substring start (point)))          (setq annotation-outline selected-
node-name)          )
 (if outline-buffer-flag (do-nothing)          (forward-word 1)          (setq a (point)) (backward-word 1)
       (setq selected-node-name (buffer-substring (point) a))          (setq annotation-outline (buffer-
name))          )
 (setq buffer-read-only t)
 (create-window-for-annotation "test")
)

(defun do-nothing ()
)

(defun create-window-for-annotation (string)
 ;; "Search forward in *Paragraph_Text* for a string"
 (interactive "sYou want to create an annotation: yes")
 (find-window "*Annotation Text*")
 (setq link (read-from-minibuffer "Enter link-type: "))
 (setq last-number (concat "~/last_number " annotation-outline))
 (find-file (concat user_name "/" last-number)) (goto-char (point-max))
 (if (bobp) (insert "-1")) (goto-char (point-min)) (save-buffer)
 (setq last-annotation-name-plus-1 "")
 (setq last-annotation-name-plus-1 (buffer-string))
 (setq last-annotation-name-plus-1 (string-to-int last-annotation-name-plus-1))
 (setq last-annotation-name-plus-1 (1+ last-annotation-name-plus-1))
 (setq last-annotation-name-plus-1 (int-to-string last-annotation-name-plus-1))
 (goto-char (point-min)) (kill-line)
 (insert last-annotation-name-plus-1)
 (goto-char (point-min)) (save-buffer)
 (setq last-annotation-name-plus-1 (concat "annot" last-annotation-name-plus-1))
 (find-file (concat user_name "/" annotation-outline))
 (setq buffer-read-only nil)
 (goto-char (point-max))
 (insert "  " last-annotation-name-plus-1 "-" link "-" selected-node-name)
 (end-of-line) (newline) (save-buffer)
 (switch-to-buffer "*Annotation Text*")
 (kill-region (point-min) (point-max))
 (insert "Function Name          :  create-annotation") (newline)
 (insert "Frame Name             :  " annotation-outline) (newline)
 (insert "S. Annotation          :  " last-annotation-name-plus-1 ) (newline)
 (insert "D. Annotation          :  " selected-node-name) (newline)
 (insert "Annotation Link Name   :  " link)  (newline)
 (insert "Author Login Name      :  " (user-login-name))  (newline)
 (insert "Date Created Annot.    :  ")
 (shell-command "date '+%m/%d/%y'" t)
 (end-of-line) (newline 2);  date
)


;; Creating Link Function
```

```
(defun create-annotation-link ()
 "Create annotation link"
 (interactive)
 (setq frame-n (buffer-name))
 (find-window "*Annotation Text*")
 (kill-region (point-min) (point-max))
 (setq s-annotation (read-from-minibuffer "Enter the source-annotation: "))
 (setq d-annotation (read-from-minibuffer "Enter the destination-annotation: "))
 (setq link (read-from-minibuffer "Enter the link-type: "))
 (insert "Function Name          : create-annotation-link")
 (newline) ; Function Name
 (insert "Frame Name            : " frame-n) (newline)      ; frame name
 (insert "S. Annotation         : " s-annotation) (newline) ; s. annotation
 (insert "D. Annotation         : " d-annotation) (newline) ; d. annotation
 (insert "Annotation Link Name  : " link)
 (newline 2)    ; annotation link name
)

