D I P L O M A R B E I T

The Use of Intelligent Hypermedia in Architectural Design Environments — a Conceptual Framework

ausgeführt am Institut für Informationssysteme
Abteilung für Datenbanken und Expertensysteme
der Technischen Universität Wien

unter der Anleitung von
o.Univ.Prof. Dr. Georg Gottlob

und

Univ.Ass. Dr. Wolfgang Slany
als verantwortlich mitwirkendem Universitätsassistenten

durch

Marcus Herzog
Matr.Nr. 8727266
Promenadegasse 57 D2/2
A-1170 Wien

Wien, im November 1994

To my parents,
Anna and Gerhard Herzog,
for their long-standing support

Deutsche Kurzfassung der Diplomarbeit

Der Einsatz von Computern zur Entwurfsunterstützung im Bereich der Architektur ist eine interessante Aufgabenstellung. Die meisten Ansätze, die in der Literatur gefunden werden können, konzentrieren sich auf eine "Automatisierung" des Entwurfprozesses — der Computer generiert selbständig Lösungsvorschläge, die dann übernommen, oder wieder verworfen werden können. Im Gegensatz dazu folgt die vorliegende Arbeit der Vorstellung, die Fähigkeiten des Menschens durch Bereitstellung einer computergestützten Umgebung optimal zu unterstützen, um so bessere Ergebnisse erzielen zu können. Ziel dieser Arbeit ist es, ein Wissensmodell für architektonisches Entwurfswissen zu erarbeiten, das einem solchen System zu Grunde gelegt werden kann.

Im ersten Teil der Arbeit werden verschiedene Theorien des Designs und deren Anwendbarkeit in Hinsicht auf unsere Problemstellung untersucht. Christopher Alexanders *Pattern Language* ist ein Entwurfssystem, das versucht, mit Hilfe einer Sammlung von Teilproblemen und korrespondierender Teillösungen alle auftretenden Entwurfsprobleme zu lösen. Dabei "steckt" das Entwurfswissen in den einzelnen *Pattern* und der Entwurfsprozeß besteht aus einer Analyse der Problemstellung und einer Synthese mit Hilfe der passenden *Patterns*.

Der Ansatz in dieser Arbeit greift diese Idee der informalen Wissensrepräsentation auf. Hypermedia als semi-formaler Wissensrepräsentationsmechanismus scheint ein geeignetes Vehikel zu sein, um diesen Ansatz umzusetzen. Zwei prototypische Implementierungen, HySAT und HySAT-W3, versuchen diese Behauptung empirisch zu untermauern. Die erzielten Ergebnisse ermuntern durchaus zu weiteren Untersuchungen in dieser Richtung. Die Evaluierung der Prototypen deckt allerdings Schwachstellen der vorliegenden Implementierungen auf. Diese sind jedoch größtenteils auf Beschränkungen in den Modellen der verwendeten Hypermedia Systeme und nicht auf konzeptionelle Fehler zurückzuführen.

Im Laufe der Arbeit kristallisieren sich zwei Punkte als besonders entscheidend für den Erfolg des Formalisierungsansatzes heraus: *Wissenserwerb* und *Wissensgewinnung*. Während ersterer nach einer flexiblen Strukturierungsmöglichkeit verlangt, beschäftigt sich zweiterer mit dem Auffinden von relevanten Wissensteilen in Hinblick auf eine architektonische Aufgabenstellung. Die Kombination von Methoden der künstlichen Intelligenz mit solchen aus Hypermedia scheint für beide Problemstellungen ein geeigneter Ansatz zu sein. Im letzten Abschnitt präsentieren wir unser Wissensmodell, das auf diesen Überlegungen aufbaut und eine flexible Darstellung architektonischen Entwurfswissens ermöglicht.

Abstract

Using computers to support the architectural design process is a challenging issue. Up to now most design-support systems concentrate on "automating" the design process to generate design solutions. This thesis emphasizes a different notion. "Augmenting man's intellect" as introduced by Douglas Engelbart means to increase the capability of a person to approach a complex problem situation. It is the objective of this work to propose a knowledge model of architectural design knowledge that can be implemented to *support* architects in their design task. To achieve this goal we first review different theories on design and evaluate their relevance according to the problem setting. We propose the use of hypermedia as a semi-formal knowledge representation technique to encode design knowledge. "Case-studies" are used to analyze particular buildings and extract general principles. The evaluations of two prototypical implementations of the proposed knowledge model point out two main purposes of the formalization of design knowledge: The *structuring* of the design space and the *retrieval* of design knowledge. We exploit the notion of *intelligent hypermedia* by combining techniques from the field of artificial intelligence with ideas originating from the field of hypermedia. The support for exploring an emergent structure within the design space is a key aspect of our considerations. We think that the use of our design knowledge model in design-supporting systems will bring a new quality to the design process and will augment the capabilities of human designers.

Contents

List of Figures

Chapter 1

Introduction

The use of computers in the field of design opens challenging research issues. In this chapter we will first review different notions towards using computers for design as found in the short history of this field. We will then argue, why a particular approach seems to us more promising than others in respect to our problem setting. Finally, we will give an overview of the remaining parts of this thesis.

1.1 Computers and Design

The topic of this thesis stems from the author's passion for both architecture and computer science. Although these two fields do not seem to have much in common at first glance, both deal with the creative act of building complex systems. As Richard Gabriel puts it in his column *Critic-at-Large* in the January 1994 issue of JOOP [35]

> *'There are few fields that blend art and science: architecture is one and computer science is another.'*

Besides the "academic" exploration of this rather surprising relationship between the two fields, the driving force behind this work is the question of how to make use of technological possibilities found in one field in the other. Particularly, how to gain advantage of the computer — broadly spoken — for support of the architectural design process. Which kind of new tools might bring a new quality to the design process? And in contrast to other approaches, we do not just mean transferring well-known paradigms to a new kind of workbench, but changing the way architects think about the very act of designing.

If people think about the use of computers in the field of design, the term computer-aided design (CAD) comes to their mind. Although the term CAD is

not very well defined, the common understanding is that of subsuming all kinds of technical drawing programs under that term. Now, what is the *intention* of this kind of programs? Their main purpose is to mimic the traditional way of producing drawings — using a pen, a ruler, and a sheet of paper — in the new "digital drawing space".

This migration from the analogue to the digital representation of drawings has of course benefits. Most of them are analogous to the advantage of using a word processor over using handwriting. The structure of electronic documents is more flexible and documents can be easier maintained or even re-used. The physical appearance literally dissolves into an abstract stream of bits. But exactly this abstraction process is at the same time the reason for the main disadvantage of using digital instead of analogue drawings. Because you are left without physical artifacts on which to work with your physical drawing utensils, you are forced to use conceptual operations instead to manipulate the electronic data.

This cognitive overhead — having to learn how to manipulate a complex system like a CAD application — is by far more painful than the transition from handwriting to writing on a computer system. For the latter you just need to learn how to compose words with the help of a keyboard and some additional operations (typically less than twenty) to operate the word processor. In the case of the CAD application, you have to learn hundreds of operations and still will not be able to emulate certain hand drawings. The problem is that in writing you work with a limited number of building blocks (characters) whereas in drawing you work with all possible sets of dots on a two-dimensional surface. In short, writing uses a much more formalized representation method than drawing in general does and therefore can be more easily supported on the computer.

But here it is not our point to analyze the usability of certain CAD applications. What we want to point out is that using CAD as we know it today does not change the way architects design. This does not mean that CAD has no influence on the practical work flow in an office. In fact, most of the time just the contrary is the fact. What we mean is that it does not affect the conceptual design process. This might be due to the fact that most CAD systems are not used as "design tools" but rather as "post-production tools". Architects still design the same way they did before they used CAD applications to visualize their ideas. They do not use CAD to organize their thoughts, analyze the constraints implied on a certain project, or line out the conceptual design of a project. They merely use it to visualize the result of the design process. In this line of argumentation, CAD applications of today are used to *present* ideas rather than to help *generate* ideas.

