Towards the Design of an Intelligent Hypermedia Architecture

Gilles Fayad

B. Eng., 1988

Department of Electrical Engineering

McGill University

Montréal

March, 1992

A thesis submitted to the Faculty of Graduate Studies and Research

in partial fulfillment of the requirements for the degree of

Master of Engineering

Abstract

A design for a unified model based on the hypermedia paradigm is proposed as a means to a better synergism of the functionalities of different knowledge manipulation tools. A survey of past and present hypermedia systems has been achieved, and their characteristics examined. Intelligent hypermedia-based system architectures have been evaluated, from which functional requirements for a unified architecture have been derived. Equivalence mappings based on Petri nets, that equate the structure and behavior of hypermedia networks and expert systems in a loose way, have been developed. A unified model, that synthesizes the structural and behavioral equivalences among different knowledge representations in an object-oriented architecture is proposed.

Résumé

Un modèle d'unification d'outils de connaissance basé sur le paradigme d'hypermédia est proposé pour une meilleure synergie des fonctionalitées inhérentes aux dits modèles. L' étude des systèmes hypermédias passés et présents a permis d'évaluer leurs caractéristiques. Les architectures des systèmes dits intelligents, et qui utilisent le concept d'hypermédia, ont aussi été étudiées, et ont servies à énumérer les fonctionalitées requises pour une architecture unifiée Des relations d'équivalence basées sur les réseaux de Pétri ont été dérivées, et permettent d'homogénéiser de façon globale les structures et les mécanismes des réseaux hypermédias et des systèmes experts. Un modèle unfié est alors proposé, qui synthétise les équivalences structurelles et mécaniques de ces différents outils au sein d' une architecture orientée objets.

Acknowledgements

I would like to thank my research advisor Dr. David Lowther for allowing me to discover the fascinating fields of hypermedia and expert systems; and for his guidance and patience throughout this work. His unconditional support allowed me to complete the task of writing this thesis. I treasure all the members of the CADLab, staff and students, past and new, and all the moments and thoughts we shared. Special thanks to Dr. Silvester, Raymond Sassine and Derek Dyck. I am indebted to Dr. Richard Furuta and P. David Stotts from the University of Maryland, for making his hypertext prototype available.

Je voudrais aussi remercier ma mère et mon frère pour leurs éternels encouragements, et mon père pour m'avoir montré la voie à suivre. En sa mémoire, je dédie cette thèse à ma mère. Je voudrais aussi mentionner Denise Pelletier, Raja Abi Dib, Naji Mouawad, Ibrahin El-Husseini, Emile Saab et Lustucru, pour leur aide, leur patiente, et leur amitié. Je les en remercie infiniment. Quant à Julie, ma gratitude et mon affection lui sont tout naturellement acquis.

Hypertext

"Consider a future device for individual use, which is a sort of mechanized private file and library. It needs a name. To coin one at random, "memex" will do. A memex is a device in which an individual stores all his books, records and communications, and which is mechanized so that it may be consulted with exceeding speed and flexibility. It is an enlarged intimate supplement to his memory. [...] So far, all this is conventional; a mere projection of present-day mechanisms and gadgetry. It affords an immediate step, however, to associative indexing, the basic idea of which is a provision whereby any item may be caused at will to select another, immediately and automatically. This is the essential feature of the memex; the process of tying items together to form trails is the heart of the matter."

– *"As We May Think"*, Vannevar Bush [Atlantic Monthly, July 1945].

"So it can be done. Will it be done? Ah, that is another question. The great digital machines of today have had their exciting proliferation because they could vitally aid business, because they could increase profits. The libraries still operate by horse-and-buggy methods, for there is no profit in libraries Government spends billions on space since it has glamor and hence public appeal. There is no glamor about libraries, and the public do not understand that the welfare of their children depends far more upon effective libraries than it does on collecting a bucket of talcum powder from the moon. So it will not be done soon. But eventually it will."

– *"Memex Revisited"*, Vannevar Bush [Science Is Not Enough, May 1967].

Table of Contents

List of Figures

Chapter 1 Introduction

1.1 The Scope of the Thesis

The objective of this thesis is two-fold: to provide a survey of hypertext with a focus on intelligent systems; and to present a model for integrating different knowledge tools within an environment that supports the hypermedia paradigm. The objective is to investigate new alternatives that the hypermedia model can offer to the domain of knowledge engineering.

The survey on hypertext was dictated by the lack of standards and structured evolution in the hypertext domain and represents a necessary background to the knowledge based approach to hypertext.

The ultimate aim of knowledge engineering is to develop a methodology that captures and delivers expertise in various domains in a natural and intuitive way. Hypertext and hypermedia have shown a great potential, both in human factors and information management. Coupling hypermedia and knowledge based systems provides a framework for *intelligent information systems*. These are defined here as systems that allow manipulation and processing of information in an intelligent manner.

This thesis is organized into two parts. Part one introduces hypertext and *static* architectures (see section 1.3.1) and consists of chapters one to three. Chapter one covers the historical background and provides a definition for the often confused terms of hypertext, hypermedia and multimedia. Chapter two is a general survey of hypertext systems, and a formal hypertext architecture is described in chapter three.

Part two focuses on intelligent hypermedia systems and consists of chapters four to six. Chapter four describes actual intelligent hypermedia features and architectures; chapter five introduces the functionalities of a unified model and derives the equivalence mappings between hypertext and expert systems structures and behaviors; and chapter six describes the object-oriented unified model. Chapter seven attempts an evaluation of the model, summarizes the achievements and proposes future enhancements.

1.2 Historical Background and Survey

1.2.1 The Hypertext Pioneers

The father of hypertext is Vannevar Bush, first director of the Office of Scientific Research and Development under president Roosevelt in the 1940s. However, the particularity of Bush is not in his position, but rather in his influence over some pioneers of the computer era Giants such as Wiener, Licklider and Engelbart shaped cybernetics, man-machine communications and interactive computing, and have either been under his supervision or were profoundly influenced by his famous article "As We May Think" [19], which was the earliest and clearest discussion of the idea that information processing technology could be used to amplify human memory and thinking. This article is recognized as the root of the modern hypertext concept.

In the article, Vannevar Bush described what is today known as facsimile, speech recognition and artificial intelligence, all applied to the problem of the growth of scientific information. He postulated a hypothetical machine, the Memex, that would become an extension of the human mind, and that embodied all the essential features of a hypertext system.

The Memex is essentially a storage of information within a conventional desk,

that allows for classification and retrieval of documents. Unlike conventional systems, Bush's Memex is not based on indexing[19]:

> *"Our ineptitude in getting at the record is largely caused by the artificiality of systems of indexing ... Having found one item, one has to emerge from the system and re-enter on a new path".*

Instead, Bush advocates *selection by association*:

> *"The human mind does not work that way. It operates by association. With one item in grasp, it snaps instantly to the next that is suggested by the association of thoughts, in accordance with some intricate web of trails carried by the cells of the brain. It has other characteristics of course; trails that are not frequently followed are prone to fade, items are not fully permanent, memory is transitory. Yet, the speed of action, the intricacy of trails, the details of mental pictures, is awe-inspiring beyond all else in nature."*

The functional description of the Memex in [19] is considered as the hallmark of today's hypertext. His machine was essentially mechanical, partly because of his background in mechanical computers, and partly because electronic computers had not yet reached a suitable level of development. However, Bush did not rule out the digital computer, and reconsiders the design of the Memex in the light of current technological advances [20] in 1967. One year later, Douglas Engelbart implemented the first operational "Memex" at the Stanford Research Institute (SRI). His system was demonstrated at the 1968 Fall Joint Conference. Among the input-devices invented by Engelbart, one was used to move a cursor on the screen, and was called a *mouse*.

While at Harvard in 1960, Ted Nelson was inspired by Bush's paper and focused on the computer as a medium that would boost creative thoughts. He decided to write a forty-thousand line machine-language program to implement his ideas,

"but like many beginning computerists, I mistook a clear view for a short distance" [85]. His specifications included "historical backtrack" on paths and versioning, all in a "point and click" metaphor. He was the first to coin the term *hypertext*, as a literary process that allows non-sequential forms of writing made possible by the advent of the computer. While Engelbart was interested in the technology, Nelson was interested in the community: A broad hypertext network with an economic structure and an automatic royalties system. "The software is on its way. But what is really lacking are the visionary artists, writers, publishers and investors who can see the possibilities and help carry such ideas into reality."

Andries Van Dam met Nelson and Engelbart at the 1967 and 1968 Joint Conferences. The potential Van Dam saw in hypertext is in teaching. In 1983, he contributed to the creation, at Brown University, of the Institute for Research on Information and Scholarship (IRIS) that designs and evaluates scholars' workstation software.

1.2.2 The First Hypertext Systems

NLS/Augment

Engelbart's "Memex" implementation, NLS, used the latest technological innovations of the 60s: a dedicated time-sharing computer with 65 Kbytes of memory and 96 Mbytes of storage, high-resolution television monitors and three input devices (a keyboard, a mouse and a five-key handset aimed at replacing the keyboard when used in conjunction with the mouse). The system was intended for collaborative work and supported the hierarchical structuring of documents. It provided *level views* (i.e. it displayed only the document paragraphs to a given depth), *truncation* (i.e. it displayed only the first n lines of the document), and *filtering* based on keywords in context (KWIC). It also supported vector graphics and television signals superimposed on the screen content using dedicated analog hardware. Engelbart's

system is now known as "Augment".

HES

In 1968, Ted Nelson and Van Dam developed another hypertext prototype: the hypertext Editing System (HES) [35], as a display application. The data structure consisted mainly of pointers and editing involved manipulating these pointers instead of the raw text. The HES featured unidirectional links automatically managed into menus, cross-references and indices. Its designers experienced *disorientation*, a problem characteristic of hypertext where the user gets lost in the structure due to the lack of orientation cues. They tried to solve it by providing *guided tours* demos. The HES was demonstrated to publishing corporations but failed to enter the commercial world because it was perceived as too complex. However, it was used by the Houston Manned Spacecraft Center for the production of Apollo documentation.

FRESS

The second system developed at Brown was the File Retrieval and Editing System (FRESS) [35], and wa. a time-sharing multiterminal enhancement of the HES. designed it in 1969 and Phillips commercialized it in 1971. In FRESS, Van Dam wanted to improve Engelbart's ideas and added free-form editing, unlimited statement size and portability. Accordingly, FRESS brings the notion of virtual I/O, bidirectional links with attributes, dynamic editing with undo and autosave features and a visualization of all the structure in the text. Multiple windows and vector graphics were supported by the 16 Kbytes mini-computer.

EDS

The successor of FRESS was the Electronic Document System (EDS), better known as the Dynabook [1]. The EDS was primarily oriented towards producing graphical documents. It was composed of three components: a viewer, an authoring paint tool and a hypertext authoring tool. The EDS also featured three automatic navigation aids: a time line that allowed backtracking and recovery of the node status at a given time, a "neighbors" display that consisted of a filmstrip of all the possible incoming nodes iconified on the left and all the possible outgoing nodes iconified on a similar filmstrip on the right, and a visual index of buttons of page miniatures arranged by keywords and color-coded by chapter.

However, the particularity of the EDS is in its architecture: a finite-state automaton that allows for dynamic changes in the hypertext. Unfortunately, the EDS was difficult to manipulate and the authoring tools were found to be lacking knowledge about the hypertext context.

ZOG/KMS

KMS [3] first started at Carnegie-Mellon as a menu-based display system under the code name ZOG. The first commercial version appeared in 1983

KMS (Knowledge Management System) adopts the hierarchical structure of Augment. It consists of a distributed database of frames. The frames consist of text, graphics and bitmaps in a WYSIWYG environment. The frame content is formatted into a title, a name, a body that holds the frame information content, two types of links: "tree items" (i.e. a menu) and "special items" (i.e. cross-referential links), and command items. An unlimited number of links allows any text item within a frame to be linked to any other frame. KMS also allows for limited

[1] The system actually had no official name (personal communication with Nicole Yankelovich, IRIS)

procedural attachments to the frames, and independent databases to be linked together. Frames are displayed as one or two pages on the whole screen, and can be assembled into a linear document for hardcopy purposes.

KMS innovated by providing *contextual distinctions*: the location of the cursor determined what operations were available. Although it used a graphical interface, KMS designers did not feel the need for a graphical browser [60].

It is perhaps the largest and most thoroughly tested hypertext system in service: It was installed as a computer-based information management system on an aircraft carrier and also provided operators of a nuclear plant quick access to emergency procedures. It has also been used for policy analysis, authoring and communications.

1.3 Defining Hypertext, Hypermedia and Multimedia:

The systems described above clearly show the wide variety of hypertext systems, and why, till now, neither a formal definition, nor applicable standards have been defined. A suitable definition is required and should clearly set the boundary between hypertext, hypermedia and multimedia.

1.3.1 Hypertext

Even though the concept is intuitive, the definition of hypertext is a highly controversial topic. The reason for this is that *the term 'hypertext' has been used quite loosely in the past 20 years for many different collections of features* [29].

The most common definition is that of Ted Nelson who coined the term hypertext [70]:

"A combination of natural language text with the computer's ability for interactive branching or dynamic display ... of a non-linear text".

The most succinct definition is that "a hypertext is a network of information nodes connected by means of relational links" [77].

A *hypertext system* is then a hardware and software configuration that allows users to manage and access the information it contains.

A more detailed definition would be that hypertext is a generalization of text documents through the computer medium that allows non-sequential formatting of the information by freeing the logical structure of the text from the physical structure of its one-dimensional conventional medium, paper.

Nodes, Links and Networks

Text is then divided into information chunks called nodes. These nodes are explicitly connected to other nodes through links. Nodes and links form a network which can have different architectures (see chapter 3).

Navigation and Browsing

Navigating a hypertext document usually consists of a non- sequential visiting of the nodes accessed through the links, in a process that is usually known as *browsing* and that constitutes the hallmark of hypertext.

In order to be non-sequential, the hypertext links should be able to provide many-to-many relations and the hypertext mechanism (thereafter called hypertext engine) should allow for backtracking. Another requirement often neglected in hypertext systems is an explicit graphical representation of the network that would reduce disorientation. Many other features stretch beyond these basic requirements, and are not common to all hypertext implementations.

Static Hypertext

Static hypertext is defined as hypertext systems that do not allow dynamic modification of their structure, status and content (i.e. without user intervention). Static hypertext architectures have limited capabilities at the level of processing.

1.3.2 Hypermedia

Hypermedia is the merging of the hypertext concept with multimedia technology.

It extends the hypertext concept to non-textual information chunks (practically graphics, video and sound) and "produces complex, richly interconnected and cross-referenced bodies of multimedia information" [95]. Accordingly, many definitions merge hypertext and hypermedia (see for example [72]).

Differentiating Hypertext and Hypermedia

The novelty of hypermedia with respect to hypertext is that it allows for *temporal* information chunks such as animated graphics, video and sounds. Rigorously, the hypertext engine should then be extended to dynamic temporal links for "non-leaf" nodes (i.e. nodes with outgoing links) and the browsing paradigm should be broadened so as to allow for the manipulation of these dynamic information chunks [16] [24]. As such, hypermedia deserves to be differentiated from hypertext.

A hypermedia system is then a truly innovative medium that integrates different media within a single system.

1.3.3 Differentiating Hypermedia and Multimedia

Whereas most computer tools today are geared toward the manipulation of alphanumeric data, multimedia integrates different data types such as text, graphics, video and sound in a single medium, solution or environment. Each data type is subject to some interactive control by the user.

The emergence of multimedia technology was made possible by the advent of high-end personal workstations, CD-ROM and videodisk storage technologies, and high bandwidth networks. The first multimedia applications were primarily geared toward collaborative work [39] and multimedia electronic mail

Multimedia and hypermedia are often confused, mainly because hypermedia applications apply multimedia technologies. However, being multimedia is not sufficient for being hypermedia: multimedia technologies can rely on pure database management techniques [45]; hypermedia with no browsing capabilities on multimedia nodes is not "pure hypermedia".

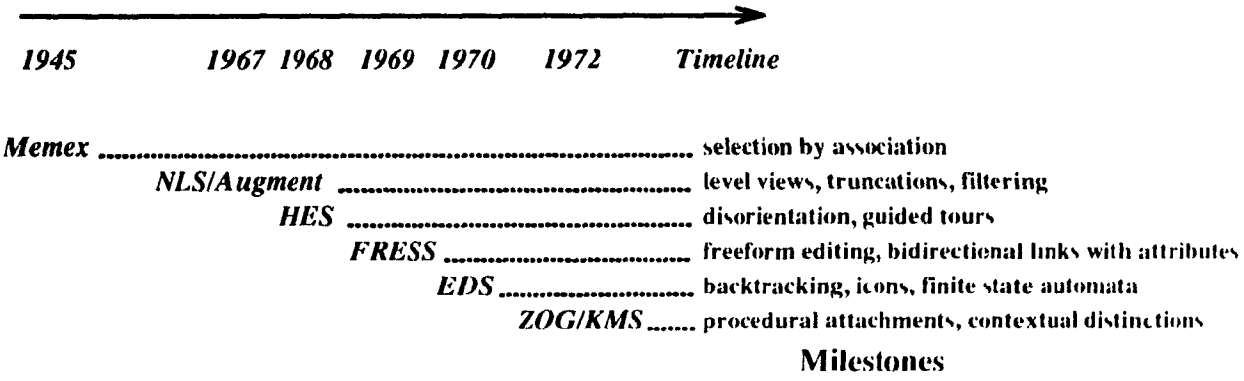

Figure 1.1: The evolution of hypertext specifications

1.4 Conclusion

A historical background has been given in order to provide an introduction to the terminology and diversity of hypertext. The historical systems surveyed carry

the foundations of all subsequent hypertext systems, as depicted in figure 1.1. Definitions that differentiate between hypertext, hypermedia and multimedia; have then been articulated. The next chapter will provide a taxonomy and survey of hypertext systems.

Chapter 2 A Taxonomy and Survey

2.1 Introduction

The 1980's witnessed the introduction of hypertext in the professional world , and the first commercialization of hypertexts such as Guide [17] and Hypercard [56] Enumerating all the hypertexts of the 80s is out of the scope of this thesis However, a taxonomy and survey of hypertext systems will be attempted in this chapter Hypertext systems will be approached from the viewpoint of the application domain, and surveyed in light of the characteristics, properties and functions they incorporate.

2.2 The Classical Taxonomy

Conklin [29] has provided a taxonomy that classifies hypertexts into four categories: macro-literary, browsing systems, problem exploration and general hypertext technology. Macro-literary systems embed the technologies to support large on-line libraries in which document links are machine-supported. Browsing systems are similar in characteristic but of a smaller scale since they are usually restricted to a certain knowledge domain. Exploration tools exploit the hypertext capability of supporting unstructured thinking for problems where many disconnected ideas occur. General hypertext technology refers to general purpose systems designed around a hypertext architecture. Worth detailing is the macro-literary class of systems.

2.2.1 Macro-Literary Systems:

Memex, Augment and Xanadu are the archetypes of macro-literary systems: they manipulate huge volumes of information and attempt to build a consistent interface. Furthermore, the body of information is to be constantly updated by readers, integrating them with authors in a single active community close to the "global village" metaphor. Immense problems have yet to be solved; these are mostly related to managing vast amounts of information (e.g. distribution, unique identification, etc ..). Despite such problems, Augment is operational, the Xanadu [69] file server has been designed and publicized by Nelson's company in association with Autodesk, and the *world wide web* project is being implemented at CERN (Centre Européen de Recherche Nucléaire). Such systems will certainly have a huge social impact, as Nelson mentions:

> *The plan is to open the franchise ... for a chain of McDonald's-like information stands, which will form a repository network. You'll be able to put your private documents in, and thus it will be a mini-self-storage system for information . it will run on the Sun, the Macintosh and the 386s. This will hit the LAN and serve as many people at once with the fragments necessary to support their documents.* [1] *[69]*

While Conklin's taxonomy has become a reference, it blurs the distinction between size, application domain and architecture. Macro-literary systems differ from browsing systems at the level of size only, while general hypertext technology is too vague a classification for different architectures. We propose another taxonomy that classifies hypertext systems according to classes of applications.

[1] Though typically "Nelsonian", the above statement is an illustration of the difficulties to be overcome and could easily turn "Orwellian"

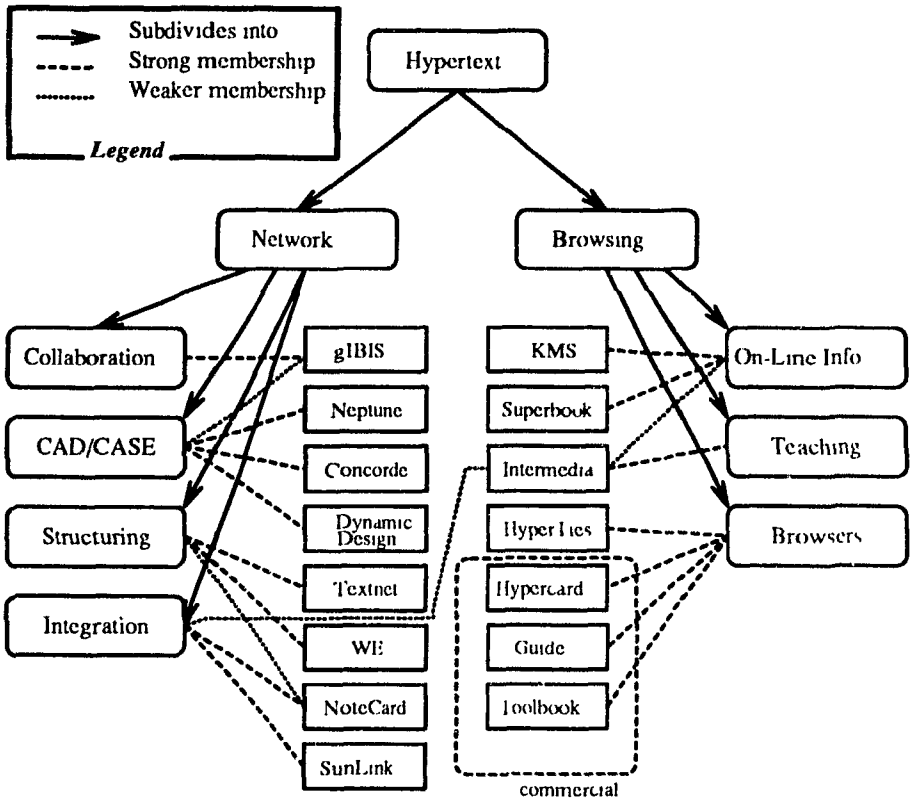

Figure 2.1: A new taxonomy of hypertext systems

2.3 A New Taxonomy

The hypertext concept essentially revolves around two paradigms: the network structure applied to information, and the authoring/browsing mechanism The network structure provides cross-referencing and a non-linearity that extends beyond hierarchical classifications. The structuring aspect of hypertext is predominant in collaborative work, CAD and CASE, and literature tools that explore non-linearity. The authoring/browsing mechanism removes the boundaries between the user and the programmer. All the hypertext systems are combinations of these two concepts applied to application classes, as depicted in figure 2.1. Briefly, the list is:

- collaborative work [39, 30, 63];
- CAD/CASE [37, 54];

- information structuring [94, 50, 89, 87];
- integration [92, 78]
- On-Line Information [3, 83, 38, 98]
- tutoring [100, 23]
- browsing [35, 17, 56, 88]

2.4 Collaborative Work

Collaborative work hypertext applications use the network structure of hypertext to capture the nature of the information and to homogenize it. The authoring/browsing mechanism is used to capture the collaborative nature of the interaction. The main approach to collaborative work using hypertext is collaborative argumentation [30, 63, 87] as applied in issue-based information systems.