;; Modification Function

(defun modify-annotation ()
 "modify annotation"
 (interactive)
 (setq frame-n (buffer-name)) (beginning-of-line)
  (forward-word 1) (backward-word 1)
  (setq a (point)) (forward-word 1)
  (setq s-annotation  (buffer-substring a (point)))
  (find-window "*Annotation Text*")
  (find-file "~/emacs-ing")
  (kill-region (point-min) (point-max))
  (insert "modify-annotation") (newline) ; Function Name
  (insert frame-n ) (newline)      ; frame name
  (insert s-annotation ) (newline) ; annotation name
  (save-buffer)
  (find-file "~/ing-emacs1")
  (kill-region (point-min) (point-max))
  (insert-file "~/ing-emacs")
  (goto-char (point-max)) (save-buffer)
  (while (bobp)
      (message "%s" "Waiting for output from database ....." )
      (sleep-for 1)
      (insert-file "~/ing-emacs") (goto-char (point-max))
  (save-buffer)
  )
 (find-file "~/ing-emacs1")
 (setq temp (buffer-substring (point-min) (point-max)))
 (switch-to-buffer "*Annotation Text*")
 (kill-region (point-min) (point-max))
 (insert "Function Name          : modified-annotation")
 (newline)  ; Function Name
 (insert "Frame Name             : " frame-n )
 (newline)          ; frame name
 (insert "Annotation Name        : " s-annotation )
 (newline)      ; annotation name
 (insert temp)
 (shell-command "cp ~/t ~/ing-emacs")
)

(defun return-to-main-menu ()
 "this function is used to go back from annotation to the top level menu"
 (interactive)
 (find-window "*Paragraph Text*") (other-window 1)
 (find-file (concat user_name "/annot-outline-history"))
 (setq last-outline (buffer-string))
 (switch-to-buffer "*Note Text*")
 (find-window last-outline)
 (switch-to-buffer "*Note Outline*") (define-function-keys)
 (function-key-crib) (clear-message-line)
)
```

Appendix 5
Indexing Algorithm

Indexing

```
;;  Main program to produce a semantic net
;;----------------------------------------


;; This program should be executed from the directory  ~mhashi/IND/semantic-net/mail
(shell-command "cp -r ~rada/open/pps2 .")
(shell-command "cd pps2")
(shell-command "ls > pps2d")
(find-file "~mhashi/IND/semantic-net/mail/pps2/pps2d")
(goto-char (point-max)) (forward-line -1)
(kill-line 1) (save-buffer) (goto-char (point-min))

;; the following command will remove some of the troff commands
(load-file "~mhashi/IND/semantic-net/mail/rm-tr")

;; the following command changes each paragraph to a word-file
(load-file "~mhashi/IND/semantic-net/mail/stp0")
; you are in the pps22 dir.
(shell-command "ls > pps22d")
(find-file "~mhashi/IND/semantic-net/mail/pps22/pps22d")
(goto-char (point-max)) (forward-line -1)
(kill-line 1) (save-buffer) (goto-char (point-min))

;; the following command will add spaces at the end of each line
;;      remove "s" from the end of each word
;;       remove "ed" from the end of each word
(load-file "~mhashi/IND/semantic-net/mail/add-spaces")

;; the following command fix the side effect which results from removing
;; "s " and "ed ". In addition, some of multiple forms for a specific
;; word will be converted to a one form. For instance: documentation will be
;; changed to document, indexing to index, browsing to browse, .... etc.
(load-file "~mhashi/IND/semantic-net/mail/add-spaces1")

;; The following command finds the frequency for the unique words and sort them
;; in decending order.
(load-file "~mhashi/IND/semantic-net/mail/stp1")

;; The following command finds the term frequncy and the co-occurunce
(load-file "~mhashi/IND/semantic-net/mail/stp2")
(find-file "~mhashi/IND/semantic-net/mail/freq1")
(sort-lines nil (point-min) (point-max))
(goto-char (point-max)) (backward-word 1) (forward-line 1)
(kill-region (point) (point-max)) (save-buffer)
(load-file "~mhashi/IND/semantic-net/mail/reverse-uniq")
(shell-command "uniq -f -c < file2 > file3")
(shell-command "cp freq1 freq-temp")
(shell-command "sort -r -o freq1 file3")
(shell-command "a.out freq1 occur1 > sem-net")


;; Put the documents which are in
;; a directory into a flat file
;; ----------------------------

(defun add-spaces ()
 (setq not-eof "t")
 (while not-eof
 (end-of-line) (insert " ")
 (if (eobp) (setq not-eof nil)
    (forward-line 1)
 )
 )
 (goto-char (point-min))
```