Will there be a change of this rather miserable condition in the near future? If you skim through the literature [90], a lot of systems are hypothesized that actually *do* design. But how do these systems achieve this goal? By applying sets of rules that transform some initial design state into some final design state. The underlying

formalisms come in different variations like shape grammars [70, 50], mutation and analogy algorithms [37, 81, 91], or in other sorts of knowledge-based systems [15, 55]. These systems usually produce a collection of design proposals, from which the architect (or the client) can choose. Or they select one design on the basis of some *a-priori* defined evaluation criteria. Although these systems may generate artifacts, they are no "architects". Why? Because, as stated at the beginning, architecture blends art and science. And while science can be formalized in general, art cannot. Art depends on some process of interpretation, which in turn depends on some form of consciousness. Have you met any conscious computer program lately?

To be serious, systems that generate whole artifacts on their own are just on one end of the spectrum of design-supporting systems. But is it the most interesting one? From our point of view the answer to this question is no. The act of designing, exploring different possible solutions, inventing novel ways of combining things is too thrilling to be left to the computer. This question is of course debatable and should be open to further discussion. For the rest of this thesis we will take our answer as working hypothesis and talk about a different kind of design-supporting systems.

This other kind of systems are based on the notion of "augmenting man's intellect" as proposed by Douglas Engelbart back in the early sixties of this century. To introduce this notion we cite the beginning of his article *"A Conceptual Framework for the Augmentation of Man's Intellect"* [25].

> *'By "augmenting man's intellect" we mean increasing the capability of a man to approach a complex problem situation, gain comprehension to suit his particular needs, and to derive solutions to problems. Increased capability in this respect is taken to mean a mixture of the following: that comprehension can be gained; that a useful degree of comprehension can be gained more quickly; that better comprehension can be gained where previously the situation was too complex; that solutions can be produced more quickly; that better solutions can be produced; that solutions can be found where previously the human could find none. And by "complex situations" we include the professional problems of diplomats, executives, social scientists, life scientists, physical scientists, attorneys, designers — whether the problem situation exists for twenty minutes or twenty years.'*

Doug Engelbart has been working on environments designed to boost the collective IQ of organizations, institutions, and nations for the last forty years. He was director of the NLS/AUGMENT [26] program at the Stanford Research Institute and developed key aspects of interactive computing like the window user interface, the mouse, and electronic conferencing facilities as early as in the late sixties. At this time systems were typically implemented on time-sharing computers with very

limited memory resources. NLS/AUGMENT was a first step to change the way in which humans do their knowledge work. And this, as he points out, *'is very different from trying to "automate" our old "ways" of doing things'* [28].

Another pioneer in this field, Vannevar Bush, envisaged that kind of working environments even earlier than Doug Engelbart. In 1945, he presented in his article *"As we may think"* [12] the idea of a personal file and library system, the "memex". This device would collect all of a user's books, notes, comments, sketches, etc., and provide some mechanism of associative indexing between these items. Items would be linked to select immediately and automatically each other on request. Users could thereby build whole trails throughout their collected information, linking items of interest together. The memex system was modeled after the way the human memory was seen to work: a web of thoughts carried by the cells of the brain. Although never implemented (Bush was considering a mechanical system based on microfilm) the thought-provoking idea stimulated researchers to investigate this topic since then with ever growing effort.

1.2 Intention of the Research

To summarize we think that one can identify two proponent research directions in design computing: The first one deals with "automating" the design process. The designer states her problem to the computer in a certain formalism and a program computes a design solution to this problem. The other approach is to support a designer — or a team of designers — in their tasks. The first one draws on algorithms that generate design solutions whereas the second deals with problems of design conceptualization to allow humans to reason and communicate about their design solutions.

We think that the second approach has a high potential that has been underestimated so far. This is especially true for the academic field where requirements for design education and design research meet. Design education is some form of "knowledge transfer" from the design expert to the design novice. Within a collaborative environment as sketched by Engelbart and Bush this knowledge transfer could happen in a seamless way. First, the system would provide tools for the individuals to organize and externalize their thoughts and knowledge. Second, through communication and collaboration this knowledge could be shared by all participants of the group.

This work is in the context of architectural design. In our understanding architectural design knowledge is especially hard to communicate. It cannot be reduced to a set of procedures or methods that once learned guarantee a certain quality of design. Experience acquired in practical as well as theoretical studies is the main source of knowledge from which architects draw. Experience *per se* cannot be transferred from one person to another. But it can be analyzed and condensed into

"evaluations" that will help inexperienced designers find their way in the jungle of possible design decisions.

Our long-term goal is concerned with building such an environment, where knowledge about architectural design is communicated between different classes of participants (i.e., expert and novice). It will be used for knowledge acquisition as well as knowledge dissemination throughout the community of users. It will serve as a pool of design knowledge where individuals can contribute and retrieve knowledge on demand. Experts can assist novice designers by presenting their views on design emphasizing certain aspects of the whole problem space. In this thesis we will try to formulate a conceptual framework for such an environment.

1.3 Organization of the Thesis

In the following we will give a short description of the remaining parts of the thesis.

- In Chapter 2, we investigate the notion of design as found in the literature. We then review theories on design and their impact on actual implemented design systems. We conclude by summarizing the aspects that we think are important for our project.
- Chapter 3 discusses the technology of hypertext/hypermedia. To show the broad span of conceptual models of hypertext we review two contrasting approaches, the Dexter Hyptertext Reference Model and the W3 model. We also investigate the influence of hypertext methods on computer-based learning. Finally, we point out the requirements that played a key role in developing our prototypes.
- In Chapter 4, we present two prototypical systems, HySAT and HySAT-W3. We describe our didactic considerations in respect to architectural design knowledge and how we think that a system can support the design process. We close by giving new research directions that resulted from evaluations of our systems.
- Chapter 5 is dedicated to intelligent hypermedia. We think that we can overcome shortcomings of our prototypes by integrating hypertext technology and methods stemming from artificial intelligence research.
- In Chapter 6, we recapitulate our problem and propose a conceptual framework for the use of intelligent hypermedia in architectural design environments. We introduce an information model and reasoning strategies. We finally describe several user scenarios to illustrate the usability of our model.
- Chapter 7 summarizes our findings and presents open problems for future research.

Chapter 2

Design

In this chapter we will first discuss the nature of design knowledge and its influence on representation mechanisms. Most of this material follows the presentation given by Coyne et al. in *"Knowledge-Based Design Systems"* [19]. We will then present different approaches towards a theoretical framework for design knowledge representation. After this, we investigate how these considerations can be applied in actual system implementations. We conclude this chapter by giving an analysis of the requirements that are appropriate for our project.

2.1 The Nature of Design

Design is the process of originating systems and predicting their performance. A design solution is an abstraction of an artifact, providing a description of this artifact. Design can be seen as a goal-directed activity akin to decision making. Defining these goals is a design process by itself. The nature of the desired state is often difficult to specify. Many different models of design are based on the three stage approach: *analysis, synthesis, and evaluation*. Analysis defines the problem and the explicit goals. Synthesis tries to find plausible solutions for the defined problem. Evaluation selects alternative solutions in respect to the original goals. Generation-and-test cycles are used to search the problem space. In these models design can thus be seen as a problem-solving process exploring different possible states, where states represent design solutions.