2.4.1 IBIS:

Issue-Based Information Systems (IBIS) are geared toward problems that lack a formulation and that could not be solved using classical analysis techniques. IBIS is designed to facilitate the "capture of early design deliberation through collaborative construction of the network of information" [29]. The IBIS method and its cousin, the PHI approach [62] have been used successfully in architectural design, urban planning and at the World Health Organization. gIBIS [8] has been used in a project called the Design Journal, "aimed at providing a team of system designers a medium in which all aspects of their work can be computer mediated and supported." Documents such as requirements and specifications are integrated together with interviews with users, scenarios, design reviews, early design notes, design decisions, internal constraints, minutes of meetings, etc ...

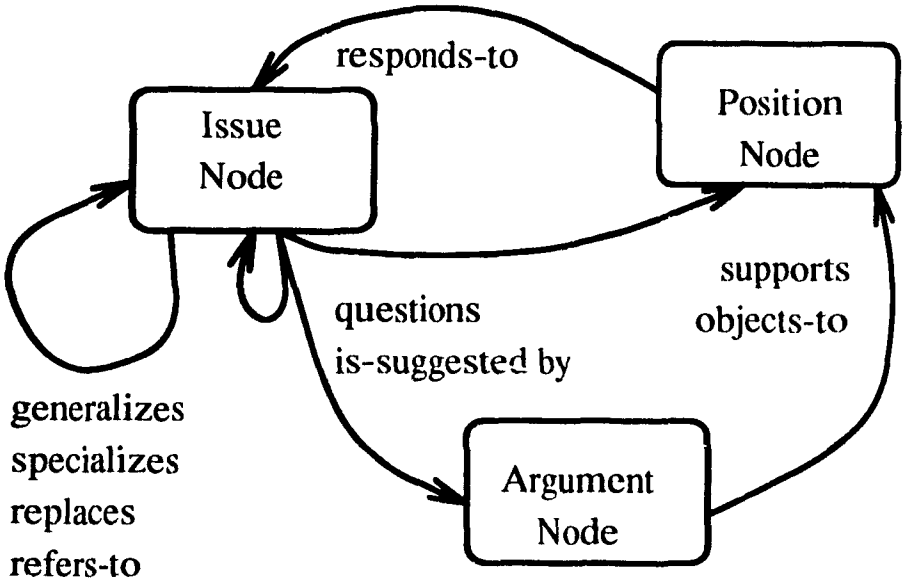

Figure 2.2: Specialized semantics for argumentation

The hypertext version of IBIS, gIBIS, provides specialized semantics at the node and link level. Three types of node cover issues, positions and arguments. These nodes are interconnected with nine possible types of links (see figure 2.2). For example, an issue is *suggested-by* an argument, which itself *objects-to* a position that *responds-to* another issue. Issues and their respective positions and arguments can be clustered into subnets. The nodes in the current network can be presented as linearized in a hierarchical index in which the ordering of positions and arguments follows a depth-first traversal of the primary links of the issue. As such, they are represented by a special icon and all their connective links to external nodes are not displayed on the graphical browser.

Information Presentation The IBIS user-interface provides four tiled windows: a graphical browser, a structured index, a control panel and an "inspection window". The graphical browser shows a dynamically updated scrollable local view of the network centered on the local issue and its ramifications depicted in full detail. Clustered subnets of issues are represented by a special icon and all their connective links to external nodes are not displayed on the graphical browser. The graphical

browser also provides a small global view that depicts the entire network. Unlike the local view, node labels, link-type icons and secondary links are pruned for the sake of clarity. The structured index displays the linearized list of issues, their positions and arguments. Information objects are accessible in a "point and click" paradigm from both the graphical browser and the index. A configuration panel allows tailoring of the index keys (i.e. by author, keyword or node label). The control panel provides diversified services, ranging from interface configuration to query templates or browsing control. Querying, like creation, uses sample data-entry templates where the user specifies the values of the attributes of the object.

Other argumentative systems have also applied the hypertext concept for the semantic structuring of information [63, 87].

2.5 CAD and CASE

Other hypertext systems [44, 37, 12, 54] use the same semantic structuring based on attributes to structure documentation relevant to CAD and CASE. Hypertext implementations for this application class develop more powerful *filtering* mechanisms and support *versioning*. Neptune [37] is a classical CAD-oriented hypertext system.

2.5.1 Neptune

Neptune [37] is a hypertext system developed by Tektronix and geared toward CAD applications. It is designed as a layered architecture on top of the hypertext Abstract Machine (HAM) transaction-based server. The HAM server provides storage and access mechanisms for nodes and links, distributed networked access with multi-user synchronization and transaction-based crash recovery. An infinite number of application layers can be built on top of the HAM. Neptune communicates with the

HAM through a remote procedure call, the HAM being run as a separate process on a network server.

Querying

Nodes and links have attributes attached to them at the HAM level. These attributes are used in two query mechanisms: LinearizeGraph that performs a depth-first traversal of outgoing links ordered by their offset within the document, and GetGraphQuery that performs a classical query on the attributes, retrieving nodes and their corresponding links.

Information Presentation

Querying, reading and editing is performed through a user-interface written in SmallTalk-80 and that provides four specialized "browsers": a graphical browser that contains a navigation pane for zooming and panning, and filtering panes for editing the visibility predicates of both nodes and links; a document browser that has four top panes to pick up nodes and a node browser to view the node; a stand-alone node browser to edit a node; and a node differences browser where two versions of a document are put side by side, and differences highlighted.

Versioning

The node differences browser reflects the importance of versioning control in CAD and CASE applications. Versioning is often achieved through backward-deltas [91] where only the differences among versions are retained. Neptune supports two mechanisms for link attachment to a version of a node, differentiating between the current (default) one and a particular one.

Most of the functionalities associated with Neptune (i.e. versioning, querying,

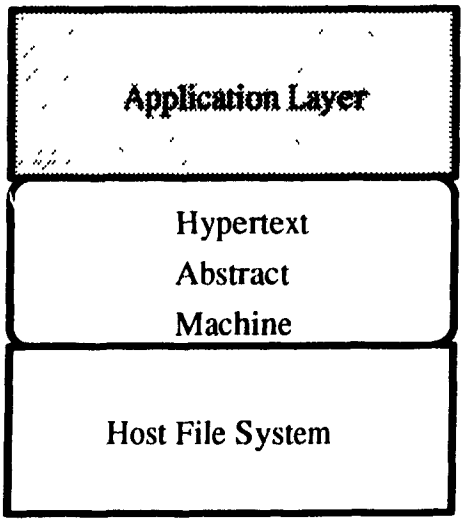

Figure 2.3: The HAM layered architecture

filtering) are inherent in the HAM architecture. Their tailoring to specific CAD domains is performed by a presentation level, on top of the HAM layer (see figure 2.3). Dynamic Design [12], for example, is a CASE tool also built on top of the HAM server.

2.6 Text Structuring Tools

Text structuring tools like Textnet [94] and WE (Writing environment) [89] explore the non-linear aspects of hypertext applied to the communication processes of reading and writing. They differ from the previous classes in that they focus on the *presentation* of information.

2.6.1 Textnet

Textnet [89] is the product of Randall Trigg, who wrote the first PhD thesis on hypertext. The system is intended to investigate strategies for text organization for an on-line scientific community. It uses a directed graph with labeled nodes

and links. Nodes can be one of text nodes and "toc" (i.e. table of contents) ones, the first having a *pointer-to-text* field and the second having a *child-out* field that encompasses all children toc nodes in an ordered list that forms a path. Paths are used to browse linear concatenation of text displayed on a "scanning window" and to produce hardcopies of the information. Links are labeled and point to nodes and to other links. This extra feature allows for criticism of the structure as well as of the content. Trigg defines three "trains of thought" based on this architecture of links and nodes: along the train of thought; side trips where examples, explanations and details can be found; and forks at which the train of thought divides into several sub-paths. The user interface consists of overlapping windows where linking options are represented as menus.

2.6.2 WE

WE [89] is based on a cognitive model that sees writing as the process of organizing a loosely structured network of internal ideas and external sources into a hierarchy, and then linearizing it into a linear stream of words, sentences, etc . Using WE, a user first "draws" his ideas as nodes in a graphical browser, without being forced to give them any particular structure. As some conceptual structures appear, the user can copy his nodes into a hierarchy window that has specialized tree operations. WE can be classified as an *outline processor*, i.e. a word processor specialized for processing outlines that inherits much of Engelbart's Augment functionalities, similar to ThinkTank and Framework [52].

2.7 On-Line Information

Hypertext is mostly perceived as an on-line information system that provides on-line manuals and technical information. Most of hypertext on-line information

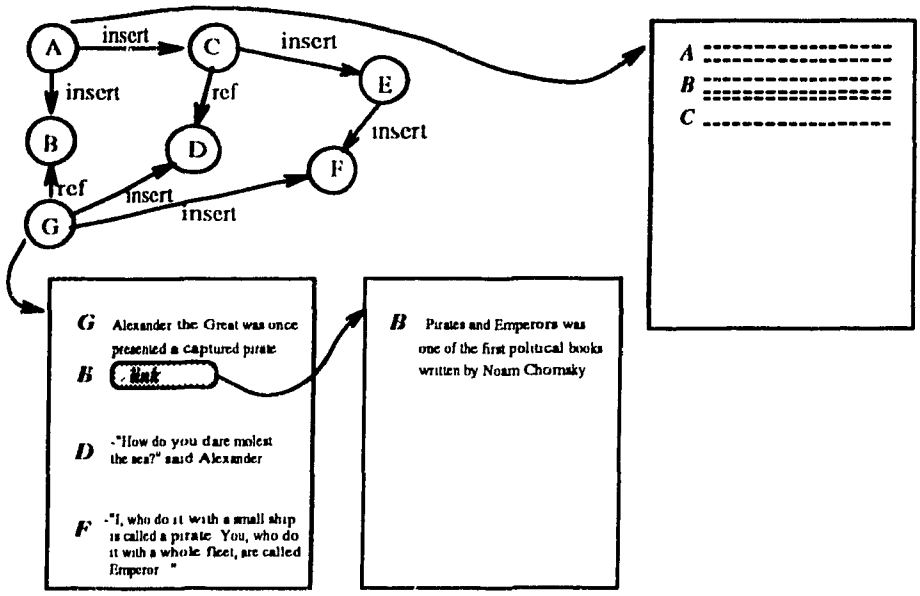

Figure 2.4: Integrating documents into one displayable unit

systems try to stress the *layout* of the displayed information [3, 38, 98] and target documentation management where the amount of documentation and the rate of updating it are huge. Consequently, they support hierarchical structures, without which updating large amounts of information becomes a very difficult task. Some systems specialize in the type of information and the automatic means for transferring information from printed material to a computerized format [83, 51].

2.7.1 SDE

The Symbolics Document Examiner (SDE) [98] has been developed as part of Symbolics Genera (software development/operating system) for Symbolics computers. The particularity of the SDE is in the presentation of the hypertext information: the directed graph is not used as its fundamental user-visible navigation model, as in most hypertexts. Instead, the contents of nodes are inserted at given locations in a WYSIWYG assembled highly structured document (see figure 2.4).

All links are directional and point to records or places in records. Four types of link refer to different insertion operations. Inclusion links insert the content

field of a given record. Precis links include the title and one-liner of a connected record, cross-referencing links are used for conventional cross-referencing, and implicit links refer to traditional hypertext browsing links. In figure 2.4, a network of seven nodes is connected using *insert* inclusion links and *ref* implicit links. If the information is accessed from node G, the nodes G, D and F get inserted in one presentation document, while a link points to the B node. Upon activation of the node, another presentation document gets assembled, that includes node B. If the network was accessed from node A, the contents of nodes A, B and C would be displayed. As in many hypertext on-line information systems, documents correspond to rigid formats and are modeled as structured records. The documentation is thus fragmented into a database of records with unique internal identifiers, assigned at creation time, and external identifiers such as name or type. The records are made up of four types of fields: content, accessory, audit and database information fields. Content fields provide a description of the node, accessory fields provide keywords and flags; audit information fields provide version numbers and publication status, and database information provide the server location of the record, the outward links, etc. . Document versioning is supported and maintained through the audit information fields. While KMS [3] also has formatted documents, it sticks to the casual one-to-one mapping between a hypertext node and its display representation.

2.7.2 OED

Dictionaries are also put on-line, in hypertext format. Since 1986, the University of Waterloo has been involved in the conversion of the Oxford English Dictionary (OED) into a hypertext structure in order to support on-line browsing. Experience has shown that browsing a dictionary is "an invaluable adjunct" to formal querying [83]. The browsing consists of a two-stage process: specify a pattern for querying, and navigate around the resulting object. Moreover, an on-line hypertext OED

removes the constraints imposed on the original paper medium and allows the integration of the OED with the user's task.

Most of the work consists in transferring the actual OED into machine-readable format, in the study of the characteristics of the 569,000 cross-references (more than 2 per entry), and in the design of the nodes and links data structure. The OED's nodes vary in size and structure, and the links are mainly lexicographical, cross-reference links being one type. Most of the actual research is geared toward the use of the OED as a generator of hypertext links to other documents and the design of specialized editors for creating a maintaining the sense structure that represents the first step in the definition of a word.

2.8 Teaching Assistance

User assistance in on-line documentation can be stretched to the educational field. Many projects have explored the on-line browsing alternative in scholar work. The NOVA hypertext in biology, the Shakespeare project at Stanford [46], the Perseus project [33] in classics at Harvard university, the *A la rencontre de Philippe* from MIT's project Athena [53] in teaching French, and many others [82]. Moreover, some hypertext systems such as Intermedia have been specifically produced for educational use.

2.8.1 Intermedia and InterNote:

Intermedia is the natural outcome of the research conducted at Brown and IRIS on FRESS and the EDS. Intermedia [100] uses hypertext technology to implement a tool designed to support teaching and research. It contains different applications and uses homogeneous mechanisms to link the contents of documents created with these applications in an Object-Oriented framework. The five integrated

applications consist of three editors (for text, graphics, and a timeline) and two viewers (scanned images and 3-D objects).

Information Manipulation The user-interface of Intermedia was designed to homogenize conceptually similar operations. Accordingly, some operations behave identically across applications and media. Making links, for example, consists of marking a certain region in a given application and medium, and linking it to another specified region in, possibly, a different application and medium. Multiple outgoing/incoming bidirectional links are supported. Links and regions (called blocks) are assigned descriptive properties stored apart from the documents in databases called *webs*. Some properties are automatically set (e.g. creation time and user ID), while others are left to the user, like "one-liner explanations" and keywords. These properties are managed by the web so as to provide viewing and navigation of the networks with a minimum of disorientation and cognitive overhead.

Webs Webs help reducing disorientation and cognitive overhead by providing *context partitioning* (i.e. filtering) and searching based on keywords and other properties of the documents. Three types of webs were initially implemented [100]: a global map that portrayed all linked nodes, a local map that displayed all the documents linked to a user-defined "focus" map, and a local tracking map that dynamically modified the local map during browsing.

InterNote Experiments conducted in English and biology at Brown University have shown a substantial increase in the critical thinking skills of the students exposed to different teaching approaches integrated in the Intermedia multiuser environment. IRIS has chosen to go further in this direction by refurbishing Intermedia into InterNote [23]. InterNote is designed as a tool to support small groups in annotative collaboration. Its main contribution is the concept of "warm links".

Warm links stand between static navigational "cold links" and automatically updated "hot links". They are used to copy the blocks, on demand, to and from the two linked nodes in a "push/pull" fashion. The copied blocks replace the information block of the initial or destination link.

2.9 Integration

Intermedia exhibits some kind of integration of the hypertext concept at the level of applications, much like the "cut-and-paste" metaphor that has evolved into a standard. The hypertext concept has been further stretched to environments [50] and even operating systems [78]. The Notecard system is a well-known hypertext system fully integrated in its environment, that inspired Bill Atkinson in the design of the Hypercard [56] system.

2.9.1 NoteCards

NoteCards [50] has much in common with Textnet (see section 2.6.1), though the aim has turned into developing a general hypermedia environment, and the design into supporting the task of transforming unrelated ideas into a homogeneous integrated interpretation. The semantic network structure found in Textnet has been retained, and 3x5 *notecards* are connected by labeled directional links. The notecards accommodate text, graphics and bitmaps. Two specialized types of cards define graphical browsers and "fileboxes" (i.e. folders) where cards are grouped together. The links semantic types are represented by different dashing styles in the graphical browser. Another feature retained from Textnet is the automatic assembling of documents. Changes in the linearized document however, do not propagate back to the original notecards.

Integrated to Lisp

NoteCards is implemented in the Xerox Lisp environment and is fully integrated within it. The programmer's interface consists of around a hundred Lisp functions and allows the user to create new types of cards , develop programs to process the network of cards, integrate Lisp programs as procedural attachments to the cards or integrate NoteCard itself into another Lisp-based application.

2.10 Browsing Systems

Browsing systems [88, 56, 17] are small-scaled hypertext systems that do not target specific application classes. On the other hand, their targeting of the general public audience makes them general-purpose hypertext systems that focus on simplicity and ease of use. Their primary interests are a user-friendly interface and solid cross-referencing tools. Some of them [88] separate browsing from authoring and implement them in two different packages. Commercial hypertext systems [56, 17] extend beyond simple browsing capabilities and additionally provide *scripting languages* and *external functions* for interfacing purposes.

2.10.1 Hyperties

Like OED, Ties started as an electronic encyclopedia, and soon changed its name into Hyperties so as to stress its hypertext aspects. Hyperties [88] started at the University of Maryland in the fall of 1983 as a practical toy-tool for browsing and as an experimental platform for studying the design of hypertext user-interfaces. Hyperties nodes consists of short articles in scrollable windows, with hypertext buttons represented as highlighted text. Activating a link leads to the replacement of the content of the window with that of the destination article. The system features paths, backtracking and a general index. String searching, bookmarks,

user annotations and multiple windows are to be part of future implementations. An authoring tool allows the user to define a title document, a 5 to 35 word small definition, the actual document and synonyms for the title. Marking hypertext links is performed using tildes { ... }. The authoring tool collects all references in-between tildes and prompts the author for the relationships to other documents.

Field Evaluation

Hyperties has been tested in museum exhibits and the Maryland University projects that served as field-experiments. It also served as an introduction to hypertext for a CACM issue (July 1988) and a hands-on introductory book on hypertext.

The particularities of Hyperties are in the input devices and in the activation of a link. A dozen empirical studies involving more than 400 participants have shown that touch-screens and keyboard keys were preferred over the mouse: keyboard keys (e.g. arrows, letters) showed 15% faster and were preferred by 90%. Touch-screens were preferred among three other selection strategies (mouse, keyboard keys, touchscreen and keypads). The other particularity is that, when a link is selected, the small description of the article is displayed, allowing an intermediate stage between unclear link selection and full article display.

2.11 Conclusion

The above hypertext systems are very heterogeneous, for the simple reason that hypertext is not an application field but a concept applied to different fields. Figure 2.5 shows clearly that filters and versions are mostly developed in hypertext systems dealing with design environments; that paths are mostly supported in fields where the material is traditionally presented on a one-dimensional media; and that very few architectures support procedural attachments, clusterings into subnets,

	Structure		Attributes		Graphical Browser	Procedural Attachments	Filters	Versions	Subnets	Paths
	Network	Hierarchical	Nodes	Links						
gIBIS	✓			✓	✓		✓	✓	✓	✓
HAM	✓		✓	✓	✓	✓	✓	✓		✓
TextNet	✓	✓		✓						✓
WE	✓				✓					
SDE	✓		✓	✓						✓
OED	✓			✓						
Intermedia	✓		✓	✓	✓		✓		✓	
NoteCards	✓	✓	✓	✓	✓	✓	✓			
Hyperties	✓									

Figure 2.5: Comparing features

or specialized operations for hierarchical structures.

Homogeneity of use and behavior appeared with commercial hypertext systems around the mid-80s. Hypertext and hypermedia have been commercially introduced with Filevision, Guide and especially Hypercard that is distributed with every Macintosh. They have become widely used in on-line documentation, ranging from help systems to small personal information bases often available on bulletin board systems.

The parallel world of research continues to evolve, concentrating on developing more or less formal models for retrieval [31, 18], hypertext architecture [1, 75] and intelligent systems [51]. Querying models [59] try to generate alternatives to database indexing schemes, and formal hypertext architectures [57, 41] try to build a platform for interchange [71]. As for intelligent systems, they try to exploit and

integrate the intuitive manipulation of hypertext as to enhance the functionality of actual intelligent tools.

Chapter 3 Formalizing Hypertext

3.1 Introduction

The diversity of hypertext systems makes a generalization of hypertext architectures, structures and models difficult. In this chapter, a classification of hypertext architectures and structures will be attempted. Hypertext entities, properties and functions will be described in order to capture the essentials of an abstract hypertext model (see figure 3.2). Based on these entities, the Dexter Reference Model will be presented. It provides a formal representation that models the core of all the major hypertext systems.