```
)

;;      MAIN PROGRAM

(shell-command " cd ~mhashi/mail/pps2 ; ls > ht ")
(find-file "~mhashi/mail/pps2/ht") (kill-line 1)
(goto-char (point-max)) (backward-word 1) (end-of-line)
(kill-region (point) (point-max))
(goto-char (point-min))
(add-spaces) (save-buffer)

;; add spaces to make the word uniqe and remove
;; "s" and "ed" from the end of each word.
;; ------------------------------

(defun get-sub-string (a1 a2)
      (buffer-substring a1 a2)
)

(defun do-nothing ()
)

(defun add-spaces ()
   (setq not-eof "t")
   (while not-eof
   (insert " ") (end-of-line) (insert " ")
   (forward-line 1)
   (if (eobp) (setq not-eof nil))  )
   (goto-char (point-min)) (save-buffer)
)

(defun remove-s ()
     (replace-string "s " " ")
     (goto-char (point-min)) (save-buffer)
)
(defun remove-ed ()
     (replace-string "ed " " ")
     (goto-char (point-min)) (save-buffer)
)


;; MAIN PROGRAM

(find-file "~mhashi/IND/semantic-net/mail/pps22/pps22d")
(goto-char (point-min))
(if (eobp)  (setq not-eof-ht nil)
         (setq not-eof-ht "t")
)
(while  not-eof-ht
(find-file "~mhashi/IND/semantic-net/mail/pps22/pps22d")
(setq a1 (point)) (forward-word 1) (setq a2 (point))
(setq file-name-in (get-sub-string a1 a2))
(setq file-name (concat "~mhashi/IND/semantic-net/mail/pps22/" file-name-in))
(forward-line 1)
(if (eobp)  (setq not-eof-ht nil))
(find-file file-name)
(goto-char (point-min))
(add-spaces)
(remove-s) (remove-ed)
)

;; Prepare freq input file
;;------------------------------

(defun get-sub-string (a1 a2)
      (buffer-substring a1 a2)
)

(defun do-nothing ()
)

(defun conv-freq1-freq2-int ()
   (setq freq1 (string-to-int freq1))
   (setq freq2 (string-to-int freq2))
)
```

```
(defun conv-freq1-freq2-str ()
   (setq freq1 (int-to-string freq1))
   (setq freq2 (int-to-string freq2))
)
(defun write-this-line ()
   (find-file f3)
   (insert " " freq1 " " word1 " " )
   (newline 1) (save-buffer)
)

(defun find-diff ()
   (setq freq1 (-  freq1 freq2 ))
)

;; MAIN PROGRAM

(setq f1 "sorted-freq-word-file")
(setq f2 "occur3")
(setq f3 "diff")
(setq not-eof1 "t")
(find-file f1) (goto-char (point-min))
(while not-eof1
       (find-file f1)
       (forward-word 1) (backward-word 1)
       (setq a1 (point)) (forward-word 1)
       (setq freq1 (get-sub-string a1 (point)))
       (forward-word 1) (backward-word 1)
       (setq a1 (point)) (forward-word 1)
       (setq word1 (get-sub-string a1 (point)))
       (forward-line 1)
       (if (eobp) (setq not-eof1 nil))
       (setq not-eof2 "t")
        (setq freq1 (string-to-int freq1))
       (find-file f2) (goto-char (point-min))
    (while not-eof2
       (find-file f2)
       (forward-word 1) (backward-word 1)
       (setq a1 (point)) (forward-word 1)
       (setq freq2 (get-sub-string a1 (point)))
       (forward-word 1) (backward-word 1)
       (setq a1 (point)) (forward-word 1)
       (setq word2 (get-sub-string a1 (point)))
       (setq freq2 (string-to-int freq2))
       (if (equal word1 word2) (find-diff))
       (forward-line 1)
       (if (eobp) (setq not-eof2 nil))
       (setq freq2 (int-to-string freq2))
    )
       (setq freq1 (int-to-string freq1))
       (if (string-lessp "0" freq1) (write-this-line)
       )
)