We can distinguish between different kinds of knowledge involved in the design activity ranging from *common knowledge* to *individual knowledge*. Common knowledge can be viewed as axiomatic in character: in a quadrilateral the sum of the interior angles is 360 degrees. This knowledge is shared by all designers and can be stated as unequivocal. Then there is the type of knowledge that is shared by a certain group, say the proponents of particular design philosophy. Finally, some knowledge

is just bound to the individual. An idiosyncratic design style may just be relevant to some individual but not necessarily accepted by colleagues. These different levels of acceptance of some notion of design are a potential source of inconsistency, which lies inside the nature of design knowledge itself.

Design knowledge deals with information about design elements and the various relationships between these elements. During the design process one has to know how to manipulate the design elements correctly. The control of the design process has to be represented explicitly. This deals with the problem of how to represent design goals. Design goals can be viewed as partial and incomplete design descriptions. They have to be transformed into complete descriptions that constitute design solutions. Because of interaction between goals it is sometimes hard to decompose goal hierarchies. Objectives are particular kinds of goals, which want to be maximized or minimized.

Existing designs are a main source of design knowledge. This does not mean that existing designs are there to be copied. They provide analogies from which partial design solutions can be drawn. Prototypes may serve as a source of general rules and principles. Types are systems of abstractions to define classes of design descriptions. Types help to find an artifact that meets the design requirements. Architectural typology deals mostly with the relationship between form and function of buildings.

The representation of design knowledge should be in a form that bears some similarity to the way human experts feel comfortable about articulating their knowledge. The transformation between multiple representations should be supported so that one can change the representation according to one's needs (e.g., different roles of a building: function, construction, aesthetics, etc.). Representing design knowledge means to represent the knowledge by which we progress from known facts to new facts. Design knowledge is embodied in whatever serves to direct and constrain the search for designs — mainly analogical designs as well as the use of typologies, design rules, and procedures.

There are two main approaches to formalize design knowledge: models and case-studies. We use these two to pass knowledge from one person to another. Models are limited abstractions of phenomena. Rules and procedures are defined to transfer knowledge on how to solve design goals. Predictions can be based on design descriptions applying design models that account for "design behavior". In architecture, case studies observe and analyze particular buildings to extract general principles. An *archetype*, such as a typical small house, embodies the features of a whole class of designs. It can be used as prototype to generate new designs.

To represent design we have to use some kind of description language. This language will be based on some vocabulary. The vocabulary shall enable us to depict reality in the appropriate granularity of abstraction. The granularity depends on the stage of the design process. The description language shall support changes in

the abstraction level. There are different approaches in the literature to implement a design vocabulary. *Objects and relations* are statements about physical and non-physical elements of the design. The knowledge representation method for this approach is a semantic network. Another approach is that of *objects and properties.* We can easily make statements about design objects (physical elements) and their properties. A property can be presented in terms of attributes and values.

We will discuss in the following the relation between theoretical frameworks and the choice of representation formalisms as well as their expressiveness.

2.2 Theoretical Frameworks

A theoretical framework is essential for any discourse. It defines the setting of a problem and the strategy to attack the arising questions. A theory is expressed using a certain language. In knowledge representation, languages tend to be formal, establishing certain axioms that in turn are used to prove clauses formulated to verify the correctness of the theory.

We will discuss theories that are formalized to various degrees. The *Pattern Language* is an informal language describing the architectural design process. *Exploration* is a more formal, model-based approach towards design knowledge representation. *Case-based reasoning* also exploits a formal representation of the design process, but is based on the case-study paradigm. Finally, *hypertext* is a semi-formal knowledge representation mechanism applicable to all domains.

2.2.1 The Pattern Language

Christopher Alexander developed an informal knowledge representation scheme for architectural design knowledge that he calls *"A Pattern Language"* [5]. To cite Alexander, a pattern

> *'describes a problem which occurs over and over in our environment, and then describes the core of the solution to that problem'.*

The idea of the Pattern Language is to decompose the overall design problem into a series of smaller problems (patterns) which can be solved individually and independent from each other. The solution to a given design problem is the synthesis of solutions found on the lowest level of decomposition.

To support the process of decomposition and synthesis, Alexander and his colleagues specified 253 patterns that describe design problems ranging from large scale (e.g., city) to small scale (e.g., stairs to sit). Each pattern expresses the relationship between a problem, the context of the problem, and the solution to this problem.

Furthermore, patterns refer to other patterns that might be of larger or smaller granularity. To solve a high-level problem like the design of an entire building, garden, or city, one has to start with the closest pattern, which will include references to related patterns on a lower abstraction level. This leads to an automatic decomposition of the high-level problem.

Alexander compares this design method to the usage of natural language. In natural language, words are combined to produce sentences. Grammar and semantics govern the construction process. By using *"The Timeless Way of Building"*, patterns are combined to form the description of buildings. Patterns of higher abstraction levels control the composition process of lower-level patterns. The Pattern Language is intended to enable all people to find meaningful combinations of patterns in the multitude of possible combinations.

The representation of all patterns follows a unified approach. First, a picture gives a typical physical example of the pattern. Next, links to related patterns of a higher abstraction level are provided. A short description of the problem as well as empirical investigations on that topic follow. Then, the solution to the problem is given. Finally, pointers to patterns of a lower abstraction level might show the necessity for further decomposition of the problem.

The Pattern Language is undoubtedly a powerful medium to express architectural design knowledge. Still, critics may apply to the rather inflexible structure of the knowledge representation system. Although Alexander encourages the user to delete and add patterns, this is hardly feasible in practice. The Pattern Language rests on a single coherent view of design. It is not the intention to render competing solutions for the same problem, nor inconsistent or contradicting theories on design. By altering some patterns, one might introduce inconsistency to the whole collection of patterns. While the design process needs a possibility for constant revision of objectives, goals, and solutions on all levels of abstraction, the Pattern Language grants an *a-priori* truth to its patterns (and solutions to these patterns) independently of the context.

Still, the Pattern Language is a promising approach to fill the gap between the language we use to communicate about architecture and the language of architecture itself. It can be used to visualize complex relationships between different aspects of architectural design. It is a good method to explicate one's own perception of architectural design. This conforms to Alexander's understanding that all architects have their own Pattern Language (at least without knowing it). Still there is his claim that at least some patterns are of "timeless" nature. These patterns are assumed to be part of the human nature and be the same in 500 years as the are today and have been 500 years ago.

2.2.2 Design as Exploration

Design is a process. How can we characterize and describe architecture's uncommon mode of action? Archea [7] describes it as puzzle-making. Differently from problem-solvers, architects cannot explicate desired effects *prior* to their realization through the design process. In problem-solving, explicit criteria to achieve the desired effects are stated before the course of action is chosen. This means, problem-solvers know exactly what they want to achieve. This knowledge can be used to test candidate solutions against the established criteria to find a fitting solution. By way of contrast, designers deal with vague and totally underconstrained problem descriptions. It is part of the design process to develop a more detailed description of the problem. Designers use kits of parts and sets of combinatorial rules to assemble these parts in a consistent way. The rule sets are not invented anew for every project — instead architects reuse rule systems of precedents, symbols, or metaphors and superimpose them on the design context at hand. It is this orchestration of several problem-solving processes that can be described as puzzle-making.

Smithers and Troxell [95] give a formal description of the design process. They build their model of the design process around the notion of *exploration*. Unlike computational search in a large problem space — a widely accepted view of the design process — exploration emphasizes a goal-oriented rather than goal-directed interpretation of the design process. What is the difference? While in the goal-directed approach the goal is the target, in the goal-oriented view the goal is rather an evaluation criterion.