3.2 Hypertext Architectures

Though many pioneering hypertexts have a hierarchical architecture [39] [3], the network *graph-based* structure is the general architecture that superseeds the hierarchical one. The basic network model is the *hypergraph* [9] [93]. A hypergraph H is a triple $H =< N, L, E >$ where N is a set of nodes; L a set of labels that identify links; and $E : N \times L \rightarrow N$ is a set of edges [1].

[1] As defined by [93], the edges of a hypergraph are *labeled* and *directed*. Our definition however, encompasses more graph-based hypertext systems and corresponds to the general definition of a *hypergraph* as a general logical graph-based representation of a hypertext.

Place	**Object**	**Place** **Transition**	**Node**
Edge	**Relation**	**Arc**	**Link**
HyperGraph	**Semantic Net**	**Petri Net**	**Hypertext**

Figure 3.1: Equivalent objects in models

3.2.1 Semantic Networks

A hypergraph can embed *attributes* associated with hypertext objects such as keywords and access rights. The structure is then referred to as *attributed hypergraph.* These attributes can form a record structure that describes the hypertext objects. Attributes can also be embedded in links. They are mostly used to describe relations and automatically created versions. An attributed hypergraph with labeled directed links is a *semantic network.* The knowledge representation is better captured by the explicit labeling of links in semantic nets. The structures, however, are similar, as is shown in figure 3.1. The use of semantic networks in hypertext is usually confined to the intuitive graphical representation. Semantic network hypergraph models and systems [49] [1] [28] [8] go beyond graphical representation and exploit semantic aggregations and generalizations to provide inheritance, instantiations and other logical mechanisms.

3.2.2 Finite State Automata

Hypergraph links can support external routines so as to provide the hypertext with procedural power. The structure is then referred to as an *augmented transition network,* which has the ability to represent any computable algorithm. The hypergraph structure $< N, L, E >$ with labeled directed edges, E, is itself a deterministic

finite state automaton (FSA) [93]. The essential feature of the deterministic FSA approach is that the *current state* of the hypergraph can be determined, and the *next states* deduced. This particularity imposes a *navigation mechanism* the pure hypergraph/semantic net approach lacks. The FSA formalism, however, does not capture the *sequence* of events, conditions, or *flow control* mechanisms well. FSAs and their dual, *marked graphs*, have very high decision power, but restricted modeling power since alternative activities cannot be modeled. FSAs can be considered as subclasses of *Petri nets* [79].

3.2.3 Petri Net Based Hypertexts

Petri nets [79] [68] are an abstract formal model of information flow. A Petri net consists of places and transitions connected by directed arcs. As such, it models the static properties of a hypergraph (see figure 3.1). Additionally, the *execution* [79] of a Petri net provides dynamic properties that extend beyond the static graph's properties, and that describe asynchronous and concurrent activities. The execution of a Petri net is controlled by the positions and movements of *tokens*. At each instant, the resulting *marking* describes the status of the Petri net. Petri nets model two aspects of a system in particular: *events* and *conditions*, and the relationship among them. Furuta [90] has mapped Petri nets to hypertext. Places represent hypertext nodes and transitions map to links. The execution of the Petri net models the navigation of the hypertext. The firing mechanism that models conditions and events extends classical hypertext navigation by providing a *browsing semantic*. The browsing semantic helps in reducing disorientation and cognitive overhead. More important in the context of this thesis, it allows inference to be modeled (see section 5.5).

3.3 Hypertext Structures

As portrayed in figure 3.2, a hypertext is composed of entities, functions that manipulate these entities, and properties that result from the functionality and data structure.

3.3.1 Entities

The hypertext entities are the data objects used in modeling and presenting hypertexts. The modeling entities are the hypertext components that form the elements of the hypergraph. The presentation objects consist of navigational aids and display widgets.

Nodes

Hypertext nodes can either be structured recipients of information [98, 8], or containers that encapsulate it. Nodes may embed links, explicitly as fields, or implicitly as control sequences and bitmasks. They may also embed attributes, which can serve purposes such as database management facilities (see 2.5.1), component formats and structures (see 2.7.1) and specialized semantics (see 2.4.1).

Links

Hypertext links can be *abstract, hot, warm, dynamic* or *cold*. Abstract links are independent of the context of the document and can be compared to embedded menus. Hot links are system links that get automatically generated and updated. Links to versions in [37] are of this kind. Warm links provide a "communication channel" between documents [100, 98] used to pass information. Dynamic links provide a "browsing semantic" mechanism: their activation depends on the hypertext status

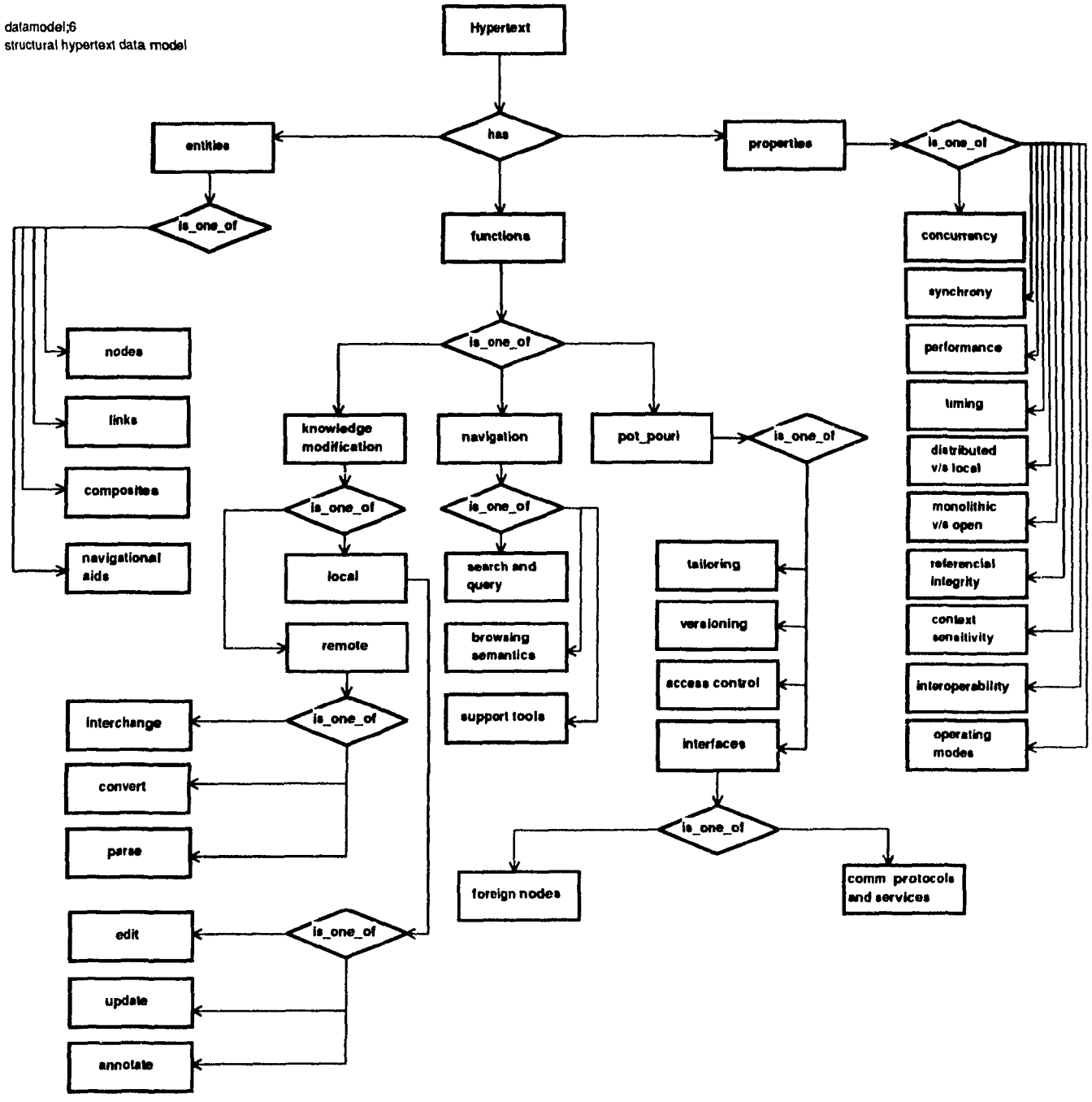

Figure 3.2: The structure of a hypertext system

[90]. Links also can embed attributes used in filtering and querying mechanisms [37], specialized semantics [28], formats and structures.

Composites

Composite hypertext objects embed paths, tours, webs[2] and networks. Essentially, their function is to reduce disorientation and cognitive overhead: paths allow backtracking to previously encountered nodes. Predefined paths form tours that linearize the hypertext and target specific applications such as tutorials for example. Webs and networks constitute a clustering of basic entities and attributes. Webs address a specific context, whereas networks reflect a structural associative nature.

Navigational Aids

Navigational aids consist of classical orientation aids - such as those found in books [3] [10] - that provide a strong sense of context, together with specific hypertext network maps in many possible formats (graphical browser, fisheye, etc...). The hypertext specific tools reflect the problems inherent in hypertext: graphical browsers with no filtering mechanisms become unreadable, unless some hierarchy and clustering [42] provides classification. Graphical browsers also carry problems inherent in graph layout: automatic layout algorithms are not generally able to capture the graph semantics, and very few make use of the current layout information, creating a user orientation loss [14]; the dimensions of the graph can be impossible to visualize all at once, or might require "clues" for their 2D projection on a screen [22].

[2] Recall that a *web* [100] is a network that represents abstract concepts: its links and the nodes they relate are meaningful for a given concept. It is similar to database views.

[3] headers, footers, page numbers, chapter headings, bookmarks, thumb marks, etc...

3.3.2 Functions

The hypertext functional aspects can be categorized [77] into knowledge modification, navigation and general system functions.

Navigation

In *browsing mode,* hypertext systems are only "navigated": users access documents either directly through a searching or querying mechanism, or through links from other documents. Classical search and query mechanisms suffer from the *context problem* [96] in hypertext as much as in any information retrieval system [6]. The context problem can be expressed as the difficulty of determining the relevance of a document to a query given keywords alone. Some techniques however, exploit hypertext characteristics [47] to refine the relevance of indexes through voting methods, or in building *hyperindices* [18]. Querying can be specified graphically [13] [40] ; or make use of hypertext structures [7] [31], or models [59] [1] [61]. Browsing mechanisms include *history* for backtracking, *filters* based on attributes to reduce cognitive overhead, *progressive disclosure* to describe what to expect at the other end of the link, bookmarks and thumb marks to mark the "information landscape". Navigation can also be supported by *scripting facilities* [101], triggering of actions (*daemons*) [21] and *procedural attachements* as in augmented transition net architectures.

Knowledge Modification

Knowledge modification consists of any editing, either of the content or the structure of the hypertext. The knowledge modification functions consist of editing, updating, annotating and acquiring knowledge. These should support cut and paste metaphors, along with structural edits of the network itself. Knowledge acquisition should allow both interchange and integration of information. Although

no actual hypertext interchange format exists [4], the SGML [71] language seems to be the best candidate and the Dexter model itself served as a reference for an interchange format. Integration, on the other hand, covers a broad spectrum of techniques. The two main approaches are in artificial intelligence aids and pure algorithmic and database approaches. AI techniques tend to merge knowledge bases and hypertext on the common ground of semantic nets [97]. Some neural net approaches [11] try to "recognize" documents from keywords. They are sought as a viable alternative to pure statistical algorithmic techniques. Database approaches rely heavily on attributes and are mostly used in specialized hypertexts [92].

General System Functions

General hypertext system functions encompass all functions not specific to hypertext, that are nevertheless essential functions of it: interfaces, access control, versioning and tailoring. Interfaces to external programs are achieved in augmented transition networks, and serve specialized applications that require specific processing power [37]. Versioning is an essential activity by which changes are recorded [91] [55] in many hypertext applications that need to keep track of modifications [12]. Access control [37] [90] is imperative in all hypertexts that extend beyond a single user manipulation. Furthermore, hypertext paths can embed nodes with different access permissions, but the denial of access should not "cut-off" users from other unrestricted documents. Access and version control are usually supported through attributes. Beyond casual software tailoring, hypertext tailoring extends to the access of information and its sequencing.

3.3.3 Properties

The main properties that differentiate between hypertexts are:

[4] The statement might be outdated at the time of submission

- operating mode: what forms of authoring and browsing are supported?
- inter-operability: are authoring and browsing performed in the same mode?
- concurrency: how many users, paths and documents can be simultaneously opened?
- formalization: does a formal model exist or is the system developed for a specific application?
- timing: does the system support dynamic documents?
- context sensitivity: do operations depend on the hypertext context?
- referential integrity: are dangling links permitted?

3.4 The Dexter Model

The key feature of the Dexter[5] model is in the spectrum of the existing hypertexts it encompasses[6], and of their respective authors, who participated in the workshops The goal of the Dexter model is to provide a reference basis for comparing hypertext systems, and to develop interchange and inter-operability standards. Though quite new, this model has already been used in the standardization of a hypertext interchange format.

3.4.1 A Layered Architecture

The Dexter model has a three layer architecture, as depicted in figure 3.3. The *runtime layer* handles the presentation mechanisms supporting the user's interaction with a hypertext. The *storage layer* describes the hypertext network of nodes

[5]named after the motel in which the first workshop was held in October 1988

[6]Augment, Document Examiner, EDS, FRESS, Intermedia, Hypercard, Hyperties, KMS/ZOG, Neptune/HAM, NoteCards, Textnet, etc...

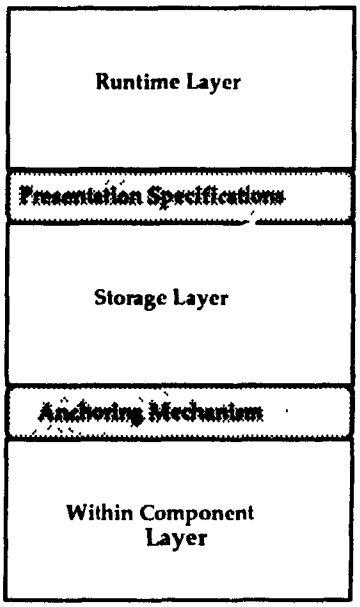

Figure 3.3: The layered architecture of the Dexter model

and links along with their management. The *within-component layer* covers the actual structure and contents of the hypertext nodes. *Presentation specifications* interface the runtime and storage layers; an *anchoring* mechanism maintains a clear separation between the storage and within-component layers.

3.4.2 The Storage Layer

The storage layer is the core of the Dexter model. It focuses on the mechanisms by which the hypertext components are associated and managed so as to form a network. The storage layer does not differentiate between component types: the component itself is a data "capsule". Its internal structure is to be dealt with by the within-component layer. This approach allows the extension of the model beyond "pure text": it encompasses different media without framing them into a rigid structure which is not adhered to by any existing hypertext system. The

storage layer consists of a set of components together with two primitive functions: a *resolver* and an *accessor*.

Data Structures

The *component* is the basic addressable entity in the hypertext. As described in the entity-relationship (E-R) diagram of figure 3.4[7], a hypertext consists of a set of components, each of which has a *unique identifier (uid)* assumed to be universally uniquely assigned, a *base-component* and a *component-information* module. The base-component can be one of an *atom* (commonly called a node), a *link*, or a *composite* component made up of base-components. Composite components are restricted to be acyclic graphs such that they cannot contain themselves, directly or indirectly. The links tie different components together. Each link has a sequence of *specifiers*. Modelling n-arity is then possible. Link endpoints are specified by an *anchor id* and a *component specification*. The *direction* specification models directional links: it indicates whether the specified endpoint is to be considered a source link, a destination link, both, or neither.

Also part of the link specifications is the *presentation specification* that is part of the interface between the storage and runtime layers. The anchor-id and presentation-specification data types are shared with the *component-information* part of a component. This way, composites and atoms can also possess presentation specifications to be conveyed to the runtime layer. The component-information part allows the description of component semantic properties (defined by each hypertext system) through a set of *attribute, value* pairs. It also allows the definition of a set of *anchor-id, value* pairs to be used by the anchoring mechanism that interfaces the storage and within-component layers. The anchor-id *aid* is unique with respect to the component. Its arbitrary value *av* can represent a region, a location, an

[7]all data flow diagrams, structured charts and entity relationship diagrams have been created using Teamworks 4.01. The use of this case tool allowed the validation of the translation of the formal model from the Z language formulation to the usual graphical representations.

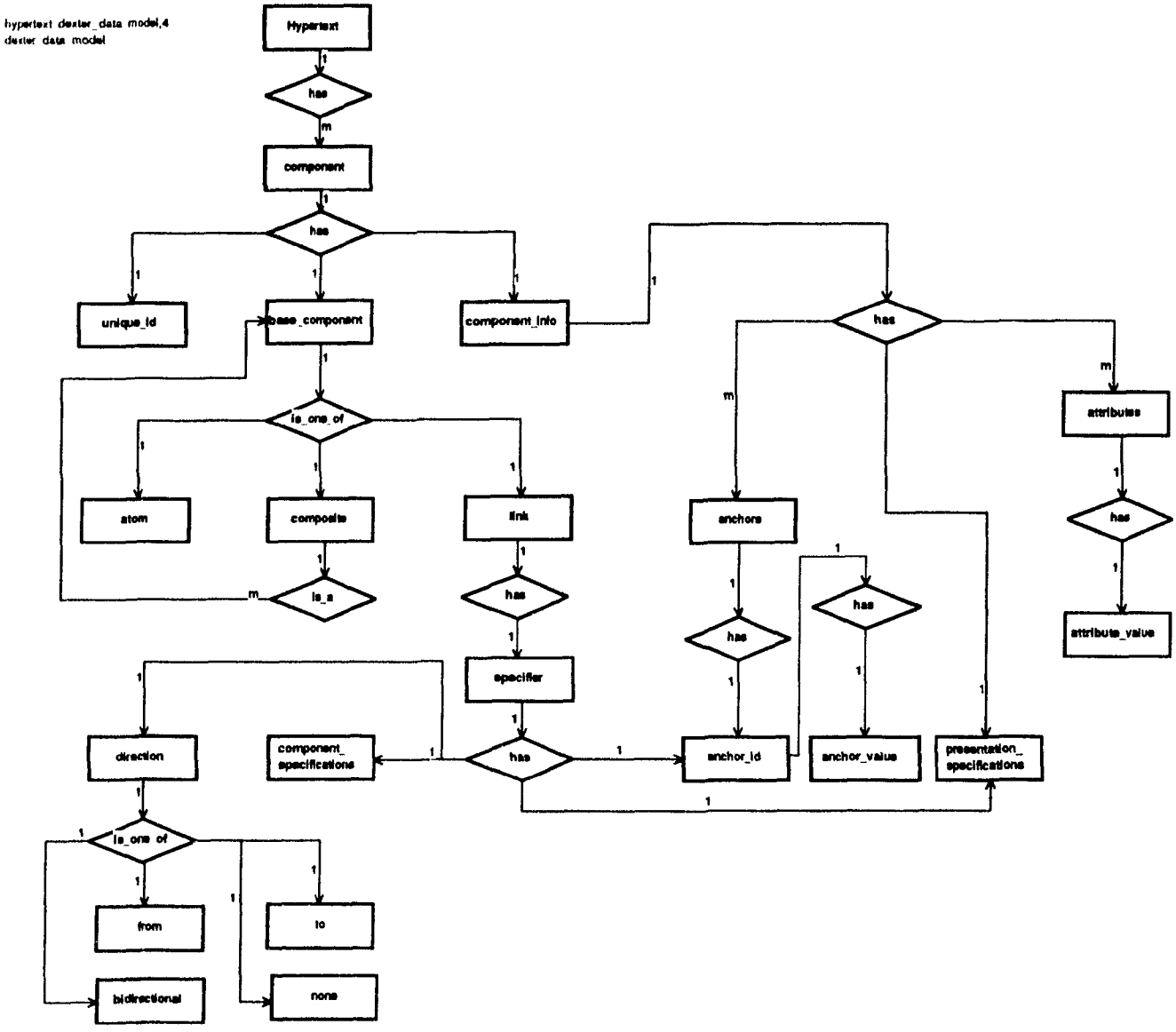

Figure 3.4: The storage layer data structure E-R diagram

item, a substructure or any other entity within a document. Its interpretation is up to the application responsible for handling the contents and structure within a component. The application is thus responsible for the within-component layer.

The Storage Layer Primitives

The storage layer has two primitive functions that together represent the whole mechanism of mapping specifications into the components themselves (i.e. retrieving the components): the *accessor* and the *resolver* functions. The accessor function accesses the component given its *uid* (i.e. unique identifier) in a direct addressing scheme, whereas the resolver function returns a *uid* from a component specification *cs*. The coupling of both functions provides an indirect addressing scheme. Often the component specification can only consist of the *uid* itself, in which case the resolver is a simple identity function.

The Storage Layer Functions

The storage layer functions deal with the creation, modification, deletion and retrieval of components. Additionally, the storage layer also covers attributes and anchors specific functions.

Atoms, links and composites can be created using specific functions. These functions all rely on the *createComponent* function described in the structured diagram in figure 3.5. All the specific creation functions are packaged as cases into a unique function *CreateNewComponent* that will be invoked from the runtime layer.

DeleteComponent and *ModifyComponent* are also called from the runtime layer functions *deleteComponent* and *realizeEdits* (in figure 3.6) respectively. *DeleteComponent* ensures that any link whose specifiers resolve to the given component is also eliminated. *ModifyComponent* requires that the component type remains the same and that the corresponding hypertext remains link-consistent.