;; Remove-Stop-List File
;;--------------------------

(defun get-sub-string (a1 a2)
     (buffer-substring a1 a2)
)

(defun kill-that-word ()
      (find-file file-name1)
      (beginning-of-line) (kill-line 1)
)

(defun word-length (a1 a2)
(setq word-l (1+ (- a2  a1)))
)

(defun check-2 ()
      (find-file file-name1)
)

;; MAIN PROGRAM
```

```
(find-file "1")
(goto-char (point-min))
(setq not-eof "t")
(while not-eof
      (forward-word 1) (backward-word 1)
      (setq a1 (point)) (forward-word 1) (setq a2 (point))
      (setq whole-word (get-sub-string a1 a2)
      (setq whole-word (concat " " whole-word " "))
      (find-file "suf-file") (goto-char (point-min))
      (if (search-forward whole-word (point-max) t) (kill-that-word)
                                          (setq step2 "t")
      )
      (if  step2  (check-2) )

;; Remove-suf file
;;-------------------------

(defun get-sub-string (a1 a2)
       (buffer-substring a1 a2)
)

(defun do-nothing ()
)

(defun check-spelling ()
     (setq first-part1  (get-sub-string a1 a2))
     (if (spell-region  a1 a2) (kill-region a2 a3)
         (backward-char 1)
         (if (spell-region a1 (point)) (kill-region (point) a3)
            (do-nothing)
         )
     )
)

;;  MAIN PROGRAM

(setq file-name "2")
(find-file file-name)
(goto-char (point-min))
(setq not-eof "t")
(while not-eof
     (setq a1 (point))
     (forward-word 1)(setq a3 (point))
     (backward-char 1) (setq a2 (point))
     (setq s-or-d (get-sub-string a2 a3))
     (if (equal s-or-d "s") (kill-region a2 a3)
         (if (equal s-or-d "d") (check-spelling)
              (do-nothing)
         )
     )
     (forward-line 1)
     (if (eobp) (setq not-eof nil))
)


;; Reverse Unique
;;----------------------

(defun get-sub-string (a1 a2)
       (buffer-substring a1 a2)
)

(defun do-nothing ()
)

;; MAIN PROGRAM

(setq f1 "file1")
(setq f2 "file2")
(find-file f1)
(goto-char (point-min))
(if (eobp) (setq not-eof nil)
        (setq not-eof "t")
)
```

```
(while not-eof
     (forward-word 1) (backward-word 1)
     (setq a1 (point)) (forward-word 1)
     (setq freq (get-sub-string a1 (point)))
     (forward-word 1) (backward-word 1)
     (setq a1 (point)) (end-of-line)
     (setq word (get-sub-string a1 (point)))
     (find-file f2)
     (while (string-lessp  "0" freq)
          (insert word ) (newline 1)(save-buffer)
          (setq freq (string-to-int freq))
          (setq freq (1- freq))
          (setq freq (int-to-string freq))
     )
     (find-file f1)
     (forward-line 1)
     (if (eobp) (setq not-eof nil))
)

;; Remove-spaces file
;;--------------------

(find-file "freq1")
(goto-char (point-min)) (setq not-done "t")
(while  not-done
      (delete-char 1) (end-of-line)
      (backward-char 1) (delete-char 1)
      (forward-line 1)
      (if (eobp) (setq not-done nil))
)
(save-buffer)


;;  Create-set2 file
;;-----------------------

(defun get-sub-string (a1 a2)
       (buffer-substring a1 a2)
)

(defun do-nothing ()
)


;;  MAIN PROGRAM

(find-file "~mhashi/IND/semantic-net/mail/freq1")
(goto-char (point-max))(setq a1 (point))
(setq not-done "t")
(while not-done
    (search-backward "Index terms for file  " (point-min) t)
    (end-of-line)(backward-word 1) (setq a3 (point))
    (forward-word 1)(setq set2-file (get-sub-string a3 (point)))
    (beginning-of-line)
    (setq ind-term (get-sub-string (point) a1))
    (setq set2-filename (concat "~mhashi/IND/semantic-net/mail/similar/set2/" set2-file))
    (find-file set2-filename)
    (insert ind-term) (save-buffer)
    (find-file "~mhashi/IND/semantic-net/mail/freq1")
    (setq a1 (point))
    (if (bobp) (setq not-done nil))
)