Newell and Simon [76] influenced the development of computational search as a widely used paradigm for designing to a great extent. Following this paradigm, an initial state is transformed into a goal state by applying a series of problem space operators on the initial and the subsequent emerging problem states. The representation of the design process is a path through a graph from a start state to a goal state. Both start and goal state(s) have to be defined before the process can be initiated. This assumption contrasts with Smithers and Troxell's observation that desired goal states emerge throughout the design process and may change during the design activity.

It is exactly this change of the problem description throughout the design process that is well-supported in the exploration model of the design process. A design task starts with the description of a single design requirement. It is part of the design process to explore the space of possible designs to discover the structure of the solution space. While the design process is going on, different sets of requirements emerge. At the same time design specifications are acquired that partially meet the current set of requirement descriptions. The result of the design process is a consistent requirement description and an associated design specification. Design requirement and specification are two tightly coupled aspects of the same process. It is thus often impossible to decompose design problems and find independent so-

lutions to these subproblems.

The formal description of design as exploration process following Smithers and Troxell is a six-tuple (P_s, T, R_{i-f}, H, D_s, E_d), where

- P_s is the space of possible solutions defined in terms of precedents that are represented by completely defined states. The structure of the space is described by sets of properties, their associated value ranges, and relations between properties.
- T is a set of operations that compute new properties from existing ones. State descriptions are refined or inconsistencies are detected.
- R_{i-f} is an ordered set of non-empty sets of properties. Attributes within these sets represent the required design solution. R_i is the initial set, R_f the final set defining the found solution. In between are sets of intermediate requirements with the current one being referred to as R_c.
- H is the design history; it is a sequence of property sets and applied transformations.
- D_s is the set of solution states that possess properties specified by a R_c.
- E_d is the process that extends D_s by taking P_s, T, R_c, and H as input and outputting a series of transformations on R_c

2.2.3 Case-based Reasoning

Case-based reasoning [53, 46, 52] is reasoning from precedents, adapting old solutions to solve new problems, or retrieving old cases to explain aspects of the current problem. It is also used to critique new solutions, interpret a new situation or create an equitable solution to a new problem. The traditional reasoning cycle in artificial intelligence and cognitive psychology has been that the reasoner is handed a problem, and by composing and instantiating abstract operators, it is able to solve it. The case-based reasoning paradigm follows a different approach. In case-based reasoning, cases similar to the current situation are retrieved from memory, the best of these is selected and adapted to fit the current situation. After a problem is solved, the case memory is updated with the new experience; learning is integrated in the problem solving process. Successful as well as failed cases are stored in the memory. Successful cases can be re-used in the future. Failed cases can warn of potential difficulties and prevent the problem solver from making the same failure twice.

The case-based reasoning cycle consists of the following tasks: *Case retrieval* interacts with the data base to retrieve the fittest of the stored cases. The problem

of selecting adequate pieces of information out of a large base of information chunks is a common one in case-based reasoning and has been labeled the *indexing problem*. If the case-based reasoner is used to solve a problem, a *ballpark solution* to the new problem is proposed by extracting the solution from the retrieved cases. *Adaptation* of these solutions to fit the new situation follows and finally *criticism*, the process of evaluating the new solution. If the case-based reasoner is used to interpret solutions, the reasoner proposes a desired result that will be justified by finding arguments for the proposed situation. Finally, the found solution is criticized by creating hypothetical situations and trying the argument out on those. If a solution could be found, it gets stored to be used for further reasoning. Of course these steps are mostly iterative. Rarely the first retrieved solution fits the problem perfectly.

In design, certain problems can be defined by a set of constraints. There are problems that are underspecified with a rich set of possible solutions. Others are overconstrained, i.e., there exists no solution that fulfills all constrains. To solve these problems, constraints which can be compromised have to be relaxed. A solution is a concrete artifact that satisfies the constraint problem. In underconstrained problems, constraints point towards a solution but don't enforce a particular answer. Standard search methods might spend a lot of time in the huge search space before finding a solution. Also there are many answers that might be sufficient. Searching for a solution would request to decompose the problem and solve the problem in small pieces. Putting these pieces back together might violate the interaction between them.

By using cases, decomposing and recomposing of the resulting parts can be avoided. A case suggests an entire solution. Those parts of a case that do not fit the new case have to be adapted. Adapting old solutions is usually preferable to building new solutions from scratch. Engineering and architectural designs use these techniques of adapting and merging old solutions to produce new artifacts. In overconstrained problems, the original design specification must be altered to solve the problem. Solving these problems with conventional methods requests for relaxing constraints in many different ways. Conflicting goals have to be resolved and a compromise has to be derived that partially achieves the goals of both. In case-based reasoning the intention is to find a *close* solution in memory. Relatively easy changes have to be made on that solution. Because most design problems are large, often more than one case is necessary to solve the whole problem. One strategy is to use one case as framework for a solution where other cases provide details to be filled in. Decomposition and recomposition as well as large constraint satisfaction and relaxation problems are thereby avoided.

2.2.4 Hypertext

So far we have discussed approaches to model design knowledge that used methodologies which have been specifically developed for the domain of design. We will now

discuss a method that is not restricted to the domain of design — *hypertext*. It is interesting in the context of design because some of the design system examples that we will present in the following section use aspects of hypertext for various reasons. Those readers who are not familiar with hypertext will find an exhaustive review in the next chapter.

The hypertext technology can be used as semi-formal knowledge representation. Atomic information units (typically chunks of natural language text, pictures, video sequences, etc.) contain knowledge that is in general not interpreted by the computer. Knowledge is interpreted by the human user, applying her knowledge about the domain. In contrary to formal representations, hypertext representations are easier to develop, because the knowledge engineer does not have to formalize the knowledge. The knowledge is implicit stored in the information chunks. It is turned explicit through the interpretation process of the user. The interaction between the human user and the information base reveals the knowledge.

In this understanding hypertext is a less restrict method to represent knowledge. This is promising in the domain of design because most formalisms that intend to capture design knowledge are too restrictive in our understanding. The expressiveness of the design language is limited to fit the constraints of the formalism. A hypertext representation can draw on the human understanding while still making certain aspects understandable to the computer.

2.3 Implemented Design-Support Systems

After the discussion of theoretical frameworks on design knowledge, we will now investigate their applicability on system designs. HENRY is a system that emphasizes the informal aspect of knowledge representation. ARCHIE is an application of case-based reasoning in the domain of architecture. The system JANUS uses a formal design knowledge representation — *Design Critics* — and a hypertext component for user interaction. Finally, SEPIA is based on a semi-formal mechanism with the additional aspect of collaboration between users.

2.3.1 HENRY

HENRY [92] is a prototype system for documentation in VLSI design environments. The authors claim that a unique characteristic of VLSI design is that it requires simultaneous reasoning in correlated domains of representation at various levels of abstraction. This is also true for architectural design. It is thus interesting to analyze this approach to tackle these problems.

Silva and Katz introduce the notion of *active documents* for the purpose of design documentation. With the help of active documents, the authors hope to

integrate the process of design and the corresponding documentation to improve the understandability of the design. Design histories and derivations play an important role for the perception of the design process. Making this information explicitly accessible is expected to lead to a new quality in the design process. Viewing the design process as writing a non-linear active notebook, design tools become "agents" filling-in the blanks of the notebook. The design methodology is represented as a document template. It is essential to version the notebooks and find configurations and design alternatives of the design data. Also different views of the design data must be supported. As an example, some users might be interested in the functional behavior while others may be interested in the layout of the design objects.

Active documents use program executions to fill-in portions of the documents. Design objects can be accessed from within the documentation. Silva and Katz use the engineer's notebook metaphor to describe the interface of their design and documentation tool. Different pieces of design data and design tool executions are related by a hyperlinking mechanism. This allows the designer to interact with the document data. Using the metaphor of an engineering notebook, a design is a log of design diagrams and annotations. Previous states of the design process can be consulted by looking at previously written pages. The same object may have a different appearance in another design context. This is important for keeping track of objects in the design database. To visualize the change dependency between different objects of the same design, change dependency bars are introduced. These bars reflect the dependency to update related design objects. Anchors associate different design objects. Whenever the content of an anchor is changed, an associated action is executed. As a side effect, this action places a change dependency bar next to the anchor of the associated design object.