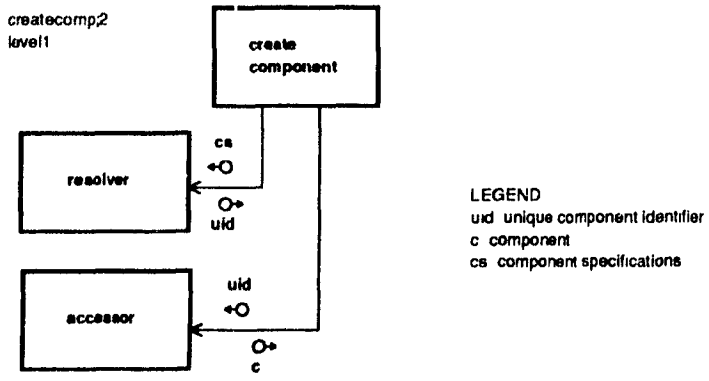

Figure 3.5: The *createComponent* structure chart

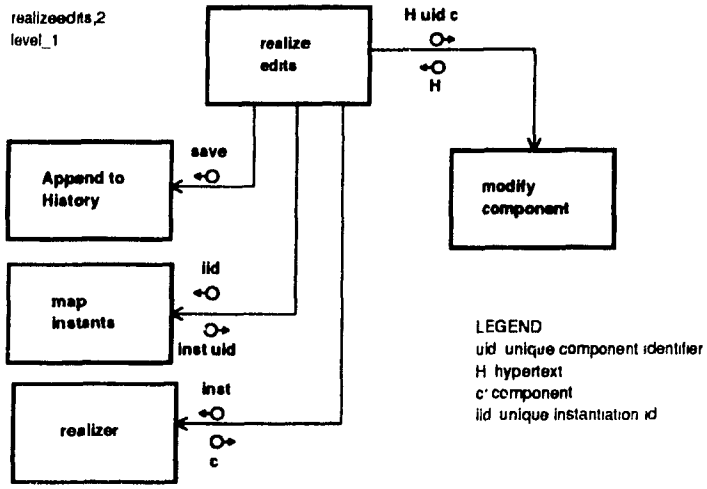

Figure 3.6: The *realizeEdits* structure chart

Retrieving a component given a *uid* simply consists of the *accessor* primitive. If the component is a link, other functions will return either a source or a destination for that component.

Attribute functions allow the querying and setting of the attribute values of a given component. They also allow the querying of all the component attributes. *LinksToAnchor* allows the determination of all the links associated with a given anchor.

All storage functions are invisible to the user, thus creating a clear division with the runtime layer. Some of these functions (those with a capitalized first letter, the attributes and anchor functions) can themselves be called from runtime functions.

3.4.3 The Runtime Layer

The runtime layer provides tools for accessing, viewing and manipulating the hypertext network structure. The objective of the Dexter model is not to formalize a user interface, but rather to provide the essential functions for browsing and authoring a hypertext. Again, since the runtime layer is left open to different implementations, a mechanism is needed for interfacing with the storage layer. The presentation specifications encoded in the storage layer provide the information about how the component is to be presented. The advantage of such a technique is that the presentation of a component is no longer solely dependent on the hypertext tool: it is also a property of the component itself (in the case of a node) or of the access to a document (in the case of a link).

Data Structures

The key idea of the runtime layer is that components are *instantiated* on a display (i.e. presented to the user). Thus, it is a component "clone" that gets viewed and edited. The actual component will not be altered unless edits are explicitly saved. Furthermore, a component can have multiple instantiations, each being assigned an *instantiation identifier*. The instantiation data structure is depicted in the E-R diagram of figure 3.7. An instantiation has a unique identifier, the *instantiation id*, a *base* and a sequence of *links* with their corresponding *link anchors*. The structure of the instantiation is a "shadow" of the essential features of a component with respect to the runtime operations.

A hypertext *session* then consists of a hypertext, together with a dynamic set of *instants* and a *history* of the operations performed during the session. Three functions are also part of the session schema: an *instantiator* that will return instantiations of components from their unique identifier and presentation specifications; a *realizer* that will perform a "write" operation, and a *run-time resolver* that will ex-

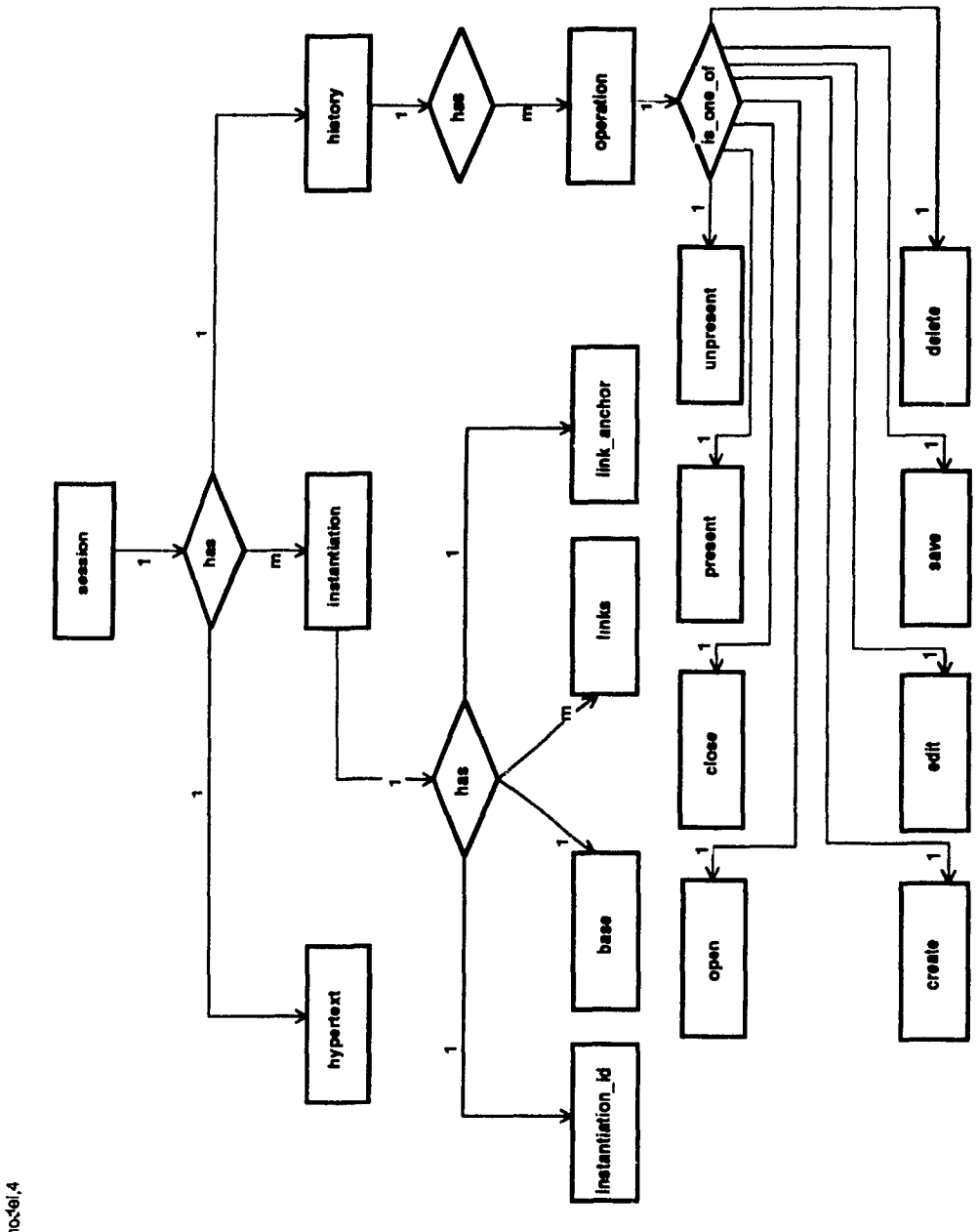

Figure 3.7: E-R diagrams of the runtime layer data structure

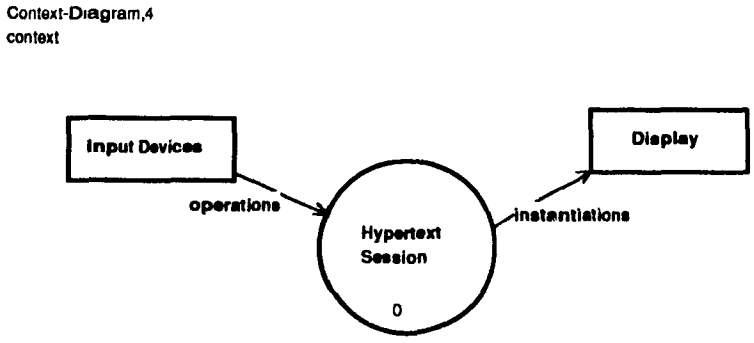

Figure 3.8: Context diagram of a hypertext session

tend the resolver operations to runtime (e.g. the "last node visited" cannot be resolved within the essentially static storage layer).

A Hypertext Session

The operations of the runtime layer are best described through a hypothetical hypertext session. This session in turn is described through a set of data flow diagrams and structure charts at the adequate level of granularity.

A hypertext session consists of operations performed by a user on hypertext objects, the instantiations of which are displayed through an independent user interface (see figure 3.8). The user first *opens* a session, thus opening a specified hypertext, initializing the instantiations set, and writing the operation name (i.e. "open") to the history file. It is then possible to *present* a component, the instantiation of which gets displayed. In a browsing mode, simple *follow link* operations will allow the navigation of the hypertext. In the authoring mode, it is possible to *edit*, *realize* the edits (i.e. actually update the component with the edits made to the instantiation), *delete* and *create* new components. In both modes, instantiations of the components can be *unpresented* (i.e. removed from the list of instantiations and from the display). Once all required operations have been performed, the session can be *closed* (see figure 3.9).

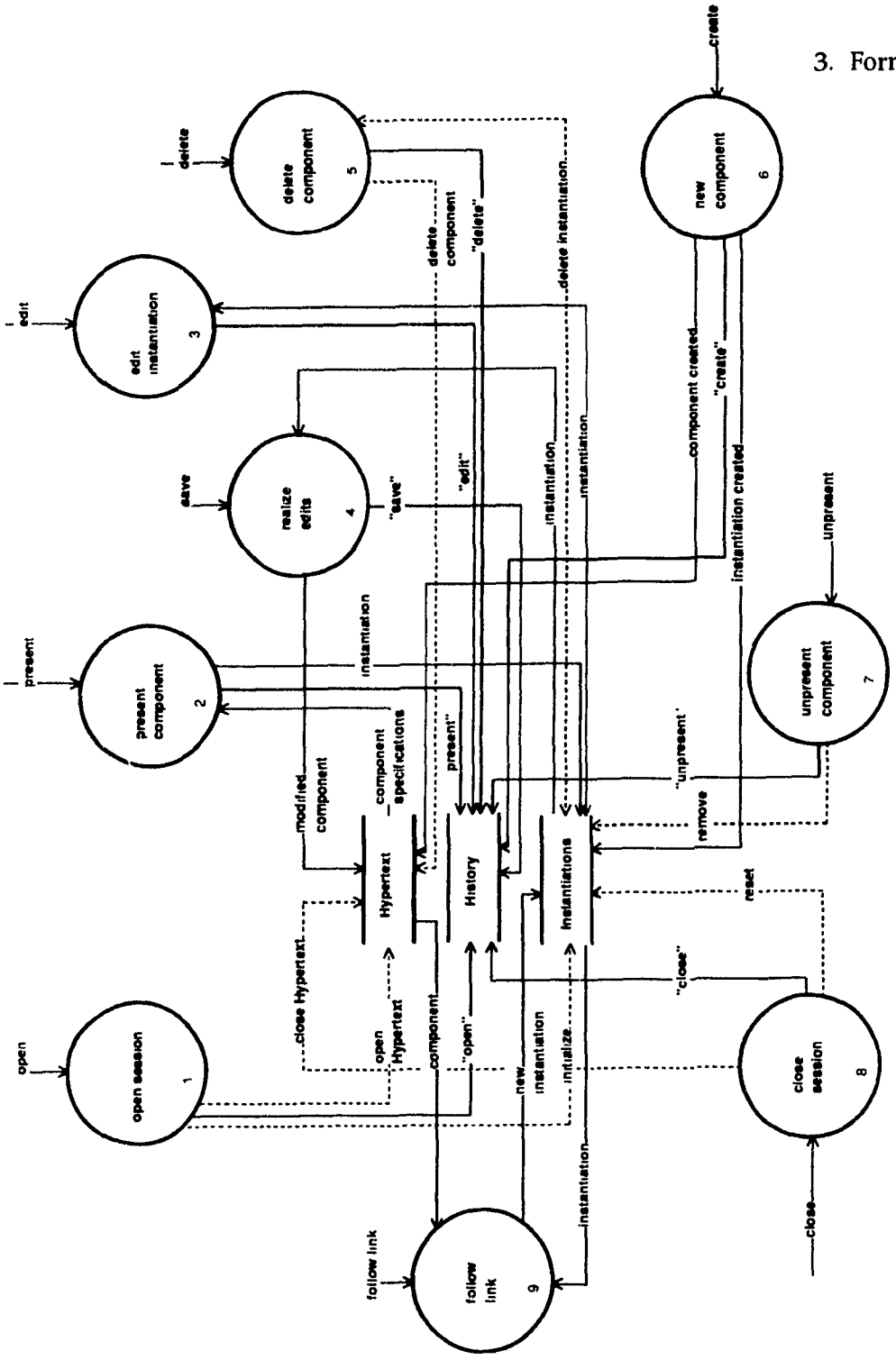

Figure 3.9: Data flow diagram of the runtime operations

3.5 Conclusion

The Dexter model is the formal representation that covers the broadest spectrum of hypertext systems. Through the specifications and anchoring interfaces, the Dexter model captures the essentials of the hypertext structure while still remaining a completely open architecture. This type of architecture can be referred to as *static* (see section 1.3.1): no inference or processing based on the *status* of the hypertext is performed. These aspects will be covered in the next chapter.

Chapter 4 Intelligent Hypertext

4.1 Introduction

Static hypertext architectures are not a form of artificial intelligence [76]. They do support the property of connectivity among information chunks that contain facts and concepts, but the resulting semantic structure is not exploited by any inference mechanism.

Many hypertext systems, however, use artificial intelligence techniques as aids to structure and to navigate a cognitive model of the domain of information. Some even try to exploit the hypermedia features and use them within knowledge engineering tools by interfacing them to expert systems. Very few attempts have been made to integrate hypermedia to expert systems.

4.2 Intelligent Hypertext Tools

Many hypertext tools try to achieve *knowledge structuring* and enhance *information retrieval*. Some of these do use AI techniques. Though the aim might not be to achieve intelligent systems, the application of inferencing techniques to hypertext structures lays down the principles for intelligent hypertext architectures.

4.2.1 Knowledge Structuring

Knowledge structuring in hypertext is called *authoring* and is traditionally the task of the hypertext writer/programmer. As hypertexts grow in size, the task of

structuring the whole body of information needs to be assisted by computer tools. Classical syntactic techniques now compete with hypertext tailored ones. These are used both for knowledge structuring and retrieval.

4.2.2 Knowledge Representation: Semantic Nets

Knowledge Acquisition in hypertext relies heavily on the use of semantic nets. Although the ultimate goal is to overcome the natural language processing problem, most tools use a semantic net representation that permits some inferencing on the concepts being captured. [49] has developed mechanisms that map the hypertext information structures into first-order logic formulae.

Semantic Net Abstractions in Hypertext

A hypertext H is a set $H =< I_0, l_0, \pi, A >$ [49], where l_0 is the set of *information objects* that can be *instantiations* of *primitive objects* in the set I_0. π is a set of predicates that characterizes objects and the relationships between them. A is a set of attributes that identify and describe objects from different perspectives Abstractions can then be defined in terms of the structures just determined. An *aggregation* is the mechanism by which a collection of objects can be referenced by an identifier; and a *generalization* allows a collection of objects to be referred to by a generic object which captures their common features Versioning can also be modeled semantically as a *revision*. The semantic abstraction structures and mechanisms have been developed by [49]. [1] practically mapped them to traditional hypertext objects. Nodes, links and their graphical correspondents are defined as predicates; procedural attachments are defined as properties of these predicates. Documents are modeled as *structured nodes*, each part of which is a *slot*. The document structure then matches a *template* or *frame* structure.

Automatic Knowledge Acquisition

In hypertext, automatic knowledge acquisition has been solely based on the semantic net approach. The process [97] consists of several stages: *parsing*, determination of *dominant concepts*, aggregation of these concepts into *topic descriptions*, and generalization of the topic descriptions into *text graphs* [1]. Frame building and slot filling result from parsing, and constitute the basic activities behind the determination of dominant concepts. Clustering uses predicate logic based on the newly created semantic network performs the aggregation into topic descriptions.

In effect, a knowledge base is built, that captures information in the form of a semantic net. The efficiency of such systems, however, is still to be demonstrated.

Retrieval and Browsing Using Semantic Nets

Once a semantic net exists, intelligent tools can take advantage of the elaborated structure to assist in information retrieval and navigation. The semantic net can either be *specific* (i.e. extracted from the information body using automatic knowledge acquisition, as described above) or *general* (i.e. "imposed" by a classification that takes into account the given document). Accordingly, the retrieval tools differ in their approach. Beside semantic browsing, most elaborate semantic retrieval techniques in hypertext use one form of querying or another. The querying mechanism can be a part or whole of the retrieval mechanism. Moreover, querying can be classified into *structured queries* and *content queries*.

Structured Queries

Structured querying mechanisms use the structural description provided by the semantic net and determine relevant documents. The expression of the query itself

[1] Text graphs are *hierarchical* subsets of semantic nets.

can have many formats. It can be expressed graphically [31], in modal logic [7], in predicate logic [1], as much as it can resemble casual database queries. It can also use hypertext *views* and indices [18]; One drawback is that they take no advantage from the content of the information body. Accordingly, the matching mechanisms are limited by the predicates and attributes syntax.

Inference Queries

Most inference queries combine one form of natural language processing with the thesaurus structure that results from the embedded semantic net of a knowledge base. The knowledge-based approach allows inferencing on the goals of the user and a duplication of the reasoning of an expert. The stated goals generate sub-goals and related goals not explicitly expressed by the user but of possible relevance. The modeling of the expert knowledge allows the expansion of the user queries and their fitting within the actual semantic structure. Furthermore, inference queries allow the retrieval process which is inherently uncertain to be better approximated Bayesian logic [34], non-classical logic [96], fuzzy logic [59] and rule-based [61] retrieval models have all been used in hypertext. Few hypertext models [59] and systems [27] use inference queries because of the difficulties of integrating hypermedia and knowledge-based features into intelligent systems. As an alternative, non-semantic intelligent retrieval tools are being investigated.

Other Intelligent Tools

Non-semantic intelligent tools exploit the acyclic hypergraph structure to help determine the relevance of documents in retrieval using *adaptive* methods such as neural networks [11], and the correspondence between documents in navigation using *affinity* criteria [81]. Note that these can also be used as refinements in semantic-based tools.

4.3 Intelligent Systems

Hypermedia is also thought of as a tool that can be integrated with databases and expert systems in order to build *intelligent systems* [15]. Such systems are characterized by their ability to use knowledge to solve problems; and by the capacity they have to exploit the powers of association and inference needed for complex problems. As such, they should behave logically (i.e. in the expert system sense); be able to make efficient use of existing information; and provide non-linear and adaptive navigation. Furthermore, their complexity and that of their operation modes requires high levels of user-friendliness and interactivity.

Intelligent systems are designed around specialized technologies. Database techniques and tools are used to efficiently manage knowledge in the form of structured information. Expert systems then add a new layer of functionality by "leveraging" the system with induction techniques that provide extended significance and usefulness. Hypermedia enhances expert systems by providing a tool for the management of large bodies of information and knowledge, irrespective of their existing form. These bodies can have incompatible formats (e.g. different structures or media) and need to be tied together. On the other hand, expert systems assist hypermedia in modeling intelligent hypermedia engines that enhance, but are not restricted to, information retrieval and navigational aid. The expert system component makes use of the structure of the embedded knowledge to assist in retrieval or reduce disorientation in navigation. Specialized expert systems can also be attached to hypermedia nodes. They provide added *local inference* on the hypermedia information body.

Since no single technology integrates all these functions, the separate systems either "hook" to each other through "external functions" or share common data repositories. These techniques are attractive because they provide several advantages. First, quick prototypes can be implemented and evaluated. Second, data acquisition is almost always already available in the form of databases. In spite of

this, few integrated prototype systems have been "polished" enough and put to the test. However, there is little binding between the hypermedia and the expert system at the structural level; as a result, the hypermedia component often duplicates part of the expert system inferencing as in [36], when it is not just limited to a simple user-interface tool.

4.4 Intelligent Architectures

Intelligent architectures have been developed from one or other of the two technologies, either by integration of one system into the other, or by interfacing two stand-alone packages. Few approaches, however, have attempted to merge them into one intelligent system.

4.4.1 Interfacing

The most common architecture model is that of interfacing, as depicted in figure 4.1. The interfacing component usually consists of mapping the external functions feature of both modules. Communication channels are then created through which data is interchanged. The interchanged data consists of information (e.g on-line documentation, entry values, etc...) and of status information useful to the other module. In this architecture, the hypermedia component can serve a double-function: on-line documentation and user interface. Often, hypermedia systems with an augmented transition network feature serve as the interface itself

Hypermedia Interfaces

Interfacing can extend beyond intelligent functions and provide support for collaborative work, bulletin boards, electronic mail, access to on-line libraries and

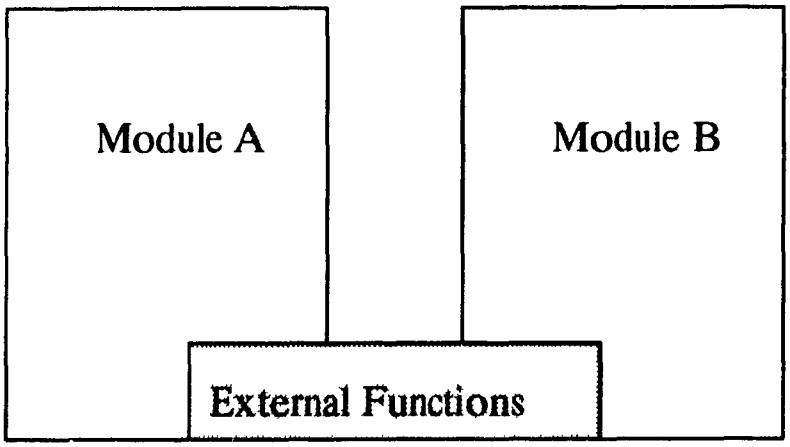

Figure 4.1: The interfacing model

specialized systems [25, 47, 5]

Hypermedia interfaces can provide several advantages:

- an intuitive manipulation of the user-interface that addresses an audience larger than computer literates only;
- greater cohesion between separate modules;
- a coherent user-interface with unified functions;
- a personalized environment; and most important:
- a personal workspace.