;; Find-similarity file

(defun get-sub-string (a1 a2)
       (buffer-substring a1 a2)
)

(defun do-nothing ()
)

(defun add-one-to-this-intersection ()
   (setq count1 (1+ count1))
   (find-file "~mhashi/IND/semantic-net/mail/similar/similar-temp1")
```

```
   (insert set1-term " , ")(save-buffer)
)

(defun consider-this-intersection ()
     (find-file "~mhashi/IND/semantic-net/mail/similar/similar-temp1")
     (goto-char (point-min))
     (if (eobp) (setq similar-temp1 "t")
              (setq similar-temp1 nil)
     )
     (if similar-temp1 (do-nothing)
        (end-of-line) (backward-char 2) (kill-region (point) (point-max))
        (save-buffer)
        (find-file "~mhashi/IND/semantic-net/mail/similar/similar-temp2")
        (kill-region (point-min) (point-max))
        (insert-file "~mhashi/IND/semantic-net/mail/similar/similar-temp1")
        (save-buffer)
        (setq the-similar-document next-set2-file )
        (setq count2 count1)
         )
)

(defun the-similaraty-is-null ()
(find-file "~mhashi/IND/semantic-net/mail/similar/similar-documents")
(insert "-------------------------") (newline 1)
(insert "There is no any document similar to the document "
      file-name-in " which has the following index terms:")
(newline 1) (insert-file file-name)
(newline 2) (save-buffer)
)

(defun the-similaraty-is-not-null ()
(find-file "~mhashi/IND/semantic-net/mail/similar/similar-documents")
(insert "-------------------------") (newline 1)
(insert "The document " the-similar-document " is the most
       similar one to the document " file-name-in )
(newline 1) (insert "The intersection set of index terms is:")
(newline 1) (insert "{ ") (end-of-line)
(insert-file  "~mhashi/IND/semantic-net/mail/similar/similar-temp2")
(end-of-line) (insert " }")(end-of-line) (newline 2) (save-buffer)
)

;; MAIN PROGRAM

(setq set2-eof nil) (setq max "5")
(find-file "~mhashi/IND/semantic-net/mail/similar/similar-documents")
(kill-region (point-min) (point-max))(save-buffer)
(find-file "~mhashi/IND/semantic-net/mail/similar/set1/set1d")
(goto-char (point-min))
(if (eobp)  (setq not-eof nil)
          (setq not-eof "t")
)
(while  not-eof
(find-file "~mhashi/IND/semantic-net/mail/similar/similar-temp1")
(kill-region (point-min) (point-max))(save-buffer)
(find-file "~mhashi/IND/semantic-net/mail/similar/similar-temp2")
(kill-region (point-min) (point-max))(save-buffer)
(find-file "~mhashi/IND/semantic-net/mail/similar/set1/set1d")
(setq a1 (point)) (forward-word 1) (setq a2 (point))
(setq file-name-in (get-sub-string a1 a2))
(setq file-name (concat "~mhashi/IND/semantic-net/mail/similar/set1/" file-name-in))
(forward-line 1)
(if (eobp)  (setq not-eof nil))
(find-file file-name)
(goto-char (point-min))

;;   Here is the beginning of the file from set1
;;   Make sure that it is not empty
(setq set2-eof nil)
(if (eobp)  (setq set1-eof nil)
          (setq set1-eof "t") (setq set2-eof "t")
)
(setq next-set2-file 0) (setq count2 "0")
  (while set2-eof
  (setq count1 0)(setq set1-eof "t")
  (find-file "~mhashi/IND/semantic-net/mail/similar/similar-temp1")
```