HENRY is implemented by assembling different already existing components: a CAD framework, a document processing system, and VLSI design tools. All these components are integrated by an electronic hypermedia notebook, called the *Navigator*. The Navigator is used to collect "notes" about the design. These notes can be of different types, such as external files, shell scripts, session logs, or simply text or a diagram. Also external tools can be accessed from within these notes. Hypermedia documentation of the design process will help designers build better and re-usable documentation for their designs. At the same time, the design notebook metaphor shall also support the design process per se. Drawbacks of this approach are the highly proprietary formats of the design descriptions which do not have the degree of portability of true notebooks. Furthermore, the HENRY approach lacks a clear idea how to structure the different types of notes within one design project. The authors give the requirement of keeping track of different design objects and their design dependencies during the design process, but they fail to give a technically concise description of a solution for this process.

2.3.2 ARCHIE

ARCHIE [38], a case-based design system prototype developed at Georgia Tech AI lab, was originally conceived as generic aiding tool for real-world design problem solving in the domain of architecture. In this spirit, the system should be able of the following tasks:

- help with elaborating stated problem specifications by suggesting additional relevant issues;
- proposing designs found in memory as solutions to these elaborated problems;
- using designs found in memory to point out potential problems to a user;
- help with adapting partially relevant design proposals to better fit new circumstances;
- evaluating how well a design met the elaborated goals under the current circumstances.

This user-centered, retrieval-oriented task aiding technology is a recent trend in the research community. The aim is to reflect the cognitive needs and abilities of human users engaged in a particular task. To fulfil this aim, knowledge has to be organized and presented to the user in an appropriate way. Domeshek and Kolodner [22] point out that this kind of systems tend to look like "clever hypermedia systems".

In order to support such a broad range of problem solving behavior in ARCHIE, cases were coded using a large number of goal, plan, and outcome descriptors. A slot filler notation has been used to represent these descriptors. ARCHIE's memory consisted of about twenty cases, all encoded in flat feature lists. Exactly this rather large feature set imposed problems on the system's input/output behavior. Users were confronted with big forms to be filled out. Unfortunately, these forms were not particularly meaningful to them. The collection of features was poorly structured. As a result the system did not communicate well with its users.

Although ARCHIE has been judged a failure by its creators, it was still a valuable step towards a better understanding of the domain of architectural design. As a result the group had compiled a vocabulary of relevant features needed to describe architectural design issues. Lessons learned could be used in the re-implementation of the system. ARCHIE-II [22] tries to circumvent the problems encountered in ARCHIE by keeping the system much simpler than the ambitious first prototype. The core task of the new system is to organize and present interesting pieces of entire building designs when they are relevant to the designer's task.

To achieve this goal, the ARCHIE-II design concentrated on the following issues:

- What are the useful parts of a case? To answer this question one has to consider the potential uses for experience in the context of design: proposing solutions, identifying pitfalls and opportunities, and suggesting evaluation criteria.
- How should case parts be presented? The level of detail, the modality of presentation (when to use graphics, when to use text), and the content of the case pieces have to be considered.

In ARCHIE-II, text and graphics are combined to tell short stories about particular lessons — positive as well as negative — learned in the design process. Three different classes of stories have been identified during the research pursuit. Point stories discuss features of a design and how these features contribute or undermine particular goals of the design. Interaction stories deliberate how features of a design interact with several goals of the design plan, perhaps even with contradicting influences. And finally cluster stories point out effects of related features in a case, but without discussing the relationship between these effects.

ARCHIE-II uses two different kinds of indices to retrieve appropriate chunks of cases. The descriptive index consists of a simple language for retrieving chunks of information under certain conditions. This can be compared to an index in the back of a book which is used to locate topics of interest in the contents of the book. The other type of index used in ARCHIE-II is the so-called relationship index. This kind of index is akin to pointers directly embedded in the text. It is used when it is possible to predict what sort of information is needed next in a particular context.

To conclude, ARCHIE and ARCHIE-II could not convince us from the applicability of case-based reasoning for complex design domains. While it proved valuable in domains with a restricted feature set, it seams to us to be a too simplistic approach to formalize a domain like architectural design by using case-based reasoning. Nevertheless, we think that certain aspects (e.g., the notion of "a case" as self-contained information unit, or the problem of retrieving cases that contain valuable information according to a given design problem) are of high importance for our problem. We think that we gain a lot of insight in the nature of the problem space by studying these aspects.

2.3.3 JANUS

Fischer and his colleagues [32] base their system on the notion of *design critics*. The act of critiquing existing knowledge fosters the understanding of design knowledge. Critique triggers further reflection on the design artifact. Through the critiquing process, designers gain a better understanding of the design space by evaluating different points of views of other "design agents" — either human or machine.

Fischer's team built several critiquing systems to prove the validity of the critiquing paradigm in design environments. All these systems are cooperative

problem-solving systems, in which the system helps the designers to find solutions instead of designing solutions for them. Two agents — a human designer and a computer-based critic — work together to solve the design problem. It is the role of the human designer to generate solutions. The critiquing system monitors the designer and produces a critique based on the analysis of the designer's solution. It is up to the designer whether to take those comments into account.

There are three different critiquing mechanisms — *generic, specific*, and *interpretive*. In all three styles, critic knowledge is captured in condition and action clauses. The critic rules are checked for compliance with the current design construction. If a contradiction is detected, the condition clause fires and the action part gives feedback to the designer. The differences between these three types of critic rules lie in the type of design knowledge they embody.

Generic critics [31] apply to all designs. They model design knowledge like standards or regulations or other kinds of domain knowledge. Generic critics reflect the analysis of the current design. Whenever a design unit is placed into the construction area it is checked against the generic critic rules. An example from the domain of kitchen design is that top cabinets should be placed 150cm above the floor. Whenever this rule is violated by the current design, the designer is notified.

Specific critics [71] reflect the designer's goals. To support specific critics each individual designer has to specify her design goals explicitly. This is done with the help of a specification component. In contrast to generic critics these rules apply just to one individual designer and will not be shared by others in general. In the kitchen design example this means that, e.g., for Smith's residential kitchen top cabinets should be placed at 80% of the cook's height.

Interpretive critics [97] focus on the interpretive character of the design process. Design situations can be interpreted differently according to personal background, experience, and concerns. As a consequence there cannot be a unique set of domain knowledge for all people and all interests. Designers work with particular perspectives in mind. At any time, designers may select a new perspective and evaluate the current design from this new point of view. The same critic rule can be evaluated differently in different perspectives. In the kitchen design example, perspectives might be an electrician's view, a plumber's view, a mortgage writer's view, or a city inspector's view. These views might share the same rules, but differ in their action parts.

Design critics are embedded in design environments. Design environments support designers in concurrently specifying a design problem and a design solution. This is done by providing the designer with a repository of design knowledge. HYDRA-KITCHEN [32], one of the design environment prototypes for kitchen design by Fischer et al., uses different components to model the knowledge involved in the design process: The construction component allows the user to design a kitchen with a set of design units. Critics are defined for each individual design unit as

well as for relationships between design units. The specification component is used to model the user's design goals. The specification of the design goals is expected to evolve over time rather than to be defined at the very beginning of the design task. The argumentative component contains design knowledge in the form of design issues, design answers, and arguments about decisions. It is used to explain the criticism raised by the system. A catalog component stores samples of previous design solutions. These are used to explain deficiencies noted by a critic.