Personalized Environments Personalized environments result from the authoring feature hypermedia provides. They provide tailoring beyond the user-interface environment, and result in the notion of *incremental* intelligent systems, where the throughput increases with the usage and the users. One example is the use of user feedback in building belief networks in the index terms of a query [47]. The weight propagation allows a fine-tuning of the queries usually performed by a given set of users.

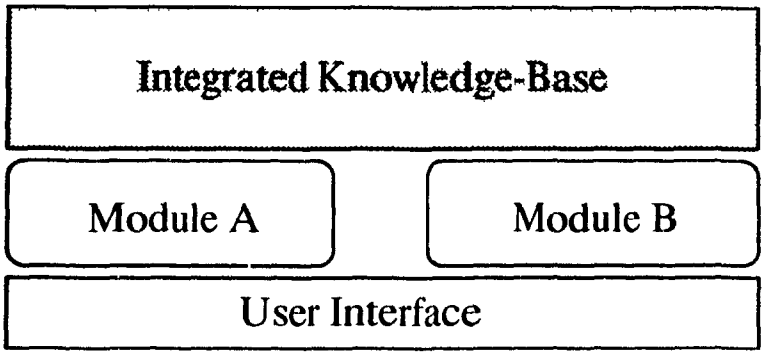

Figure 4.2: The integration model

Personal Workspaces The notion of a personal workspace is supported by the "edit realization" concept detailed in the Dexter model in chapter 3.4. Users can retrieve a set of information, create their own versions, edit and test using the system features, and copy back the information to the shared information-base. Before integration, the manipulated data and the operations performed on it represent a *context* [37].

One possible approach is to use argumentative methods (see 2.4.1) and infer on these. An alternate approach consists of the integration of the knowledge-base and the information-base in hypermedia structure.

4.4.2 Integrated Systems

Integration architectures try to merge the information and knowledge representations, as depicted in figure 4.2. The concept is similar to that of blackboard architectures [73, 74]. The essential feature is the representation and presentation of the information, both at the level of problem specification and solution explanation. The two main approaches in integrated systems consist of using inference in argumentative methods and in knowledge elicitation.

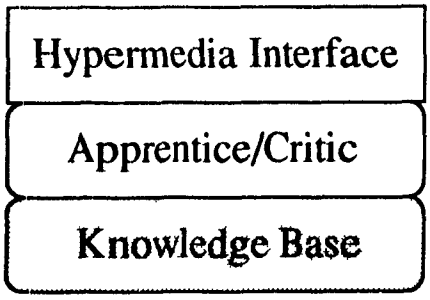

Figure 4.3: The argumentation model

Argumentative Architectures

Argumentative methods [86] were first implemented in issue-based hypertexts [8] (see 2.4.1) and have been used in intelligent systems 4.3 that use design apprentice/critics editors. The design apprentices can typically only provide brief and awkward explanations that fail to provide the complex argumentative background behind their critiques. These critiques include assumptions, conditions and controversial design issues that can be structured into an issue-based semantic net. They provide personally meaningful abstractions to the user, and thus eliminate computer-generated explanations as found in expert systems. The net effect is to reduce the difference between problem-oriented and system-oriented descriptions.

Argumentative architectures [44, 63, 43] (see figure 4.3) are based on the semantic nets such as those that the IBIS [86] and PHI [62] argumentation methods create. The restricted set of derived relations allows a mapping from the knowledge-base system to the issue-based semantic net. The knowledge-based system is usually built around state-driven condition-action rules which are triggered by the apprentice/critique inference engine when a non-satisfactory solution is detected. The corresponding hypermedia information is then accessed and the related argument is made available. The apprentice/critique has thus provided an "intelligent" entry point within the hypertext.

Next, authoring capabilities founded on the issue-based semantic network [63] are needed. Full integration of the issue-based semantic net and the knowledge

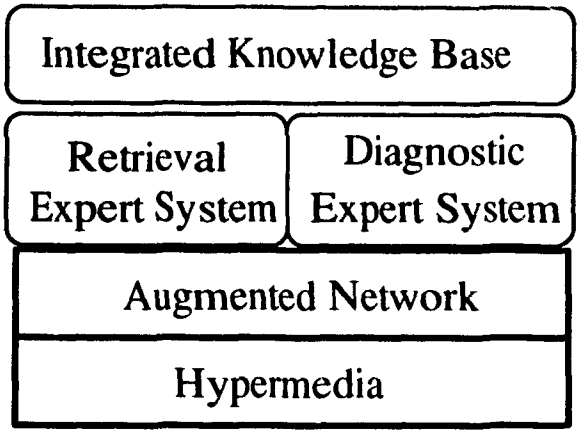

Figure 4.4: The smartbook architecture

base, however, should allow inferencing based on the knowledge just acquired, which in turn implies an inverse mapping from the issue-based semantic net to the knowledge base.

Smartbook Architectures

The *smartbook* architecture [51] (see figure 4.4) tries to answer the needs of large complex documentation systems integrated with appropriate field expertise. Expert systems are developed alongside hypermedia on-line documentation nodes, in an augmented transition network architecture. The tight coupling of expert systems to the corresponding hypermedia information provides an intelligent system in which the hypermedia structure allows easy extension and integration. The hypermedia information-base, however, remains separate from the knowledge base, even if some automatic knowledge acquisition can be achieved for some procedural documentation.

The interest in smartbook architectures goes beyond the incremental approach to knowledge and lies essentially in the knowledge elicitation for diagnostic expert systems. Classical diagnostic expert systems rely on a set of questions to assert a diagnostic/solution. The smartbook architecture uses the descriptive power of hypermedia together with a clear navigation structure to determine the diagnostic.

Once the fault is determined, the local expert system is accessed and a set of rules and procedures determine the repair . At any instant, supportive documentation is available in hypermedia format. Knowledge integration in smartbook architectures remains however at the conceptual level.

4.4.3 Knowledge Modeling Systems

Knowledge modeling hypermedia systems are intelligent systems specialized in the support of the process of knowledge acquisition. They represent integration tools for the storage, structuring and maintenance of different sources of knowledge. At the same time, they provide a collaborative platform for knowledge transfer between knowledge engineers and experts, together with personal workspaces.

The intelligence of this kind of architecture is currently very limited, and, to our knowledge, no inferencing has been implemented on the resulting structures. They are worth mentioning, however, because they reflect a knowledge acquisition methodology that represents one aspect of the intelligent hypermedia architecture proposed in the next chapters.

4.5 Conclusion

Intelligent systems do provide the "appearance" of intelligence, and loosely map hypermedia information pools to knowledge bases. Till now, the mapping has been unidirectional, and the developed systems have failed to provide inference *from* the hypermedia structure. These lacunae reflect at the level of architectures. Interfacing and integration have been implemented, but no true unification of the two architectures has been attempted. This is what we will attempt to develop in part 2.

Chapter 5 A Unified Model

5.1 Introduction

The incentive behind a unified model is to provide a knowledge authoring/consulting environment within a symmetric architecture. The unified model consists of modular toolsets integrated in a tightly-coupled architecture The functionalities of the integrated tools provide a framework for knowledge manipulation. The integration and synchronization of these tools provide functionalities that extend beyond the added functionality of each module alone (i.e. synergism occurs). The functionality of this model will be covered in this chapter. The structural and behavioral aspects of the model will be described in the forthcoming chapters.

5.2 General Architecture

The general architecture of the unified model is depicted in figure 5.1. The *kernel* is the core of the model. It models the information base and automatically reflects the modifications and inferences performed by any peripheral module. Each of the peripheral modules provides a different view of the information base, and a different processing approach; yet they can all be dynamically manipulated at once, providing a "leverage" over their own specificities. For example, a fact asserted in a running expert system will correspond to a token inserted at the corresponding place in the Petri net module . Should the place (and hence the fact) correspond to a hypermedia document, this document could be automatically displayed in the hypermedia module. Inversely, links activated from within the hypermedia

module would instantly fire a transition in the Petri net module. Should the link also correspond to a rule in the expert system, it would be triggered and the resulting actions would be inferred. The hyperstructure of the kernel should then be able to model the static structures used in the different modules and to synchronize their dynamics.

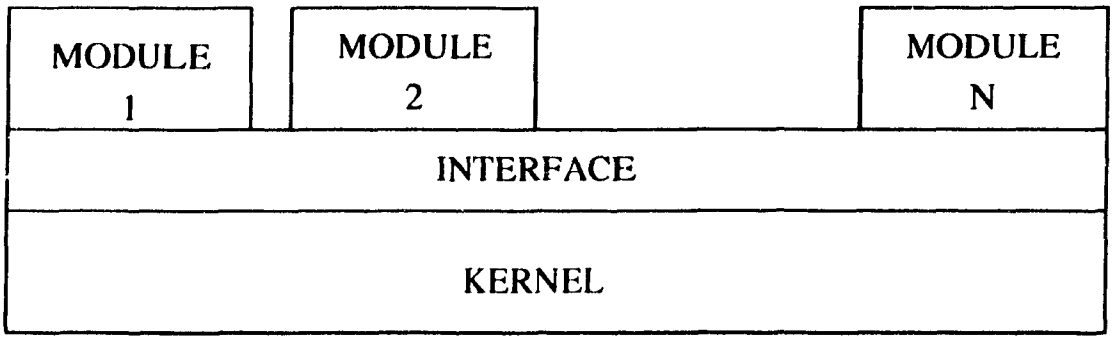

Figure 5.1: general architecture

5.3 Fundamental Functions

The unified model is to support, but is not restricted to, the following functions:

- multiple media real-time support in distributed environments;
- various information access strategies;
- multiple knowledge processing techniques;
- knowledge elicitation and acquisition;
- validation and analysis.

The above functions are distributed over several modules. Two of these modules will be detailed throughout these chapters. They are the hypermedia engine and the expert system engine. Other obvious engines could be validation engines, object-oriented databases, intelligent retrieval engines, intelligent tutorials, etc ....

As will be shown, the open architecture of the model does not restrict the number or diversity of modules. The only restriction is that their data structures should all be modeled using the kernel *hyperstructure*.

5.4 Leverage By Integration

Multimedia support allows for different sources of information to be integrated in a single computer *medium*. The hypermedia authoring support provides a cognitive modeling of the information conveyed by the media; and the support for browsing allows intuitive user-friendly access to this information. Embedded knowledge can be inferred by a *production system* that integrates procedural and declarative knowledge processing. Knowledge can be elicited from the browsing of the hypermedia. The unified hyperstructure representation of the kernel enables the automatic building of the corresponding knowledge objects in the rule-base. Conversely, expert systems programming offers an attractive declarative language for intelligent hypermedia navigation. An *evolutive* knowledge concept follows from both the authoring/browsing interoperability and the unified knowledge elicitation mechanism. It allows incremental information and knowledge integration, thus producing a system that *evolves* with time and use, unlike most actual expert systems. The complexity resulting from multiple media, integration, and knowledge base maintenance, requires analysis techniques that maintain and validate the hyperstructure. In this general context, collaborative work using issue-based methodologies and multi-user access would augment the system capabilities.

But first, equivalence mappings should be built between the different representations. In the following sections, equivalence mappings will be built between hypermedia, Petri nets and expert systems. The resulting synergetic will be demonstrated with the use of an example. Additionally, support for dynamic documents will be modeled using the Petri net representation. An example will illustrate the modeling of multimedia nodes that can be embedded as dynamic documents

within the previous example.

5.5 Equivalence Mappings

Mappings are to be described, that will demonstrate the *global* equivalence in behavior between a hypermedia, a Petri net, and an expert system. The Petri net mapping is necessary for modeling timed events in hypermedia, as will be demonstrated. Petri nets can also be useful in modeling and validating both the hypermedia and the expert system module. Furthermore, using the Petri net as an intermediate representation simplifies the building of a mapping between the hypermedia and expert system modules.

5.5.1 The Petri Net Model

A *marked Petri net structure* [79] is a five-tuple, $N =< S, T, I, O, M >$in which S is a finite set of *places*, T a finite set of *transitions*, I an *input function* that defines for each transition the set of *input places*, O an *output function* that defines for each transition the set of *output places*, and a marking M which is a *vector* that gives the number of tokens in each place of the Petri net Executing a Petri net results in a sequence of markings, beginning with an initial marking μ_0, and ending with a final marking M_f. Going from one marking to the next is performed by firing any transition that is enabled under the current marking. Firing a transition results in tokens being removed from places incident with transitions and added to the places that follow these same transitions.

5.5.2 Mapping Petri Nets to Hypermedia

The model of a *Petri net based hypermedia* (PNBH) was developed by [90]. A $PNBH$ is a sextuple $H =< N, D, W, B, P_l, P_d >$, in which beside the Petri net N defined above, D is a set of *documents*, W a set of *windows*, B a set of *buttons*, P_l a *logical projection* of the hypermedia documents and links onto Petri net places and transitions, and P_d a *display projection* of the same document. The logical projection P_l provides the mapping between the Petri net structure and the $< D, W, B >$ hypermedia structure. The display projection P_d is a collection of mappings that associates the logical elements in $< W, B >$ to the front-end user-interface widgets of the hypermedia.

5.5.3 Refining the Mapping

The refinement of the $PNBH$ model is required to keep the mapping to the expert system model comprehensible. Indeed, since the two representations are not *exactly* equivalent, a direct mapping to the $PNBH$ would result in meaningless hypermedia nodes that would correspond to assertions needed for inferencing at the expert system level. Note that the "adjustment" of representations is not required under the hyperstructure, as will be demonstrated in the next chapter

Let $H =< N, D, W, B, P_l, P_d >$ be a $PNBH$ as defined above.

We refine the definition of D to be $D =< D_i, D_a, D_p >$, $D_i \cap D_a \cap D_p = \phi$, where D_i is the set of *invisible documents*, D_a the set of *active documents*, and D_p the set of *passive documents*. An invisible document is a document that does not appear under the display projection P_d. An active document is a document from which the user will make or collect an inference. A passive document is a descriptive document that provides further information on the contents of displayable documents. Furthermore, we define D_n as the subset $\{D_i, D_a\}$ of D.

Similarly, we refine the definition of B to $B =< B_p, B_a, B_i >$, $B_p \cap B_a \cap B_i = \phi$, where B_p is the set of buttons that link to passive documents D_p, B_a is the set of buttons that link to active documents D_a, and B_i is the set of invisible buttons used in internal inferencing and that can link to any document D. B_i buttons cannot have manual user-triggered firings since they are invisible. Furthermore, we define B_n as the subset $\{B_i, B_a\}$ of B.

Observe that documents D_i and buttons B_i are not mapped to the display under P_d.

5.5.4 Mapping Petri Nets to Expert Systems

Mapping logic clauses to high-level Petri nets has been studied by [67]. A high-level net can be considered as a structurally folded version of a regular Petri net. For the purpose of the discussion, and unlike [67], we are going to elaborate a mapping between *simple* Petri nets and propositional clauses such as those found in rule-based syntax.

Definitions

Let a basic Petri net structure be a triple $< P, t, Q >$ where $P = p_1, \ldots, p_n$, $Q = q_1, \ldots, q_m$ and t the transition.

Let a basic rule be a triple $< C, S, A >$ where $C = c_1, \ldots, c_n$ are *conditions*, $A = a_1, \ldots, a_m$ are *actions*, and S is an *inference structure*.

Let $F =< C, A >$ be the set of facts in an expert system ES.

Let k_l be a mapping from the set of basic rules to the set of basic Petri nets which maps c_i onto p_i, a_j onto q_j, and s onto t (see figure 5.2). Observe that k_l is a one-to-one mapping and denote the inverse mapping by k_l^{-1}.

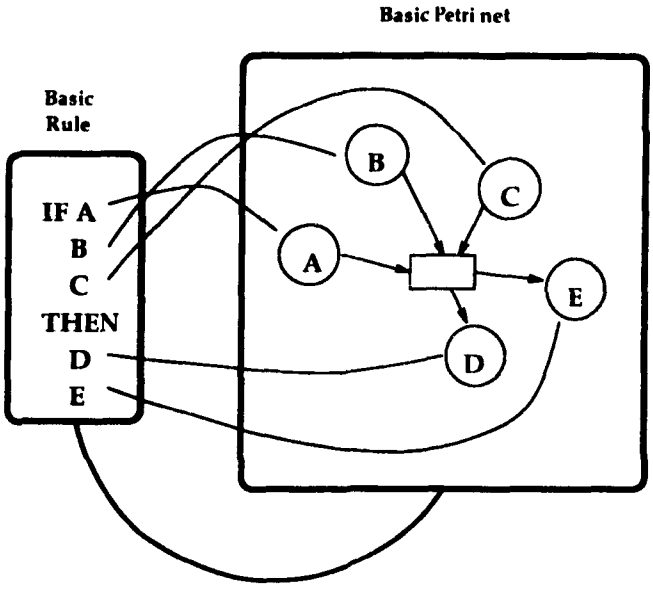

Figure 5.2: The k_l mapping

A Petri net, N, may be viewed as a set of basic Petri nets connected in a weak order fashion.

Similarly, an expert system, ES, may be viewed as a set of basic rules connected in a weak order fashion through an inference mechanism

We extend the mapping of k_l to a mapping K_l between expert systems and Petri nets in the natural way (see figure 5.3). Observe that K_l^{-1} can be extended in the same way.

Under this projection, the facts and rules of the expert system are modeled in a Petri net. Inference is modeled as a set of markings where transition firings modify the state of the search space. An assertion corresponds to a token in the corresponding place. The retraction of a fact corresponds to the removal of the token from the corresponding place. As such, the browsing semantics would not accurately mimic the inference mechanism: a token would be removed from a place upon firing the corresponding transition, but the fact need not be retracted from the knowledge-base. Accordingly, the mapping k_l accounts for each place in P to be a place in Q unless a retraction of an antecedent fact is specified in the consequent

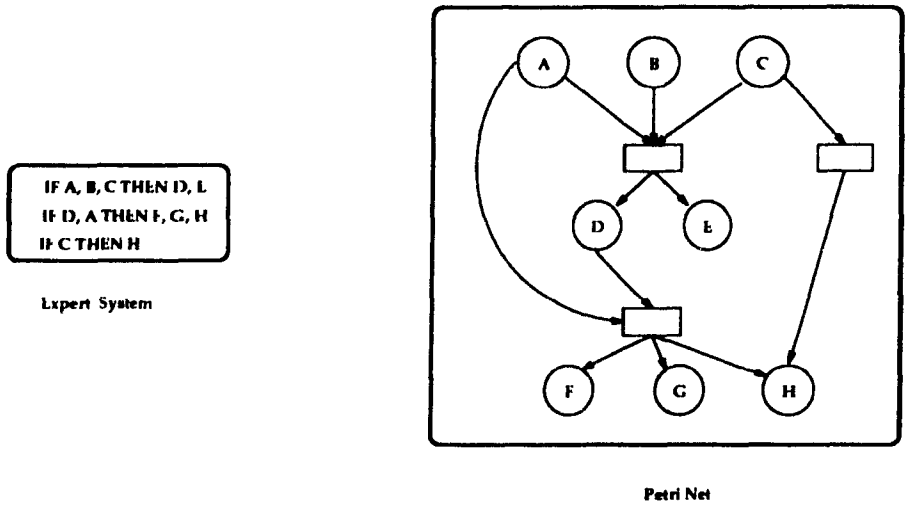

Figure 5.3: extending k_l to K_l

part of the corresponding rule. Observe that all retractions can be modeled in this fashion[1].

5.5.5 Modeling Equivalence

Recall that K_l maps the expert system to the Petri net and that P_l maps the Petri net to the hypermedia. Let $K = K_l \circ P_l$. K then represents the mapping from the expert system to the hypermedia. Similarly, denote by K^{-1} the inverse mapping $K^{-1} = P_l^{-1} \circ K_l^{-1}$. The display projection P_d still maps the internal representation of the hypermedia to the interface widgets.

Under the K mapping, an expert system ES can be modeled into a hypermedia H, as depicted in figure 5.4.

F maps to D_v under K. The invisible documents D_i correspond to facts that do not require documentation. They, for instance, can represent internal states used in the inference mechanism. Observe that the passive documents D_p are not

[1] In our approach, transitions are instantaneous or manual. Thus, the facts status does not change. Furthermore, only one transition can be fired at a time. Thus, no more than one token can reside in one place.

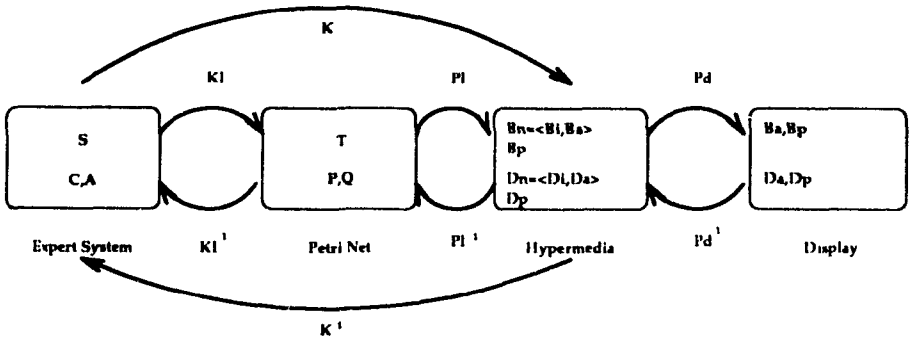

Figure 5.4: Mapping expert systems to hypermedia

mapped to ES. They only provide further information that extends and broadens the knowledge documented in the active documents D_a.

Similarly, S maps only to the B_n buttons and to the D_n documents associated to them. The B_p passive buttons are hypermedia links to passive documents D_p, and thus have no mapping under K.

The inference structure S of ES becomes, then, the browsing semantics of H, and thus provides an intelligent navigation mechanism.

Under the inverse mapping K^{-1}, active and invisible documents in D_n are mapped onto facts in F. The B_n buttons together with the D_n documents represent the inference structure S of the expert system ES.