```
  (kill-region (point-min) (point-max))(save-buffer)
  (setq next-set2-file (1+ next-set2-file ))
  (setq next-set2-file (int-to-string next-set2-file))
  (setq set2-filename (concat "~mhashi/IND/semantic-net/mail/similar/set2/" next-set2-file))
  (find-file set2-filename) (goto-char (point-min))
  ;; If the doc. in set is empty there is no need for the comarision
  (if (eobp) (setq set1-eof nil))
  ;; Be careful not to compare the program in set1 with itself in set2
  (if (string-equal file-name-in next-set2-file) (setq set1-eof nil))
  (find-file file-name) (goto-char (point-min))
  (while set1-eof
  (find-file file-name)
  (setq a1 (point)) (end-of-line)
  (setq set1-term (get-sub-string a1 (point)))
  (forward-line 1)
  (if (eobp) (setq set1-eof nil))
  (find-file set2-filename) (goto-char (point-min))
  (if (search-forward set1-term (point-max) t)
      (add-one-to-this-intersection)))
  (setq count1 (int-to-string count1))
  (if (string-lessp count2 count1) (consider-this-intersection))
  (if (string-equal next-set2-file max) (setq set2-eof nil))
  (setq next-set2-file  (string-to-int next-set2-file ))
  (setq count1 (string-to-int count1)) )
  (find-file "~mhashi/IND/semantic-net/mail/similar/similar-temp2")
  (beginning-of-line)
  (if (eobp)   (the-similaraty-is-null)
  (the-similaraty-is-not-null)
  )
 )


;; Sort-and-Uniq file
;;-------------------------

(defun get-sub-string (a1 a2)
       (buffer-substring a1 a2)
)

(defun do-nothing ()
)

(defun  write-this-line ()
       (find-file file-name2)
       (setq as (int-to-string a))
       (insert "   " as " " first " ")(newline 1)
       (setq not-eof nil)
       (save-buffer)
)

(defun write-first ()
       (if (eobp) (setq not-eof nil))
       (find-file file-name2)
       (setq as (int-to-string a))
       (insert "   " as " " first " ")(newline 1)
       (save-buffer)
)

(defun remove-spaces ()
      (find-file file-name2)
      (goto-char (point-min))
      (if (eobp)   (setq not-eof nil)
      (setq not-eof "t")
)
  (while not-eof (end-of-line)
      (just-one-space)
      (forward-line 1)
      (if (eobp) (setq not-eof nil))
  )
)
;;  MAIN  PROGRAM
(setq file-name1 "occur3")
(setq file-name2 "occur")
(setq a 0)
(find-file file-name1) (beginning-of-buffer)
```

```
(setq reverse "t")
(sort-lines reverse (point-min) (point-max))
(save-buffer)
(if (eobp)  (setq not-eof "t")
        (setq not-eof nil)
)

; get first line
(if not-eof (setq not-eof nil)
     (setq a1 (point)) (end-of-line)
     (setq first (get-sub-string a1 (point)))
     (setq a (1+ a)) (forward-line 1)
     (if (eobp)  (write-this-line)
     (setq not-eof "t")  )
)

(while not-eof
      (find-file file-name1)
      (setq a1 (point)) (end-of-line)
      (setq second (get-sub-string a1 (point)))
      (if (equal first second) (setq a (1+ a))
              (write-first)
              (setq first second)
              (setq a 1)
      )
      (find-file file-name1)
      (forward-line 1)
     (if (eobp) (write-first) )
)

(remove-spaces)
(find-file "occur")
(goto-char (point-max))
(backward-word 1) (forward-line 1)
(kill-region (point) (point-max))
(goto-char (point-min))
(while (search-forward " 1 " (point-max) t)
      (beginning-of-line) (kill-line 1)
)
(goto-char (point-min))
(delete-blank-lines)
(save-buffer)

;; Sort file

(find-file "freq1") (goto-char (point-min))
(sort-numeric-fields -1 (point-min) (point-max))
(save-buffer)
```