The argumentative component has been first introduced in the JANUS system [30], a predecessor of HYDRA-KITCHEN. The Hypertext'89 paper investigates the applicability of using an issue-based hypertext system to model the argumentation for design decisions. One of the problems of information systems is to provide users with adequate access to information. In JANUS, critics are used to provide the designer with entry points into the hypertext system where the exact information to the design task at hand is stated. From this entry point, designers can use the browsing capabilities of the hypertext system to navigate through the issue base of the argumentative component. In this mechanism critics are used as hypertext activation agents. The user does not have to search for relevant information. The system displays the argumentative context of the current design problem whenever it detects a design principle violation.

While HYDRA-KITCHEN is an interesting research prototype, the question of scalability of this approach arises when thinking of "real world" problems in the domain of architecture. Architecture is not assembling of predefined design units. Furthermore, most architectural criticism cannot be formulated in rules but is developed in a discourse. The process of critiquing is a design process in itself. What seems to be promising in our view is the introduction of an argumentative hypertext component, where designers can explicate their design rationales. Thus, argumentation becomes part of the design process. Previous design examples play an important role in the design process and can be used to illustrate the design argumentation.

2.3.4 SEPIA

SEPIA (Structured Elicitation and Processing of Ideas for Authoring) — an active, knowledge-based authoring and idea processing tool for creating and revising hyperdocuments — is a research prototype from GMD-IPSI and was first presented at Hypertext'89 [98]. Since then the system has been under continuous development and has now evolved into a seamless desktop and meeting room collaboration system by integrating SEPIA with the DOLPHIN system [100].

What is interesting from the design perspective is that Streitz and his colleagues view the authoring process as a design problem solving process. To support this process one has to build tools based on a sound cognitive theory of writing. Results

of writing research have identified three closely related subproblems of authoring: the *content problem*, the *rhetorical problem*, and the *planning problem*. The content problem deals with structuring the author's knowledge about the domain of the intended document. Sequencing and reformulating of the information are part of the rhetorical problem whereas decisions about the structure and organization of the document are clearly part of the planning problem. These problems are solved by opportunistic activities, which are strongly interacting and building on each others' results. Design is the construction of artifacts that have to fulfill specific constraints and which might be composed from building blocks. To help the designers build their product — the text — the authoring environment is seen to be required to provide means for exploring the complex design space.

In SEPIA these problems are mapped into corresponding activity spaces: the content space, the rhetorical space, and the planning space. To support authors in structuring their arguments, an additional argumentation space is provided. Each activity space provides the author with task-specific design objects and related operations when working on the above described problems. During the design task, the author travels through these activity spaces in a non-linear sequence. There is a constant interaction between the various levels of text representation and related subproblems. The idea of *cognitive compatibility* is the driving force behind the system design of SEPIA. Easy mapping of internal structures to external task structures is the primary principle. Providing active support to the author in the form of feedback, advice, or guiding is the goal of the system. In the rhetorical space, SEPIA provides the author with a "construction kit" and a predefined link taxonomy. The planning space uses the IBIS — Issue Based Information System [85] — approach.

The latest development of SEPIA is the migration from a single-author system to a multi-author system [99]. A main feature of the authoring process is that in many cases it involves more than one person. Authors concurrently work on shared hyperdocuments. Three different modes of group work are supported: *individual*, *loosely-coupled*, and *tightly-coupled*. In individual mode, the system behaves like the previous single-author system. In loosely-coupled mode, authors are aware of other authors working on the same cluster of nodes, but do not interact directly with each other. This direct interaction is the domain of the tightly-coupled mode, where authors cooperate synchronously in conference-like "meetings". A smooth transition between these different modes of collaboration is provided within the system, i.e., authors can switch between the different modes of collaboration on demand.

The aspect of collaboration raises new demands. Whereas single-author systems naturally do not need to provide means for communication with others, this becomes a crucial aspect in a multi-user environment. As a consequence, users must have the ability to externalize certain aspects of their considerations. Without this, users will never be able to understand what others are doing. Information sharing is a key aspect of collaborative environments. Group communication and coordination systems contain formal knowledge about their domain to various extends.

2.4 Conclusion

The nature of design — a highly underconstrained process of generating systems and predicting their performance — is responsible for the complexity that is associated with every computational process that tries to tackle the design problem. This is the reason for the variety of approaches towards design representation and design reasoning — stemming from different theoretical frameworks.

The Pattern Language seems interesting to us because of its dedication to the domain of architecture and its holistic approach to architectural design knowledge representation. Unfortunately it is neither a formalism nor can it be implemented directly. The Pattern Language combines case analyses and design directives to a consistent (critics may call it dogmatic) view of architectural design. Case-based reasoning also uses previous design cases to model design knowledge. In contrast to the Pattern Language, it is a formal system. The choice of a cases' granularity of abstraction — which features are formally represented and which are not — is the crucial point for the feasibility of this method. Experience with implemented systems such as ARCHIE shows that this decision cannot be made at the very beginning once and for all effectively.

This conforms to the model of the design process as an exploration process. The finding of a proper formalization — a design process itself — favors an evolutionary approach. This is one of the strengths of hypertext. Hypertext support for knowledge acquisition is a prevalent topic in the literature [93, 74]. JANUS uses hypertext methods for its interface design. Hypertext is used to "explain" critiques to the user which are based on the formalized knowledge of JANUS. HENRY is a hypertext system approach without formal character. It rather provides an "open environment" for knowledge acquisition using the metaphor of an "active notebook". SEPIA on the other hand supports the user with an idea processing environment that uses also a formal representation of the domain knowledge. The emphasis in SEPIA lies on the analysis part of the design process. For the domain of architecture we think that the emphasis should shift to the synthesis part of the process. We think that this can be achieved by providing users with examples of previous designs and appropriate access structures.

It is our intention to build a system following the "spirit" of the Pattern Language as a holistic approach towards the modeling of architectural design knowledge. We think that hypertext is a promising technology for this endeavor. It allows a stepwise refinement of the granularity of formalization. It integrates features of knowledge representation mechanisms with "knowledge communication" capabilities. Man and machine can likewise be supported to various degrees. This will be shown in the next chapter, when we discuss the topic of hypertext in more detail.

Chapter 3

Hypertext and Hypermedia

After discussing the topic of design and what kind of requirements emerge for a design-supporting system we will now review the constraints of the underpinning technology — hypertext — and its influence on the design process. First, we introduce taxonomies of hypertext applications and the terminology that we use throughout this thesis. Next, we discuss two models of hypertext, the *Dexter Hypertext Reference Model* and the *World-Wide Web Model*. We chose these two because they represent the extreme ends of the rather broad spectrum of hypertext models. We then review the influence of hypertext methods on computer-based learning. We conclude by relating our work to the field of hypertext.

3.1 Taxonomies and Terminology

Hypertext — what is it all about. There exists no single definition. Instead, there are numerous "working definitions", emphasizing different aspects of the same idea. Ted Nelson, who coined the term "hypertext" in the late sixties, states

> *'by hypertext I mean* non-sequential *writing'* [73].

His definition stresses the literary quality of the new technology. In contrast to the well-known methodology of linear writing and reading, hypertext allows the reader to choose within the body of text and follow different branches. Bolder [11] and Landow [59] elaborate on the new rhetoric of hypertext in detail. Conklin [17] defines hypertext from a technology-oriented point of view:

> *'The concept of hypertext is quite simple: windows on the screen are associated with objects in a data base, and links are provided between these objects, both graphically and in the data base.'*

In his understanding, the technology of hypertext cuts across traditional boundaries of computer science by combining techniques as database methods and interface design.