The equivalence in behavior can be described in a three-layer architecture as depicted in figure 5.5. The three layers are made up of an acquisition layer, an integration layer, and a representation one. Vertically, each column corresponds to a domain: expert systems, Petri nets and hypermedia. The representation layer is the heart of the equivalence mapping. All the operations performed in the upper layers end up in modifications at the representation level. The interfacing between the different domains consists of the same mapping as that described above and depicted in figure 5.4. The integration layer is concerned with the integration of knowledge into the existing representation in the three domains. It consists of integrating rules and facts with an existing expert system, integrating Petri net

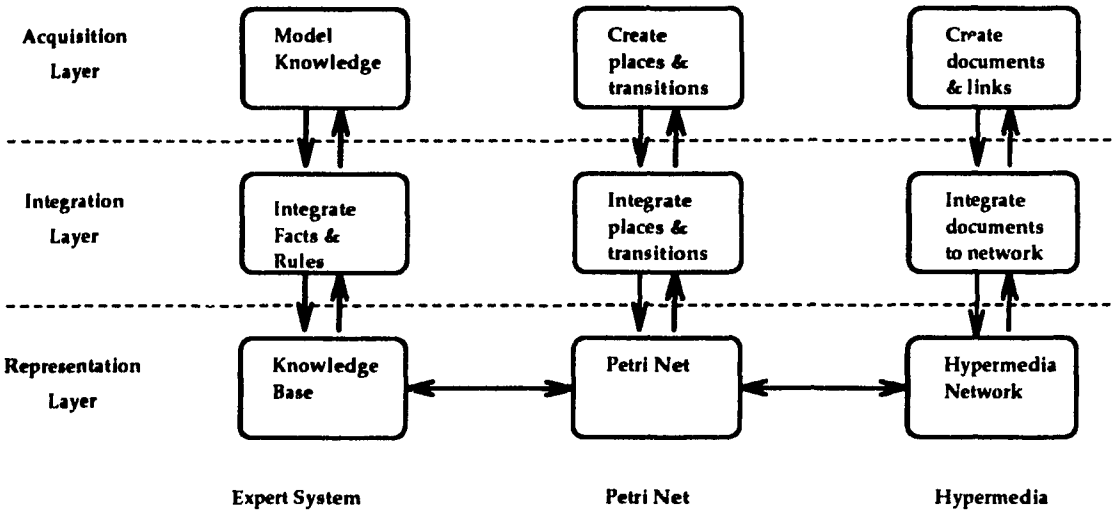

Figure 5.5: The unified model

places and transitions with the current Petri net network, and integrating new documents and links with the actual hypermedia network. The acquisition layer describes the modeling functions that are to be integrated at the integration level. One can notice that there is no perfect one-to-one mapping between the hypermedia structure and browsing semantics and the expert system rules, facts and inference. Some hypermedia nodes and links have no equivalent in the expert system, and vice-versa. The overall behavior and knowledge, however, are similar. This will be illustrated in the example below.

5.5.6 An Example

Consider an intelligent system concerned with assisting with car repairs. The system does not target a specific audience, and accommodates the knowledge evolution of the user. Accordingly, it embeds hypermedia "tours" of the car in full-motion video with a separate sound track [2] and dynamic links (i.e. in time and space). The system also embeds a fault-diagnosis expert system that diagnoses faults based on the user description of the fault. The expert system can be

[2] one can easily imagine a video sequence with both French and English narrations for example.

manipulated on a stand-alone basis, or coupled to the hypermedia features. In the second case, the hypermedia browsing exhibits intelligent navigation semantics, and guides the user into successive steps, from the illustration of possible symptoms to description of repair procedures. As such, the system provides on-line documentation in different media, that can be accessed and processed in different ways.

Intelligent Navigation Derived

The sample expert system can determine repairs and adjustments based on the state of the engine, its components, and on some symptoms the engine might exhibit. A sample set of rules that detects the need for timing adjustments could be (in pseudocode):

```
symptoms rule:
if (engine unsatisfactory) then (check symptoms)
misfiring rule:
if (car backfires) then (misfiring engine)
timing problem rule:
if (engine unsatisfactory)
and (misfiring engine) then (timing adjustments)
```

The corresponding equivalent representation is depicted in figure 5.6. This static representation is automatically acquired [3] when the diagnosis program is loaded within the expert system module. The kernel representation can also serve for other modules, such as the hypermedia one. The hyperstructure is then mapped to the hypermedia structure, places coinciding with documents, and transitions with links. The resulting hypermedia structure then looks like figure 5.6.

[3] Algorithms have been implemented and used with a prototype hypertext [90]. The algorithms are listed in appendix A.

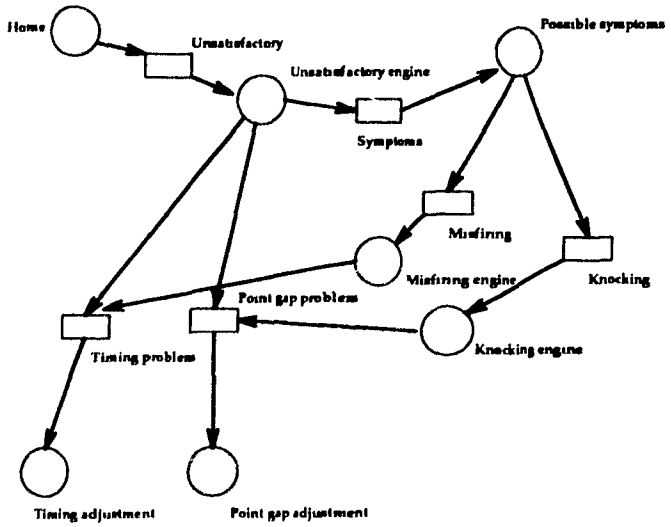

Figure 5.6: The fault-diagnosis hypermedia

Document *Home* is a data-gathering node where information on the state of the engine is acquired. The static document explains in detail each button option. Should the user choose the *Unsatisfactory* option for example, the *Normal* and *Does not start* buttons would be deactivated and a new document *Unsatisfactory engine* would appear that would offer new button options along with, say, a video sequence introducing a speaker, and a setting with a car. At the level of the browsing semantics, the token would leave document *Home* upon the activation of the *Unsatisfactory* button, and appear in node *Unsatisfactory engine*, thus automatically launching the video sequence. Two buttons *Timing problem* and *Point gap problem* provide hints to possible adjustments. Their activation, however, is dependent on facts not yet determined and they are thus disabled. The third button *Symptoms* calls for further information on some engine symptoms. Upon activating this button, a new document *Possible symptoms* appears as a video sequence that describes engine misfiring and knocking symptoms, together with their appropriate temporal links. Should the user click on the *misfiring engine* button, the browsing semantics of the hypermedia would enable the *timing problem* link that corresponded to the diagnosed fault. The *timing adjustment* node can then be visited. This node can represent a starting point for on-line documentation for timing adjustments, an automatic repair process, or represent a sub-layer of further intelligent expert system

coupled hypermedia, using the hierarchical structure previously described. The hypermedia structure and execution semantics acquired from the hyperstructure are an indirect translation of the rules of the expert system. The resulting browsing semantics have reduced the disorientation by enabling only those links for which prerequisite material has been covered. For example, the *timing problem* link from the *unsatisfactory engine* node could only be enabled after the relevant symptomatic documentation on misfiring engines has been covered.

Unformatted Knowledge Support For Expert Systems

The hypermedia module could also serve as a front-end to the expert system module, and provide additional support to unformatted information. Knowledge could then be inferred from the hypermedia itself. Back to our example, suppose the user is within the *Possible Symptoms* document, faced with a description of misfiring and knocking engines. Should the user decide that one of these applies to his situation, and click, for example, on *Knocking engine*, a token would be inserted in the relevant place in the hyperstructure. The expert system would then be notified of this change of status in the hyperstructure, and the *(misfiring engine)* fact would be asserted. The *(engine unsatisfactory)* fact being previously asserted the same way, the inference engine of the expert system would trigger the *timing problem* rule. The *(timing adjustment)* fact would then be asserted, and the fault diagnosed. On the other hand, when the information is manipulated from the expert system interface, asserted (retracted) facts also modify the hyperstructure The kernel then notifies the hypermedia module of which corresponding documents have been enabled (disabled). In our example, should an inference determine that a point gap adjustment is needed, the corresponding document would pop-up on the hypermedia interface even though it was not accessed through any hypermedia path. The integration also allows users to assert conditions that could only be represented by unformatted documents (e.g. an x-ray image).

5.5.7 Knowledge Elicitation

Beyond mere consistency in the manipulation of the two modules, integration also provides a platform for knowledge elicitation. The merged authoring/browsing functionality of the hypermedia provides a user-friendly knowledge elicitation tool. The automatic declarative coding that results from the alteration of the hyperstructure requires no special programming skills. It could often remove the need for knowledge transfer from the expert to the programmer. This can be best described using an example.

Suppose now that, in our previous example, the user decides to upgrade the knowledge-base with a new adjustment, *New adjustment*, that corresponds to a new symptom, *New symptom*. The first step consists of creating the new document *New adjustment*, and its corresponding link, *New problem*, and to link them to the *Unsatisfactory engine* document in the current hypermedia network. Similarly, a *New fault* document should be created, and linked to *Possible symptoms* through a new link, *New symptom*. The new hypermedia network is depicted in figure 5.7. The resulting browsing semantics for the *New problem* button are identical to the ones for *Timing problem* and *Point gap problem*. The documents with new button options should be upgraded so as to provide the relevant information associated with the new alternative. Further on-line information on the adjustment can also be added to the hypermedia structure. In this example, the expert system module automatically generates a set of rules corresponding to the added diagnosis. At that point, knowledge acquisition of the new diagnosis is complete.

5.6 Validation and Analysis

Due to the variety of knowledge representations and manipulations, validation is required at several levels within the unified model. First, at the level of the modules themselves, it verifies numerous properties. Second, at the level of the integration,

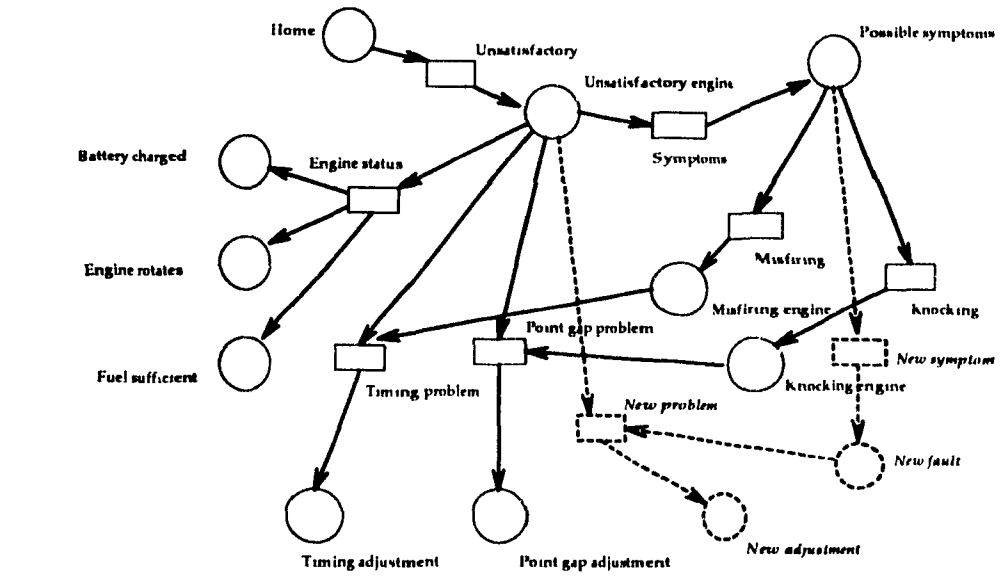

Figure 5.7: Adding a diagnosis

Property	Concept	Interpretations
Boundedness	# of tokens in p < k	A given document could be accessed by a maximum of k-1 users
Conservation	# of tokens in net < k	A maximum of k-1 documents can be simultaneously accessed
Liveness	No deadlock	sharing resources, security access
Reachability	Is marking M reachable?	Is document C accessible?
Coverability	from marking N?	Is document C1 accessible from document C2?
Persistence	Does firing t1 inhibits t2?	Would triggering button B1 inhibit button B2?

Figure 5.8: Some behavioral properties of Petri nets

it verifies homogeneity and coherence.

Validation within the unified model is greatly facilitated by the Petri net structure under the equivalence mappings. Analysis techniques of Petri nets and their properties can contribute to the validation of the hyperstructure. Furthermore, the Petri net properties can be equivalent to some module specific ones (see figure 5.8) The hyperstructure then provides a unique structure through which the different modules can be validated and analyzed. Analysis can be classified in terms of reachability, matrix equations and reduction techniques.

5.6.1 Reachability

The reachability graph allows the determination of all possible marking states from an initial marking μ_0. Considering the fault-diagnosis example of figure 5.6, one can verify that the *Timing adjustment* document can be reached from the *Home* document. Furthermore, the reachability graph will determine what intermediate markings are necessary to reach the document, hence creating a path to that document.

Within the integrated approach, the *Timing adjustment* document might have been triggered from a corresponding assertion in the expert system. The reachability graph then provides all *supportive* documentation through the path it determines, providing a cognitively structured hypermedia equivalent to the "why" answer of the expert system. At the level of the expert system, reachability provides a demonstration that search space states are reachable, and what the "inference thread" to each state is. At the level of real-time multimedia applications, reachability provides the temporal analysis tool of synchronous hypermedia [48].

5.6.2 Matrix Equations and Reduction Techniques

Matrix equations and reduction techniques are also useful analysis tools. Matrix equations are a powerful representation of the dynamic behavior of Petri nets, but contain certain weaknesses that make them difficult to apply in some cases [68, 79]. As for reduction techniques, they provide simple operations [66] for reducing large Petri nets, while preserving their properties. Reduction techniques can extend beyond Petri net analysis and can be used to cluster nodes in hypermedia graphical browsers, or in simplifying rule-based representation.

5.7 Supporting Multimedia Data

While equivalence mappings have demonstrated an underlying similarity the hyperstructure will exploit, no dynamic media support was modeled. Support of multiple media imposes specific functional requirements on the data structure:

- real-time support for dynamic media (e.g. video, audio, animation);
- concurrency and synchronization of the different media
- synchronized manipulation of concurrent activities and coordination of their accessing mechanisms.

The equivalence mapping should be able to support the above features. The next section will demonstrate what properties of the underlying Petri net model are relevant in the support of multiple media.

To be able to support multimedia data, timing characteristics are to be introduced. These characteristics can be added to the *PNBH* by extending the Petri net model as to include for timing properties. This will be detailed below, and an example will be used to demonstrate the mechanisms.

Timed Petri Nets

A *timed* Petri net is a quadruple $N =< S, T, I, O, \tau >$ where $< S, T, I, O >$ constitute the ordinary Petri net, and $\tau T :\rightarrow \{0,1,2,\cdots\} \times \{\infty, 0, 1, 2, \cdots\}$ is a function mapping each transition to a triple $(\tau_t^r, \tau_t^m, \tau_t^a)$ termed *release time, maximum latency,* and *maximum availability* respectively [48]; such that $\forall t \in I, \tau(t) = (\tau_t^r, \tau_t^m)$ and $(\tau_t^r \leq \tau_t^m \leq \tau_t^a)$. The release time is the minimum time that must elapse before the transition can be fired, and the maximum latency is the maximum time that can elapse before automatic firing of the transition. Such nets have been widely investigated [32, 64, 84]. The application of Petri nets to timed hypertexts [48] has been

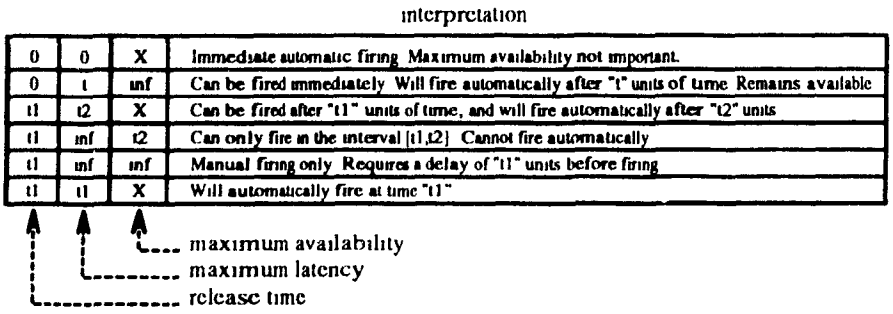

Figure 5.9: Combining timing attributes

extended here so as to provide a more elaborate dynamic temporal specification. We have added an *availability time* limit τ_t^a that allows the building of a temporal interval for the hypermedia link.

Usage

The combination of the three timing parameters provides temporal characteristics that naturally extend to places: a place is said to be *active* as long as it has a token in it, (i.e. within the interval $[\tau_t', \tau_t'']$). Release time and maximum availability provide lower and upper bounds to its *liveness*. Maximum latency provides a default automatic firing. Several combinations and their interpretations are detailed in figure 5.9. Timings and priorities can be easily modeled using these characteristics. Multimedia applications, among other real-time systems, can be modeled using the resulting properties of timed Petri nets.

In our sample application, video and sound sequences should be synchronized, and temporal links should be accessible at specific locations and time intervals. A typical presentation would have the timings of figure 5.10, where the video and sound sequences extend from t_1 to t_6; and links *link1* and *link2* respectively provide access to a static document (such as text or graphics) and another multimedia sequence. The presentation and accessing of the documents satisfies our definition of *true hypermedia* (see 1.3.3).

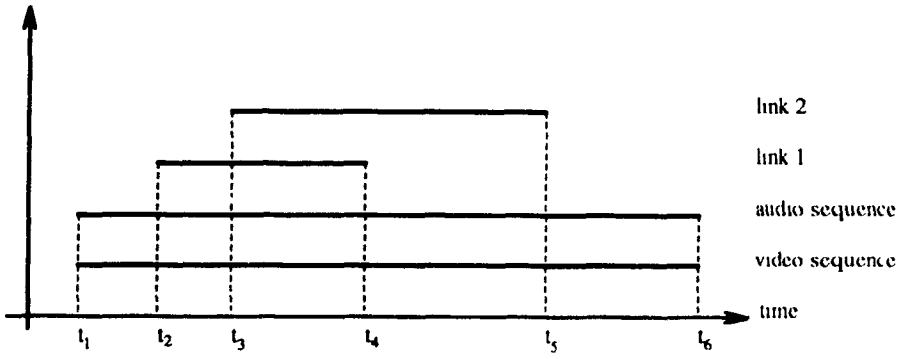

Figure 5.10: The timing sequence of multimedia events

The corresponding timed Petri net is represented in figure 5.11. Recall that Petri net places can consist of procedural attachments and other processes. The separation of structure from content allows the modeling of the video and audio sequences as events in Petri net places The start and end of the multimedia presentation are modeled as transitions *start* and *end*. Supposing that the presentation is to be manually started, and that it should start instantaneously, the corresponding timing attributes of link *start* are $(0, \infty, \infty)$: the link should be instantaneously available, should not be fired automatically, and is always valid The *end* link, on the other hand, corresponds either to an *end-of-sequence* or an *interruption* event of the multimedia presentation. Accordingly, it should be immediately operational and automatically fired after a maximum latency of $t_6 - t_1$ time units corresponding to the end of the sequence. Its timing attributes are then $(0, t_6 \quad t_1, \infty)$

Hypermedia access

The timing diagram of figure 5.9 shows that the static document accessed through *link1* can only be available in the $[t_2, t_4]$ time interval. It could, for example, provide technical details about an object that appears in the video sequence during that interval. *link1* can then be available only after $t_2 - t_1$, and for a period up to t_4. As for default firing, it is not required here. The timing attributes of *link1* are then $(t_2 - t_1, \infty, t_4 - t_1)$. The annotative document should not replace the video sequence, but rather superpose in an overlapping window. As for the sound sequence, it

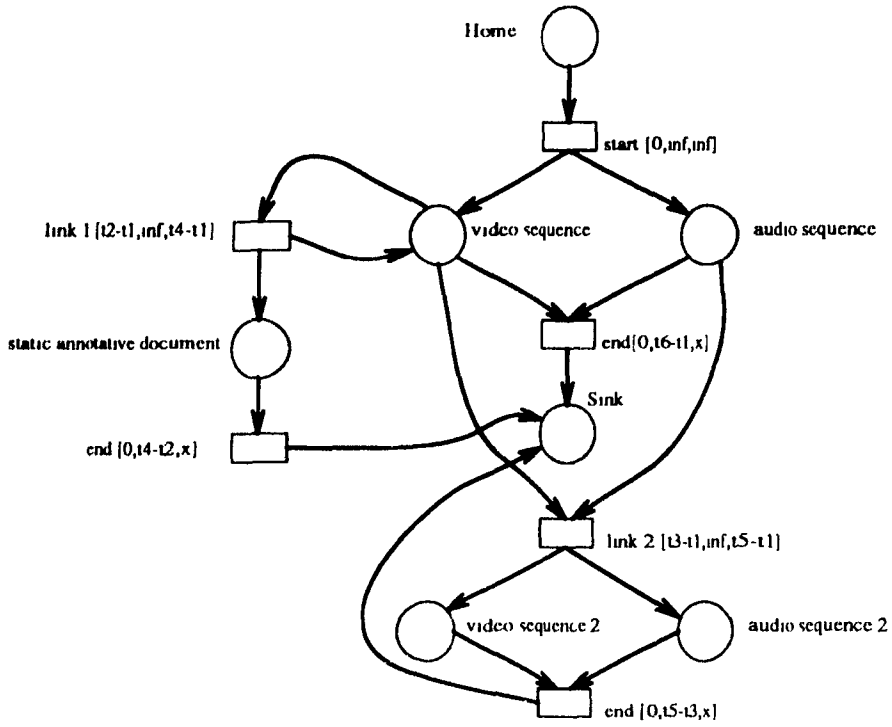

Figure 5.11: The Petri net model of the hypermedia

should not be affected by the selection of *link1*. The modeling of these execution semantics is illustrated in figure 5.11. A self-loop between the video sequence and *link1* allows the continuation of the video sequence while the annotative link is present, while there is no arc linking the sound sequence to *link1*. Note that the annotative document can be *killed* any time between t_2 and t_4 by the activation of the corresponding *end* link. The timing characteristics of the corresponding *end* link are then $(0, t_4 - t_2, \infty)$.

The second document is accessible through the link *link2*. Unlike the annotative one, this document consists of another dynamic presentation (e.g. another video sequence). Figure 5.10 shows that it is active in $[t_3, t_5]$. The execution semantics for *link2* are different from those of *link1*, as depicted in figure 5.11. The structure of the second sequence, though, is identical to that of the first sequence: it starts by firing *link2* that will, in turn, activate two multimedia processes. These processes will be interrupted either manually by the user, or automatically, through the maximum latency timing. The readability of the graph can be enhanced by using hierarchical

Petri nets. Under these, the whole process could be reduced to a place that would be labeled "multimedia sequence". The resulting place could then be part of an upper layer representation of a hypermedia diagnosis system, as depicted in figure 5.6.

5.8 Conclusion

In this chapter, the principal functionalities of the unified model have been enumerated. Equivalence mappings have been built in order to show that synergism is possible. Examples have illustrated the synergism. A Petri net intermediate representation has proved capable of integrating the two representations, and of modeling the synchronization of their similar behaviors Furthermore, the Petri net could model the hypermedia dynamics, and provide interesting analysis and simulation properties. The hyperstructure of the unified model, however, should not be restricted to a Petri net structure Rather, it should model the equivalence mappings described above in an object-oriented framework, and account for their imperfections. Moreover, it should support further integration of modular tools This will be detailed in the next chapter.

Chapter 6 The Architecture of the Unified Model

6.1 Introduction

This chapter will present the structural aspects of the unified model. As expected from the functional descriptions of the model, the architecture should be *symmetric* in order to allow for the synchronization of events among the different modules. The cement of the model is the hyperstructure that ties together the data structures of the different modules. Its object-oriented nature should allow for a growing structure in an open architecture. The corresponding object methods should bring integrity and consistency among the operations of the module. Moreover, inheritance and aggregation should lay the ground for the interaction between the different parts of the system. As before, the examples will focus on the hypermedia, Petri net and expert system modules.

6.2 A Symmetric Architecture

The intelligent system architectures detailed in chapter three all exhibit the same master-slave architecture. Accordingly, the master application takes advantage of the slave characteristics: expert systems with hypermedia interfaces, and hypermedia systems with intelligent retrieval techniques. Moreover, the integration of the two components often requires the duplication of their execution semantics. While attractive for rapid prototyping, the merging of different off-the-shelf packages cannot provide extensive enhancement over the packages themselves, because of the limitations in the manipulation and representation of knowledge inherent to each module. Furthermore, adding functionality in the form of other modules is

highly restricted to the integration capabilities and compatibilities of the existing systems.

The symmetric architecture approach should be able to avoid the master-slave shortcomings, by providing a truly integrated environment, both at the level of the knowledge pool and of its manipulation. Accordingly, the kernel of the model consists of an object-oriented hyperstructure.

The hyperstructure class captures the knowledge structure and its status. The associated execution semantics are captured by the corresponding system classes (see figure 6.1). The hyperstructure does not capture the internal knowledge representation and the internal processing mechanisms specific to each module. Whether the knowledge is a fact being asserted, a variable carrying a result value, or a piece of on-line documentation is irrelevant to the hyperstructure. What matters is that it has a representation in the hyperstructure, and that its status can be reflected at that level The separation of structure from content allows for this abstraction and the integration of knowledge representations The execution semantics of the modules are defined by the corresponding system classes. However, only the status of an object is reflected at the hyperstructure level, and only if the object corresponds to a hyperobject instance. This way, the internal mechanisms of modules can be modeled without affecting the hyperstructure To be truly symmetric, the architecture should be able to dynamically reflect the manipulations within one module in the others. This is achieved through inheritance and aggregation, as detailed below.

6.3 An Object-Oriented Hyperstructure

A symmetric architecture requires a single mechanism to represent different data types and the relations among these diverse types of data. In the unified model, the kernel provides this mechanism in an object-oriented solution The kernel

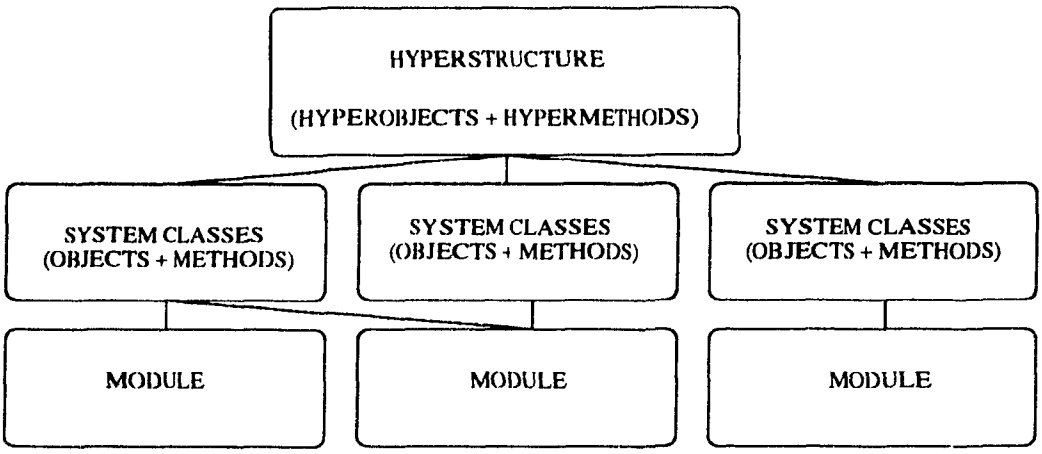

Figure 6.1: The classes hierarchy

hyperstructure represents the *superclass* of all objects. As a class, it represents the abstract data type of the kernel objects, which may be thought of as instances of the hyperstructure; and it defines the operations that can be applied to the objects. As a superclass, it encompasses *system classes* specific to each module attached to the kernel. The system classes in turn depict, the data types of all the objects and the operations the modules perform on these. They can also, in turn, be superclasses for other classes. These can benefit from the object-oriented environment by creating generalization lattices that inherit from different system classes, thus building a richer layered architecture. The last class-layer has instances that correspond to the objects of the modules (see figure 6.1). By analogy with the Dexter model, the hyperstructure could be thought of as the storage layer and its operations, with *several* runtime layers running on top of it. Each runtime layer would represent a module, and each of its objects and operations would be *instantiations* of the storage layer class. The hyperstructure is not directly accessible, but is *realized* only from modular instantiations. Should the same hyperstructure object (hereafter called *hyperobject*) be instantiated under more than one module, the realizations brought to it in one module would reflect in the others. This architecture brings equivalence of objects and methods through a hyperstructure superclass, and synchronization of manipulation through the realizations and instantiations, as will be explained later.

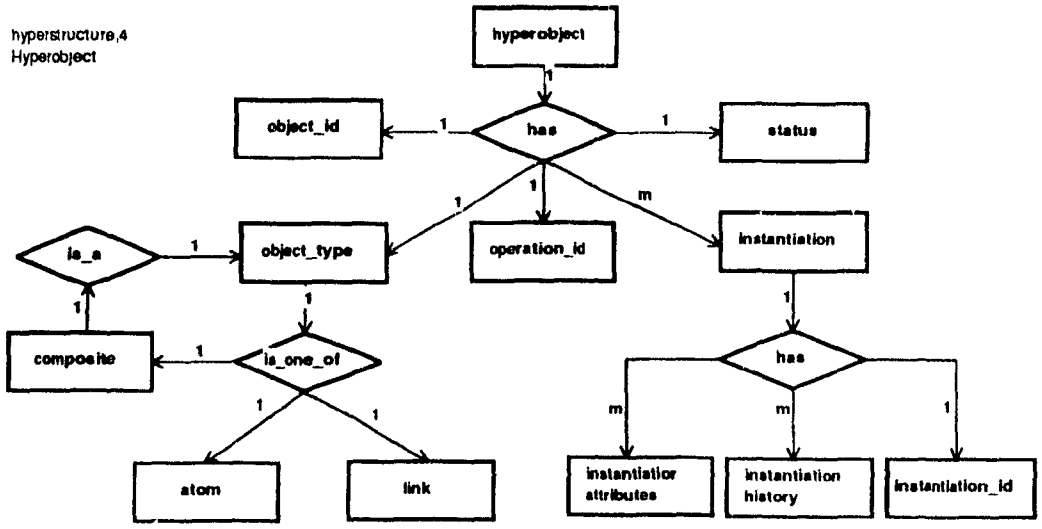

Figure 6.2: The E-R diagram of a hyperobject

6.3.1 The Hyperobject

The hyperobject is depicted in figure 6.2. Each hyperobject has a unique identifier, the *object id*, an *object type* that could be one of *atom*, *relation* and *composite*; and an *operation id* that determines which operation is to be performed on it. Note that the composite type is recursively defined, as in the Dexter model. Additionally, each hyperobject has a *status* determined by the type of object, the operations performed on it, and its instantiations. Finally, the hyperobject has an *instantiation id* that uniquely identifies each of its several instantiations; *instantiator* attributes that determine to which module the instantiation is attached and identify the operations being performed on it; and an *instantiation history* that constitutes the list of such operations.

Several points are worth mentioning here:

- Only the most essential attributes have been elaborated. All attributes specific to system classes or user-defined classes should not be included in the hyperobject. The purpose is to keep an abstraction level that could guarantee the independence of the hyperstructure from any application structure or semantics. Accordingly, the operation-id attribute uniquely determines the

hypermethods to which the hyperobjects react.

- The composite object type is recursively defined, as in the Dexter model. It essentially allows relationships to be captured as objects, and hence to model a network object by aggregation of atoms and links. Furthermore, since relations are modeled as objects, they can also have relations to other objects.

- Any hyperobject can have multiple instantiations (most often these are system class objects), each one bearing a unique instantiation-id, many instantiation attributes and an instantiation history. Note that hyperobjects can have multiple instantiations in one module, and that instantiations need not be unique: several *versions* of a given instance in a given module might coexist, until one is *realized*. At that point, the instantiation history gets updated. The instantiation is at the heart of the equivalence and synchronization mechanism built around the hyperstructure.

Instantiations

Hyperobjects and hypermethods are abstractions that cannot be directly manipulated. They represent equivalence classes with different instances in different system classes. The instantiations inherit properties and behavior from the class they are an instance of, and they can also be manipulated directly by the users through the module interfaces. When manipulations are explicitly realized (i.e. written) by the users, the corresponding methods will be applied to the objects they instantiate, and will eventually propagate to the corresponding hyperobject.

Generalizations

System class hyperobjects can be used to implement generalizations: an object may refer to a set of objects from different classes. The resulting lattice allows the

object to inherit *default behaviors* from diverse classes. Its ordering determines the precedence in case of inheritance conflicts.

6.3.2 HyperMethods

Methods consist of code that manipulates or returns the state of an object. Objects interact with one another through *messages*. Each message corresponds to a method that executes it. Objects react to messages by executing the corresponding method. The hyperstructure methods subdivide into class and instance methods and are the primitive operations allowed on the hyperstructure. They essentially consist of alterations of the object attributes and methods; changes to the object lattices, and alteration of system classes. The hierarchy of hypermethods is as follows:

- Class changes:
    1. add a parent class;
    2. delete a parent class;
    3. add a system class;
    4 delete a system class;
    5. modify a class name.
- Object changes:
    1. Attribute changes: add, delete, modify name, modify domain, modify inheritance;
    2. Method changes: add, delete, modify name, modify code, modify inheritance.

6.3.3 Instance Methods

The hyperstructure methods classify into:

- primitive operations (create, destroy, get, set);
- execution operations (enable, select, execute);
- access operations (open, close).

The primitive operations allow the creation and destruction of instances of an object; and the getting or setting of an attribute value. The access operations allow an instance to be opened after checking privilege clearance and access status (i.e. if it is already being accessed); and to reset it accordingly before closing it. Execution primitives allow an instance to be enabled according to its status, select it among other instances and execute the operation corresponding to its operation-id.

6.4 Modeling Petri Nets

A simple Petri net can be modeled as a system class instance of the hyperstructure. The places and transitions are instances of atoms, and the arcs are instances of relations. The Petri net itself can be a *composite* object, made of *atoms* and *links*, as depicted in figure 6.3. Specialized attributes and operations can be defined using the hypermethods. For example, a new "has token" attribute can be created and added to the place object, from which a "live" status could be determined by an appropriate operation "evaluate place". Instances of the place object that hold a "live" status will see the "enable" execution operation adding them to the set of selectable objects ready for an operation execution. The operation itself is determined by the operation-id attribute inherited from the atom object. The transition object can also have *priority*, *minimum release time*, *maximum latency* and *maximum availability*. The corresponding *network* object instance can then decide on priorities

and enabling conditions in firing the Petri net transitions and sending adequate messages to all objects concerned with the execution semantics (i.e. passing tokens from places to other, firing transitions, etc...). Note that all interaction is performed by message passing, except for instances of system class objects (e.g. Petri net objects) that are instantiations that can be also directly manipulated by the Petri net module interface. The mechanism is identical for all the modules. The modeling of expert and hypermedia systems is straightforwardly derived (see figure 3.1) using the equivalence relation developed in the previous chapter (see figure 6.4). As for object-oriented multimedia databases, they have been investigated by [99, 26], and others. Moreover, an object-oriented hypertext model has been presented by [57], and formalized in the VDM [1]. The Dexter model itself (see 3.4) can be easily turned into an object-oriented structure, thus covering the broadest spectrum of actual hypertext systems. Furthermore, the hyperstructure straightforwardly accommodates the storage data structure of the Dexter model. At the level of integrated object-oriented systems, similar work has been investigated [4] for databases and expert systems, and [65] for multimedia authoring systems.

6.4.1 Aggregations

While hyperobjects provide the structure by which different knowledge representations can be abstracted, aggregation is the key mechanism by which they get integrated and through which manipulations propagate back to all the modules concerned. In aggregations, an object abstracts (i.e. "stands for") all the objects and instances, possibly from different classes, in its aggregation lattice. Hyperobjects are then aggregations of objects that represent the same knowledge abstraction in different representations.

As depicted in figure 6.4, the unsatisfactory status of the engine is depicted as a fact in the rule-base, a place in the Petri net, and a document in the hypermedia. The

[1] Vienna Design Methodology

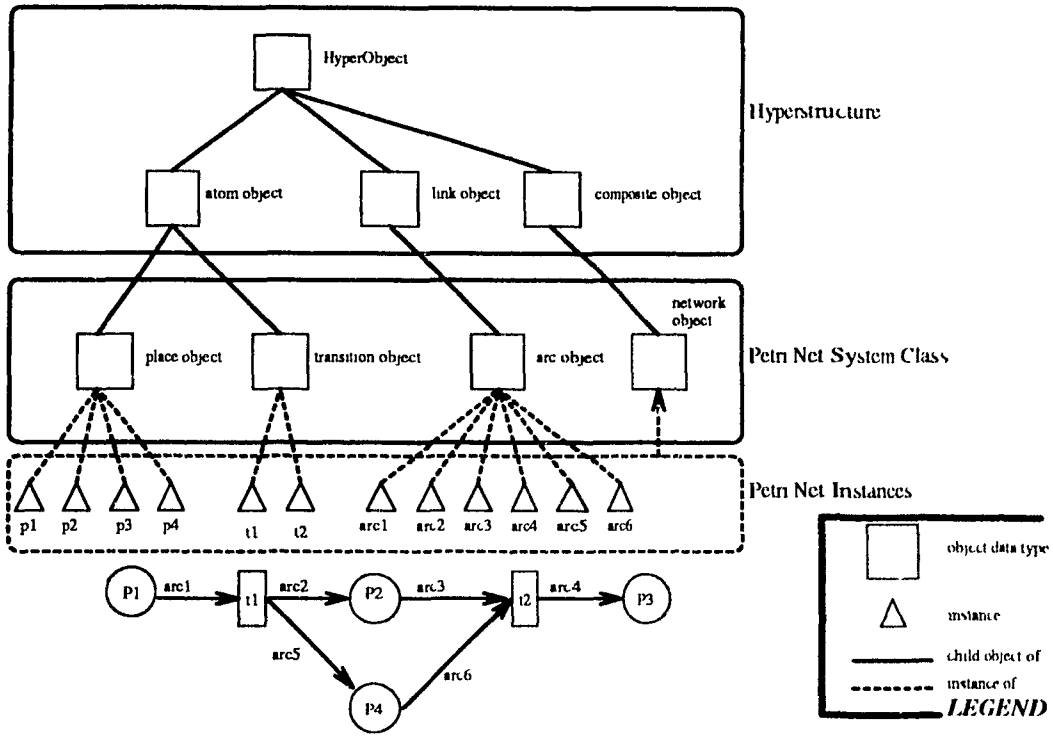

Figure 6.3: Modeling Petri nets

corresponding hyperobject has thus three instantiations; one in each module. The lattice of objects and instances that connects the instantiations to the hyperobject is an aggregation lattice. Realizing instantiations means passing a message (that represents both a method and an attribute) to an instance. The instance being also a logical instance of its parent classes, the instance status propagates to the hyperobject. By aggregation, the same status and properties propagate down the aggregation lattice, and all the instantiations that belong to the same hyperobject get updated. In the example, opening the *unsatisfactory* document in the hypermedia module reflects on the status of the hyperobject, and the action is thus propagated to the corresponding fact and place in the expert system and Petri net respectively. The *(engine unsatisfactory)* fact gets asserted, and place *p1* of the Petri net gets a token. In turn, these updated instances can trigger or apply methods that will reflect back to the hyperobject and down the aggregation lattice to their equivalent objects in other modules. In the example, the *(engine unsatisfactory)* fact being asserted, the *timings rule* will be triggered as soon as the hyperobject corresponding to *(misfiring*

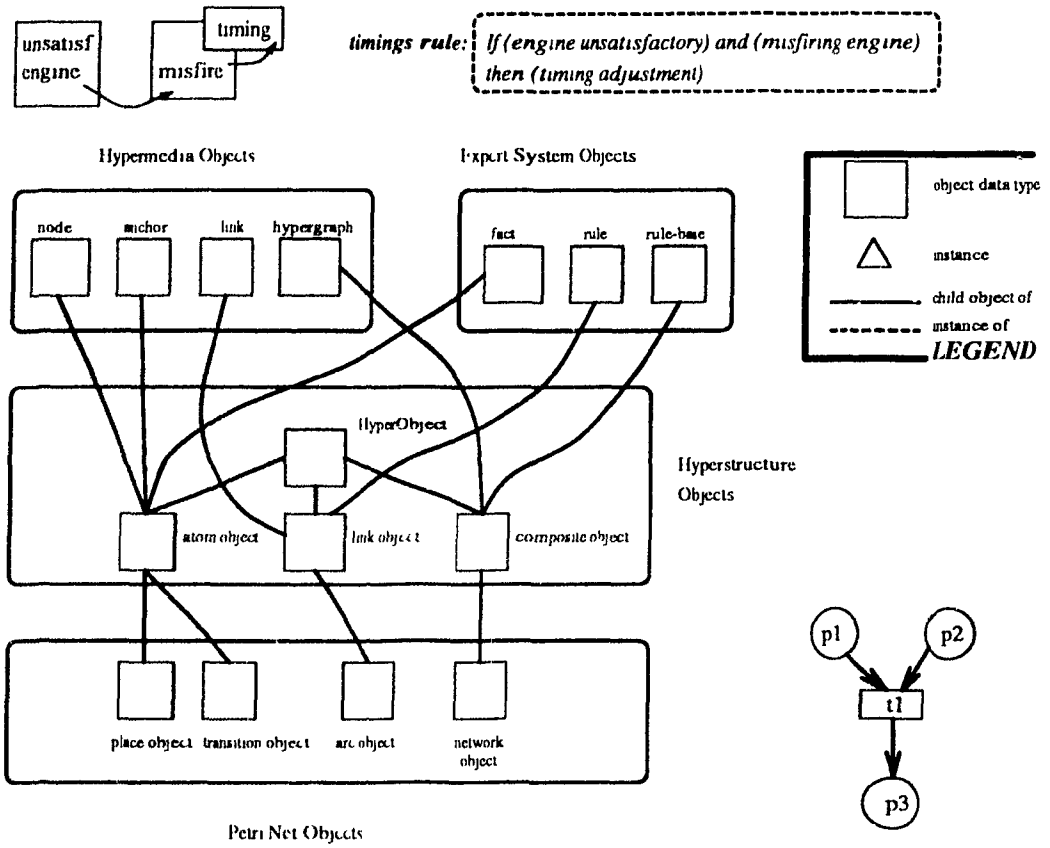

Figure 6.4: Equivalence using hyperobjects

engine) gets activated in any module[2]. The *(timing adjustments)* fact is then asserted. In turn, this instantiation modification reflects at the hyperobject level and down to the relevant instances of other modules. Aggregation and abstraction thus provide equivalence and integration among knowledge structures, while methods, inheritance and aggregate propagation of properties achieve the synchronization in manipulation. Several points are worth mentioning here:

- Propagating properties down the aggregation lattice is different from inheritance. In the first mechanism, attribute values get propagated from a given instance to its children in the aggregation lattice. In the latter, whole classes inherit properties from their parent classes. If inheritance replaces propaga-

[2]If there exists no equivalent object in a given module, then there is obviously no effect on this module. What was forced through Petri nets in the equivalence mappings is here nicely eluded by the object-oriented architecture.

tion in our example, opening a document would assert all the expert system facts!

- The knowledge representations need not be *exactly* equivalent. For example, removing the *symptoms rule* from the expert system does not harm the encapsulated knowledge, and its equivalent hypermedia representation can be retained. The overall behavior of the system does not get disrupted, and the *timing problem* rule can still be activated upon the assertion of *(misfiring engine)*. This *loosely coupled* architecture removes some of the rigidities and artificialities of exact equivalent modeling, while retaining a similar global behavior.

- The knowledge representations remain independent and separately accessible through their own module interfaces. The leverage obtained by different representations and manipulation paradigms does not pose restrictions on the module's own representations and manipulations

6.5 Conclusion

The unified model has an open, symmetric architecture that avoids the drawbacks of the intelligent systems architectures depicted in chapter 4 The hyperstructure supports a powerful object-oriented network representation. It is based on a hyperobject that is application independent, and that provides integration of structure through abstraction. Instantiations allow the modification of the hyperobject through the module interfaces. Inheritance and aggregate propagation update the hyperobject and its corresponding instances through message passing, thus providing synchronization of the modules. Leverage is achieved by this integration and synchronization of modules.