You can find many more definitions in the literature [2, 3, 80, 96, 102]. What is common to all of these is the notion of *nodes* and *links*. Nodes and links are organized in a network structure (often referred to as "web"), where nodes resemble vertices and links edges of the network. Nodes are used to store "information chunks" (self-contained information units). Links model some kind of relationship between these units. By following links these relationships between information chunks can be explored. Support for tracing of links is essential to any hypertext system.

The term "hypermedia", also coined by Ted Nelson, emphasizes the diversity of media that can be used to encode information in nodes. While in hypertext information chunks are only portions of text, hypermedia supports different media types (e.g., video, audio, still images, animations, etc.) in nodes. Thus, hypermedia is a generalization of hypertext. The terms "hypertext" and "hypermedia" will be used interchangeable throughout this thesis.

The literature provides us with different ways to categorize hypermedia systems. Categories may help to evaluate the various properties of the application areas. The taxonomies, introduced at different times, mirror the state of the art in research at their time. We will review three of them in the following.

Conklin [17] introduced the first taxonomy of systems. His classes are:

- *Macro literary systems* — large on-line libraries with machine supported inter-document links.
- *Problem exploration tools* — tools to support unstructured thinking, problem solving, and designing.
- *Browsing systems* — small scale systems, where ease of use is crucial.
- *General hypertext technology* — systems built to experiment with the technology.

Leggett, Schnase, and Kacmar [60] introduced an application-oriented taxonomy of hypermedia systems consisting of five classes. They developed their taxonomy by analyzing and categorizing different systems that were available at that time.

- *Literary* — this class of systems emphasizes the relationships between information nodes. Links are very important and have a small granularity of referent.
- *Structural* — structural systems have fewer types of associations to model relationships between nodes. The ability to express the structure of the domain is the main issue of this type of systems.

- *Presentational* — similar to the structural class with an additional requirement of separate author and browser components.
- *Collaborative* — this class combines the properties of the literary and the structural class. Furthermore, systems are distributed and secure.
- *Explorative* — augments the collaborative class by adding a spatial-oriented user interface. This "space" allows the user to organize information elements and abstractions using a well-known metaphor.

Rada [84] structures the application area of hypertext in the following way:

- *Small-volume hypertext* — examines single documents and explicit links among their components as well as the browsing people do across these links.
- *Large-volume hypertext* — deals with links that exist among document data bases and the ability to store and retrieve these documents.
- *Collaborative hypertext* — emphasizes the computer-supported collaboration work (CSCW) [41] aspect of hypertext. Hypertext is used to support group activity.
- *Intelligent hypertext* — artificial intelligence methods are introduced to hypertext to let systems behave more knowledgeably.

The following paragraphs introduce some basic terms used throughout this thesis. Readers who are familiar with the terminology of hypertext may skip the remainder of this section. Readers who are not familiar with the research field of hypertext are also referred to introductory books by Nelson [73], Nielsen [77], Kuhlen (in German language) [58], and Rada [84]

We already presented the notions of *links* and *nodes*. Links and nodes are conceptually the basic building blocks of any hypertext system. Nevertheless, there is a great variety in the actual implementation of these concepts. Nodes differ in their granularity from single words to whole documents. Links may be unidirectional, bidirectional, typed, manually created, or computed on the fly. The representation of nodes and links on the screen as well as the interaction capabilities of the user with the system are issues related to the interface design of the system. *Anchors* are representations (e.g., buttons, highlighted text phrases, marked areas in drawings, etc.) of start and end points of links on the screen.

Browsing is the activity that hypertext users are engaged in when using the system. By browsing, users explore the information content of the hypertext. Usually hypertext systems encourage browsing by stimulating the user to follow links. Today, most hypertext systems employ the point-and-click paradigm for user interaction; information is just one click (of the mouse button) away. Carmel, Crawford,

and Chen [14] identify three basic cognitive classes of the browsing activity: (1) *search-oriented browsing* is scanning and reviewing of information in the context of a fixed task, (2) *review-browsing* is scanning and reviewing of information without a fixed goal, and (3) *scan-browsing* is scanning (without reviewing) of interesting information.

The notion of browsing is tightly coupled to the notion of *navigation* [78]. Navigating in hypertext is the path-finding part of the browsing activity. Navigation can become difficult in a rich information space and one can easily get *lost in hyperspace*. To avoid this anomalous state of getting lost, hypertext systems usually provide navigation facilities. Some of them are: *Overview diagrams*, which represent the network of the hyperspace graphically, *guided tours*, which are preset paths trough the network, and *interaction histories*, which allow to backtrack a path.

Finally, *authoring* is the activity of constructing a hypertext document. Both, nodes and links, have to be designed by the author. Some hypertext systems explicitly support the process of authoring [98]. The authoring process is often limited by the capabilities of the hypertext system. Documents can be authored by a single author or by a group of authors. If a group of authors is working together on a document we speak of collaborative environments.

3.2 Hypermedia Frameworks

In the following we present two frameworks for hypertext applications. We chose these two because we think that they represent two rather different philosophies in their understanding of hypertext. We think that the first — the Dexter Hypertext Reference Model — results from a very deliberate "top-down" approach, while the second — the World-Wide Web Model — is a rather straight-forward "bottom-up" proposal. Nevertheless, in terms of applications, the latter is the "big winner" with thousands of existing installations. With the dissemination of a public domain code library for the Dexter model, which has been announced recently, we will perhaps see more Dexter-compliant applications in the near future.

3.2.1 The Dexter Hypertext Reference Model

The original Dexter Hypertext Reference Model [45, 42] is the result of a series of workshops between October 1988 and July 1990. The Dexter reference model tries to integrate and formalize aspects found in different hypertext systems. Among them such well-known systems as Intermedia [105], KMS [3], NoteCards [44], or Augment [27]. All major developments, at least from research within the United States, were presented at the workshops and their architectural abstraction mechanisms taken into account for inclusion in the standard. The goal of the model

is to provide researchers with a standard for comparing systems and developing interchange formats.

The Dexter model introduced three conceptual layers for hypertext systems: the *run-time* layer, the *storage* layer, and the *within-component* layer. An important aspect of the model is the definition of interfaces between these layers, namely the *presentation specifications* between the run-time layer and the storage layer and the *anchoring interface* between the storage layer and the within-component layer. The focus of the model is on the storage layer, which models the node/link network of the hypermedia system.

The notion of component replaces the weakly defined concept of nodes in the Dexter model. Components are the basic data-objects provided in the storage layer and interconnected by relational links. The storage layer does not attempt to model the structure within components. Components are treated as generic data containers. The model of the structure of components is open to other standards such as ODA [79], SGML [39], etc. Still, composite components are introduced to provide hierarchical structuring of atomic components.

A special type of component is the link component. Because of the generic definition of components, the Dexter model supports computed as well as static links. Links can be single or bi-directional as well as multiheaded. In addition, links can be endpoints of links. Typing is supported through attributes that are added to the link component. To address specific locations within a component (which are not defined in the Dexter model) the link component relies on the anchoring interface between storage layer and the within-component layer. Anchoring provides a mechanism for addressing specific locations (span-to-span links) within all component types known to the system. Anchors consist of two parts: an anchor identifier, which is unique across the universe of discourse (not just within the scope of one hypertext), and an anchor value, which defines the actual region of the anchor within the component.

The run-time layer captures the dynamic aspects of hypertext systems. These include especially the presentation of hypertext components and the interaction processes with the user. The way in which a component is presented to the user is encoded in the *Presentation Specifications*. Because links are also components, link behavior can be customized as well. The same link may bring up different "modes" of the destination component depending on the status of the user who invoked the link. All link specifiers indicate a direction of the link of the form "FROM", "TO", "BIDIRECT", or "NONE". Still, the semantics of the links are not explicitly defined.