Chapter 7 Conclusion

7.1 Thesis Summary

A survey of past and present hypermedia systems has been achieved, and their characteristics examined. Intelligent system architectures have been evaluated, from which functional requirements for a unified architecture have been derived. Equivalence mappings that equate the structure and behavior of hypermedia networks and expert systems in a lose way have been developed. A unified model has been designed, that synthesizes the structural and behavioral mappings in an object-oriented architecture.

The model was designed so as to merge the knowledge representation and manipulation paradigms found in knowledge-based and hypermedia systems. With regard to knowledge-based systems, the model allows the problem of minimalist explanations to be remedied and allows to knowledge to be elicited from unstructured sources. Minimalist explanations are complemented with unformatted supportive documentation that is presented in an intuitive organization by a hypermedia module. In addition, knowledge elicitation is achieved by translating the network structure and browsing behavior of the hypermedia module on the very same unformatted documentation into the corresponding rule-based format.

With respect to hypermedia systems, the model provides intelligent processing capabilities, and partly remedies to the characteristic problem of disorientation. Intelligent processing results from the equivalence mappings and integration with the expert system module. The unified model architecture differs in this sense from hypermedia architectures in that it does not exploit "end-node processes" as in augmented-transition networks, but rather exploits the parallel rule-based

representation that can be derived from the network structure itself. Disorientation is attenuated by modeling browsing semantics that exhibit an intelligent behavior These semantics can be modeled by the user, or derived from a corresponding rule-based system. When the browsing semantics are modeled by the user, they can be translated into an equivalent rule-base using the same mechanism as for knowledge elicitation.

7.2 Future Developments

Many enhancements could be brought to the model before actual implementation; notably at the level of knowledge representation, classes, and user interface

7.2.1 Representation Enhancements

The Petri net representation used in the equivalence mappings could be extended to high-level Petri nets such as colored or logical ones. This enhancement would allow the modeling of a fully-fledged rule-based syntax, and to extend the equivalence mappings to frame-based representations. High level Petri nets would also affect the mapping to the hypermedia representation at the level of node clustering and browsing semantics. The enhancement of Petri nets is also required if the representation is going to serve as a basis for a modeling and simulation module At that level, matrix representations and analysis tools need to be explored and their correspondence to hypermedia and expert systems needs to be refined.

7.2.2 Model Enhancements

Class Enhancements

Only the classical object-oriented mechanisms have been tackled at the level of the hyperstructure superclass. Methods need to be further specified in terms of hyperstructure objects. At the level of system classes, only the atomic objects have been modeled. Path, network and web objects and their respective methods need to be defined for the hypermedia system class before new methods can be derived. Though frame modeling naturally derives from object structures, the expert system class also needs to be refined so as to model a complete shell structure. Derivation of Petri net methods for reachability analysis and reduction are also required at the level of the Petri net system class.

User Interface Development

Interface paradigms should be derived from the methods at the level of the model operations. These methods could easily benefit from the homogeneity derived from the object-oriented structure. Separate classical approaches should be available, that interface to the modules directly.

7.3 Application Fields

The *processing* of information in a hypermedia structure and format enables new, and improves existing, application areas. The abstraction layer of the hyperstructure allows the integration of manipulations and representations in multiple media. In the domain of design, the unified model allows the alleviation of the problem of minimalist information, either at the level of explanations, or at that of user input (e.g. rating a plot, or more generally a graphic document). It should also help in

the integration of expert systems facilities and on-line documentation of a CAD package such as [58]. As such, the model architecture should be able to provide an attractive alternative to current solutions.

7.4 Conclusion

The conclusion of this dissertation is that an intelligent hypermedia architecture that integrates and synchronizes different knowledge representations in a synergistic approach can be derived. The hypermedia paradigm offers a network architecture that supersedes hierarchical ones; a support for unformatted information that provides new possibilities in many application fields, an authoring paradigm that fits the concept of an evolving structure; and an aptitude for integration that should be an attractive alternative to many hybrid architectures. The unified model was elaborated for these purposes, and its hyperstructure reflects the hypermedia paradigm and distinguishes it from other intelligent systems architectures.

References

[1] F. Afrati and C.D. Koutras. A hypertext model supporting query mechanisms. In *First European Conference on Hypertext*, pages 52–66, November 1990.

[2] Alfred V. Aho, Brian W. Kernighan, and Peter J.Weinberger. *The AWK programming language.* Addison-Wesley, Reading, Mass., 1988.

[3] Robert M. Akscyn and Donald L. McCracken. Experience with the ZOG human-computer interface system. *Int'l J. of Man-Machine Studies*, 2:293–310, 1984.

[4] Nat Ballou and Hong-Tai Chou. Coupling an expert system shell with an object-oriented database system. *Journal of Object Oriented Programming*, pages 12–21, June-July 1988.

[5] Thierry Barsalou and Gio Wiederhold. Cooperative hypertext interface to relational databases. In *Proceedings: Thirteenth Annual Symposium on Computer Applications in Medical Care (SCAMC-13)*, pages 383–387, November 1989.

[6] M. Bartschi. An overview of information retrieval subjects. *IEEE Computer*, 18(5):67–84, May 1985.

[7] C. Beeri and Y. Kornatzky. A logical query language for hypertext systems. In *First European Conference on Hypertext*, pages 67–80, November 1990.

[8] Michael L Begeman and Jeff Conklin. gIBIS: a tool for all reasons. *Journal of the American Society for Information Science*, 40(3):200–213, May 1989.

[9] C. Berge *Graphs and Hypergraphs.* American Elsevier, New York, 1973.

[10] Mark Bernstein The bookmark and the compass: Orientation tools for hypertext users. *ACM Transactions on Office Information Systems*, 7(1):34–45, January 1989.

[11] F. Bienner, M. Guivarch, and J.M. Pilon. Browsing in hyperdocuments with the assistance of neural networks. In *First European Conference on Hypertext*, pages 288–297, November 1990.

[12] James Bigelow. Hypertext and Case. *IEEE Software*, 5(2):23–27, March 1988.

[13] Czejdo Bogdan. Using an E-R query and update interface for rapid prototyping of hypertext systems. In *Proceedings of the Hawaii International Conference on System Science*, pages 227–236, January 1990.

[14] Karl Friedrich Bohringer and Frances Newbery Paulisch. Using constraints to achieve stability in automatic graph layout algorithms. In *CHI'90*, pages 43–51, April 1990.

[15] Larry Brelawski and Robert Lewand. *Intelligent Systems Design*. John Wiley & Sons, New York, 1991.

[16] H.P. Brondmo and G. Davenport. Creating and viewing the elastic Charles – a hypermedia journal. In C. Green and R. McAleese, editors, *Hypertext: Theory into Practice II*. Intelect Press, 1990.

[17] P. J. Brown. Interactive documentation. In *Software – Practice and Experience*, volume 16, pages 291–299, March 1986.

[18] P.D. Bruza. Hyperindices: A novel aid for searching in hypermedia. In *First European Conference on Hypertext*, pages 109–122, November 1990.

[19] Vannevar Bush. As we may think. *Atlantic Monthly*, pages 101–108, July 1945

[20] Vannevar Bush. *Science is not enough* Morrow, New York, 1967

[21] Brad Campbell and Joseph M. Goodman. HAM: A general purpose hypertext abstract machine. In *Communications of the ACM*, volume 31, pages 856–861, July 1988.

[22] Michael Caplinger. Graphical database browsing. In *ACM SIGOIS Bulletin*, volume 7, pages 113–119, 1986.

[23] Timothy Catlin, Paulette Bush, and Nicole Yankelovich. Internote: Extending a hypermedia framework to support annotative collaboration In *Hypertext'89*, pages 365–378, November 1989.

[24] T.J.O. Catlin and K.E. Smith. Anchors for shifting tides: Designing a 'seaworthy' hypermedia system. In *Proc. Online Information 88*, pages 15–25, December 1988.

[25] R. Jesse Chaney, Frank M. Shipman, and G. Anthony Gorry. Using hypertext to facilitate information sharing in biomedical research groups In *Proceedings. Thirteenth Annual Symposium on Computer Applications in Medical Care (SCAMC-13)*, pages 350–354, November 1989.

[26] S. Christodoulakis, F. Ho, and M. Theodoriou. The multimedia object presentation manager of minos: A symmetric approach *ACM OIS*, pages 295-310, 1986.

[27] Peter Clitherow, Doug Riecken, and Michael Muller. VISAR: A system for inference and navigation of hypertext. In *Hypertext'89*, pages 293–304, November 1989.

[28] Georger H. Collier. Thoth-II: Hypertext with explicit semantics. In *Hypertext'87*, pages 269–290, November 1987.

[29] Jeff Conklin. Hypertext: An introduction and survey. *Computer*, 20(9):17–41, September 1987.

[30] Jeff Conklin and Michael Begeman. gIBIS: A hypertext tool for exploratory policy discussion. *ACM Transactions on Office Information Systems*, 6(4):303–331, October 1988.

[31] Mariano P. Consens and Alberto O. Mendelzon. Expressing structural hypertext queries in graphlog. In *Hypertext'89*, pages 269–292, November 1989.

[32] James E. Coolahan. A timed petri net methodology for specifying real-time systems timing requirements. In *Proceedings of the Int'l Workshop on Timed Petri nets*, Torino, Italy, July 1985.

[33] G. Crane. From the old to the new: Integrating hypertext into traditional scholarship. In *Hypertext'87*, pages 51–55, November 1987.

[34] W. Bruce Croft and Howard Turtle. A retrieval model incorporating hypertext links. In *Hypertext'89*, pages 213–224, November 1989.

[35] Andries Van Dam. Hypertext '87 keynote address. *Communications of the ACM*, 31(7):887–895, July 1988.

[36] Centre de Recherche Informatique de Montreal. An expert system with a hypermedia interface. Personnal communication with Carlos Saldanha.

[37] Norman Delisle and Mayer Schwartz. Neptune: A hypertext system for CAD applications. *ACM*, pages 132–143, 1986.

[38] Dennis E. Egan, Joel R. Remde, and Carol C. Lochbaum. Formative design-evaluation of superbook. *ACM Transactions on Office Information Systems*, 7(1):30–57, January 1989.

[39] Douglas C. Engelbart and W.K. English. A research center for augmenting human intellect. In *AFIPS Conference Proceedings*, volume 33, Washington D.C., 1968. The Thompson Book Company.

[40] Richard Gary Epstein. Graphical query language for hypertext database systems. In *Proceedings - Graphics Interface*, pages 47–54, June 1989.

[41] Halasz F. and Schwartz M. The Dexter hypertext reference model. *Proceedings of the Hypertext Standardization Workshop*, pages 95–133, January 1990.

[42] Steven Feiner. Seeing the forest for the trees: hierarchical display of hypertext structure. In *Conference on Office Computing Systems 88*, pages 205–212, 1988.

[43] Gerhard Fisher, Thomas Mastaglio, Brent Reeves, and John Rieman. Minimalist explanations in knowledge-based systems. In *Proceedings of the Twenty-Third Annual Hawaii International Conference on System Sciences*, pages 309–317, January 1990.

[44] Gerhard Fisher and Raymond McCall. JANUS: Integrating hypertext with a knowledge-based design environment. In *Hypertext'89*, pages 105–118, November 1989.

[45] H.C. Fordsick, R.h. Thomas, G.G. Robertson, and V.H. Travers. Initial experience with multimedia documents in diamond. In *IEEE Database Engineering Quaterly Bulletin*, volume 7, September 1984.

[46] L. Friedlander. The Shakespeare project. In S. Ambron and K. Hooper, editors, *Interactive Multimedia: Visions of Multimedia for Developers, Educators and Information Providers*, pages 115–141. Microsoft Press, 1988.

[47] Mark E. Frisse. Retrieving information from medical hypertext systems. In *Annual Symposium on Computer Applications in Medical Care*, pages 441–444, 1988.

[48] Richard Furuta and P. David Stotts. Timing analysis of synchronous browsing in Petri net-based hypertext. Technical Report UMIACS-TR-89-53, University of Maryland, May 1989.

[49] Pankaj K. Garg. Abstraction mechanisms in hypertext. *Communications of the ACM*, 31(7):862–870, July 1988.

[50] Frank G. Halasz. Reflections on Notecards: Seven issues for the next generation of hypermedia systems. *Communications of the ACM*, 31(7):836–852, July 1988.

[51] Phil Hayes and Jeff Pepper. Towards an integrated maintenance advisor. In *Hypertext'89*, pages 119–127, November 1989.

[52] W. Hershey. Idea processors. In *BYTE Magazine*, page 337. McGraw Hill, June 1985.

[53] M.E. Hodges, R.M. Sasnett, and M.S. Ackerman. A construction set for multimedia applications. *IEEE Software*, 6(1):37–43, January 1989.

[54] M. Hofmann, U. Schreiweis, and H. Langendorfer. An intergated approach of knowledge acquisition by the hypertext system CONCORDE. *First European Conference on Hypertext*, pages 166–179, November 1990.

[55] Randy H. Katz and Tobin J. Lehman. Database support for versions and alternatives of large design files. In *IEEE Transactions on Software Engineering*, volume 10, pages 191–200, March 1984.

[56] Susan K. Kinnell. Comparing Hypercard and Guide. In *Database*, volume 11, pages 49–54, 1987.

[57] Danny Lange. A formal model of hypertext. In *Proceedings of the Hypertext Standardization Workshop*, pages 145–166, Gaithersburg, MD, January 1990.

[58] D.A. Lowther and P.P. Silvester. *Computer-Aided Design in Magnetics*. Springer-Verlag, Berlin, 1985.

[59] D. Lucarella. A model for hypertext-based information retrieval. In *First European Conference on Hypertext*, pages 81–94, November 1990.

[60] Robert M.Akscyn, Donald L. McCracken, and Elise A. Yoder. KMS: A distributed hypermedia system for managing knowledge in organizations. *Communications of the ACM*, 31(7):820–835, July 1988.

[61] Brian P. Mc Cune and Daniel G. Shapiro. Rubric: A system for ruled-based information retrieval. *IEEE Transactions on Software Engineering*, 11(9):939–945, September 1985.

[62] R. McCall. Phibis: Procedurally hierarchical issue-based information systems. In *Proc. Conf. Arch. Int'l. Congress on Planning and Design Theory*, pages 17–22, 1987.

[63] R. McCall, P. Bennet, and P. D'Ornozio. Phidias: Integrating cad-graphics into dynamic hypertext. *First European Conference on Hypertext*, pages 152–165, November 1990.

[64] Phillip M. Merlin. Recoverability of communications protocols. *IEEE Trans. on Comm.*, 24(9):1036–1043, 1976.

[65] Max Muhlhauser. A software engineering environment for distributed applications. *The Euromicro Journal*, 27:327–332, September 1989.

[66] T. Murata and J.Y. Koh. Reduction and expansion of live and safe marked graphs. *IEEE Tran. Circuits Syst.*, 27(1):68–70, 1980.

[67] T. Murata and D. Zhang. A predicate-transition net model for parallel interpretation of logic programs. *IEEE Trans. Software Eng.*, 14(4):481–497, April 1988.

[68] Tadao Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–579, April 1989.

[69] Ted Nelson. The Xanadu file server. In *Byte Magazine*, pages 298–299. McGraw Hill, September 1990.

[70] Theodore H. Nelson. Getting it out of our system. *Information Retrieval: A critical Review*, 20(9):17–41, September 1987.

[71] Steven R. NewComb. Explanatory cover material for section 7.2 of x3v1.8m/sd-7. In *Proceedings of the Hypertext Standardization Workshop*, pages 179–188, January 1990.

[72] Jakob Nielsen. *Hypertext and Hypermedia*. Academic Press, Boston, Mass., 1990.

[73] H.P Nii. Blackboard systems. *AI Magazine*, 7(3):38–53, Summer 1986.

[74] H.P Nii. Blackboard systems. *AI Magazine*, 7(4):82–106, Summer 1986.

[75] R. Ogawa, H. Harada, and A. Kameko. Scenario-based hypermedia: a model and a system. In *First European Conference on Hypertext*, pages 38–52, November 1990.

[76] Tim Oren. The architecture of static hypertext. In *Hypertext'87*, pages 291–306, November 1987.

[77] H. Van Dyke Parunak. Reference data model group (rdmg): Work plan status. In *Proceedings of the Hypertext Standardization Workshop*, pages 9–13, March 1990.

[78] Amy Pearl. Sun's link service: A protocol for open linking. In *Hypertext'89*, pages 137–146, November 1989.

[79] James Lyle Peterson. *Petri net theory and the modeling of systems*. Prentice-Hall, Englewood Cliffs, New Jersey, 1981.

[80] James Lyle Peterson. *Petri Net Theory and the Modeling of Systems*. Prentice Hall, Englewood Cliffs, New Jersey, 1981.

[81] X. Pintado and D. Tsichritzis. Satellite: Hypertext navigation by affinity. In *First European Conference on Hypertext*, pages 274–287, November 1990.

[82] Martha C. Polson and J. Jeffrey Richardson, editors. *Foundations of intelligent tutoring systems*. L. Erlbaum Associates, Hillsdale, N.J., 1988.

[83] Darrell R. Raymond and Frank Wm. Tompa. Hypertext and the Oxford English Dictionary. *Communications of the ACM*, 31(7):871–879, July 1988.

[84] Wolfgang Reisig. *Petri Nets: An introduction and Survey*. Springer-Verlag, 1985.

[85] Howard Rheingold. *Tools for thought: the people and ideas behind the next computer revolution*. Simon & Shuster, New York, 1985.

[86] W. Rittel and W. Kunz. Issues and elements of information systems. Working Paper. Centre for Planning and Development Research, University of Califirnia, Berkeley, 1970.

[87] W. Schuler and J. Smith. Author's argumentation assistant (AAA): A hypertext-based authoring tool for argumentative texts. *First European Conference on Hypertext*, pages 137–151, November 1990.

[88] Ben Shneiderman. User interface design for the hyperties electronic encyclopedia. In *Hypertext'87*, pages 189–194, November 1987.

[89] J. B. Smith. WE: A writing environment for professionals. Technical report, University of North Carolina at Chapel Hill, August 1986.

[90] P. David Stotts and Richard Furuta. Petri-net-based hypertext. document structure with browsing semantics. *ACM Transactions on Office Information Systems*, 7(1):3–29, January 1989.

[91] Walter F. Tichy. Design, implementation and evaluation of a revision control system. In *ICSE'82*, pages 58–67, 1982.

[92] Toomas Timpka. Knowledge-based decision for general practitioners; an integrated design. *Comput. Methods Prog. Biomed.*, 25:49–60, August 1987.

[93] Frank Wm. Tompa. A data model for flexible hypertext database systems. *ACM Transactions on Office Information Systems*, 7(1):85–100, January 1989.

[94] Randall H. Trigg. *A Network-Based Approach to Text Handling for the Online Scientific Community*. PhD thesis, University of Maryland, 1983.

[95] Kenneth Utting and Nicole Yankelovich. Context and orientation in hypermedia networks. *ACM Transactions on Information Systems*, 7(1):85–100, January 1989.

[96] C. J. van Rijsbergen. Towards an information logic. In *SIGIR'89*, pages 77–86, 1989.

[97] Udo Wahn and Ulrich Reimer. Automatic generation of hypertext knowledge bases. In *Conference on Office Computing Systems*, pages 182–188, March 1988.

[98] Janet H. Walker. Document examiner: Delivery interface for hypertext documents. In *Hypertext'87*, pages 307–323, November 1987.

[99] Darrell Woelk, Won Kim, and Willis Luther. An object-oriented approach to multimedia databases. *ACM OIS*, pages 311–325, 1986.

[100] Nicole Yankelovich and Steven M. Drucker. Intermedia: The concept and the construction of a seamless information environment. *Computer*, 21(1):81–96, January 1988.

[101] Polle T. Zellweger. Scripted documents: A hypermedia path mechanism. In *Hypertext'89*, pages 1–14, November 1989.

Appendix A Translation Algorithms

This appendix contains the algorithms based on the equivalence mappings and tested on the αTrellis Petri net hypertext prototype developed by Richard Furuta and P. David Scotts at the University of Maryland [90]. The algorithms have been implemented in awk [2].

A.1 From Expert Systems to Hypermedia

The awk translator actually creates two files: one that contains the Petri net description and one that describes the mapping from the places to the hypertext nodes. A simplified version of the general algorithm will be described here. It is linearized and avoids special cases (e.g. taking into account retractions) and parsing rules, for the sake of clarity. It uses the incidence matrix representation described in [68] and [80]. Two separate matrices are maintained: a D_{in} matrix that records all places with arcs pointing *towards* transitions, and a D_{out} matrix that records all places with arcs pointing *from* transitions. The resulting matrix D, used in reachability analysis for example, is $D = D_{out} - D_{in}$, with D, D_{in}, D_{out} having dimensions $(m \times n)$, where m is the number of transitions (rules) and n is the number of places (facts) in the Petri net [1]. As can be seen from the implementation, the inverse translation is straightforward.

Initialize variables and arrays:

$trcount \leftarrow 0$; initialize transition counter

[1] Similar work has been achieved by [67] and has resulted in a formal procedure for transforming a given logic program into the incidence matrix of high-level nets.

$plcount \leftarrow 0$; initialize place counter

For each rule do:

map rule identifier (name) to transition name:

$trcount++$; increment transition counter

$transition[trcount] \leftarrow [rulename]$; assign the rule identifier to the transition

With the antecedent part of the rule do:

For each condition do:

$for(i = 1 to\ plcount)$; check if the fact s already exists

$if(s == place[i])$

$plcount++$; it doesn't: increment the place counter

$place[plcount] = s$ create a new place and assign it an identifier

$din[trcount, plcount]++$; update D_{in} accordingly

$din[trcount, i]++$; it does: update D_{in} accordingly

With the consequent part of the rule do:

For each action do:

$for(i = 1 to\ plcount)$; check if the fact s already exists

$if(s == place[i])$

$plcount++$; it doesn't: increment the place counter

$place[plcount] = s$ create a new place and assign it an identifier

$dout[trcount, plcount]++$; update D_{out} accordingly

$dout[trcount, i]++$; it does: update D_{out} accordingly

until EOF(expert system).