In addition to the data model, Dexter also specifies a set of operations on this data model. The resolver function is responsible for resolving a component's unique ID given a component specification. This indirect addressing is used to cope with dynamic changes of component IDs, i.e., when components are edited. In the simplest case the component specification is the component ID, in which case the resolver function is the identity function. The component's ID is then fed to the

component accessor function, which is responsible for mapping that ID into the data object assigned that ID. Furthermore, there are functions for determining the interconnectivity of the network by *LinksTo* and *LinksToAnchor* operations. The first resolves all links to a given component whereas the latter resolves all links referring to a given anchor within a component.

The original Dexter Hypertext Reference Model has been presented by Halasz and Schwartz in January 1990 at the NIST (National Institute of Standards and Technology, Gaithersburg, Md) workshop on hypertext. Since that time, several extensions to the original model have been proposed. They are answers to shortcomings of the original model that got revealed through practical use. We will discuss two of these extensions in the following.

The Amsterdam Hypermedia Model (AHM) [47] tries to tackle the complex timing and presentation relationships found in multimedia presentations. In particular, the AHM extends the Dexter model by adding high-level presentation attributes and link context. The AHM follows in its multimedia aspects the CMIF model. It allows authors to specify how individual pieces of information relate to each other over a period of time. This is a non trivial problem; dynamic and static objects fetched from distributed sources have to be synchronized. The model has to be capable of defining requirements for links, time, and global representation semantics in a hypermedia system.

There are three approaches to synchronization within hypermedia: the *hidden structure* approach, the *separate structure* and the *composite structure* approach. The hidden structure approach places all information about the context and time-based relationships within the content portion of a component. Collections of different media are synchronized by internal techniques of a component. This approach does not change the hypermedia model but also does not scale very well when media from different servers are combined. The separate structure approach is quite the opposite to the hidden structure approach. Here, each piece of multimedia information is an individual component. Collections of media are realized by multiheaded links. It is the responsibility to start all objects simultaneously, or according to their timing-information. It is clear that this can become a significant problem. Finally, the composite structure approach groups a limited number of media objects together. This combines the advantages of the previous approaches. While complicated synchronization is avoided outside of components, collections are less complex than in the hidden structure approach.

How does the AHM extend the definition of the Dexter component to capture these new requirements of time-based synchronization? The extension to the atomic component is an expanded presentation information section. Some of that information is used to model time-related aspects whereas others are used as high-level presentation attributes. The presentation structure is encoded in composite components. Timing offsets among the children of a component as well as composite

type (either parallel or choice, meaning either all children or only one child gets displayed) are added to the composite specification. Synchronization arcs are used to define the fine-grained temporal relations within components.

AHM also defines the notion of *link context*. Dexter does not define the behavior of components when a link is followed out of that component or affected by a linking operation. The source context is that portion of components that are affected by initiating a link whereas the destination context is that part of the hypermedia presentation that is activated on arriving at the destination. The context mechanism is used to allow the run-time environment determine appropriate display operations, i.e., whether to reload resources or not. Another abstraction of presentation attributes in AHM are channels. Channels are abstract output devices, which are used to define global characteristics for a certain media type such as volume for an audio channel. During run-time these abstract channels are mapped onto physical output devices.

The timing constructs added by AHM to the original Dexter model allow the specification of time-based behavior of a document. The formal description of this behavior can be transformed and represented in any form such as HyTime [75]. This allows the interchange of documents between different applications.

A second extension to the Dexter Hypertext Reference Model comes from Aarhus University in Denmark. People there are working on computer-supported cooperative work (CSCW) aspects encountered in large engineering projects [42]. Cooperative work raises several new requirements for hypermedia applications such as explicit communication and coordination between team workers as well as implicit coordination through shared materials. To accomplish these, new notions have to be introduced to the model such as shared databases, event notification, open access from different platforms and from different, already developed applications.

The kind of computer support for cooperative work depends on the type of cooperation. The Aarhus team has identified six different modes of cooperation, ranging from separate to fully synchronous sessions. In the mode of separate responsibilities, the material is divided into disjoint parts. The material can only be inspected by other users, but not altered. Turn taking is quite similar to the first mode, but allows different users to work on the same material while no more than one user is working at a time. This mode requires support for cooperation to coordinate work over time. In dynamic exchange, users work at the same time on different objects, but are able to transfer locks on these objects from one user to the other. The mode of alternative versions allows each user to develop her own version of a part. Different versions of parts have to be merged later. Finally, mutual sessions allow two or more users to work on the same material synchronously, using direct communication channels, whereas in fully synchronous sessions, all users share exactly the same view of the shared material.

The first three modes are directly supported by the hypermedia model. The

asynchronous cooperation can be provided by means of annotation or other methods of creating awareness among users about the use of the shared material. Mode four needs the support of the storage layer to enable versioning of objects. This is an ongoing research topic in database research and will be solvable on the database level. The fully synchronous modes need some additional means of communication beyond the point that is currently available in the Dexter model. Mode five and six differ mainly in whether a shared view of objects is provided or not. Whereas without shared view the synchronous cooperation is restricted to shared commitment of changes on objects, the shared view option would allow all users to see exactly the same state of an object. Changes on that object would immediately propagate to all users of that session.

To prove their concepts for developing a cooperative hypermedia framework, the DeVise Hypermedia Architecture (DHM) has been implemented. Based on a distributed object system, the DHM introduces several client and server processes compliant to the Dexter model. These processes may run on different platforms and communicate over a network with each other. The editor process is end-user oriented and integrates hypermedia facilities into conventional editors like text editors, video or audio editors, or hypermedia browsers. Each editor is associated to a specific data object, i.e., to a specific type of component. An editor's functionality mainly belongs to the Dexter within-component layer. Editors communicate to run-time processes to exchange anchor values with the storage layer. The run-time process is a server for the editor process and is itself served by the OODB server process, which provides the physical storage facilities. The run-time process instantiates the data objects of the physical storage layer at run-time and provides the basic link service independent of the editor type. Thus, it functions as a kind of mediator between the editors and the hypermedia resources.

This kind of open architecture allows to make all kinds of editors, even already existing ones, DHM compliant. To integrate new applications with DHM, a component type, a presentation specification, an anchor specification, and a linkMarker specification have to be defined for the new component type. Generic classes of the DHM framework can be used as basis and have to be specialized to fit the new needs. Furthermore, the editor must be capable of communicating with the run-time process. The DHM framework supports the asynchronous cooperation modes one to three described earlier. This includes notification of changes on objects as well as transfer of responsibility for objects, i.e., changing locks on parts of the hypermedia material. Event notification is directly implemented in the OODB system. Users can subscribe to certain events occurring on the shared material. Events propagate from the OODB system to the tun-time processes, which in turn propagate the notification to their clients (editors).

3.2.2 The World-Wide Web Model

The World-Wide Web (W3, the Web) project was initiated by Tim Berners-Lee in 1990 while we was working for CERN in Geneva, Switzerland. He envisioned a system that would allow all kinds of collaborators work together on a common project. The idea has been inspired by the settings of CERN, where a lot of scientist and engineers, often scattered throughout different institutes, work together on large-scale projects in the field of high-energy physics. To support these people in their work, they should be able to generate documents which could be accessed from remote sites in a seamless way.

What is the basic set of definitions that must be provided to implement such a system? First, all items that can be reached in this information universe need a unique identifier. The W3 project realized this by introducing Universal Resource Identifiers (URIs), which implement a unique address schema. URIs are strings that encode a certain protocol, a certain network address, and the local location of an object of the information universe together with certain attributes to access the object. Different interpretations of these strings through different protocols guarantee extensibility for future applications. Today, there are URIs for Internet news services (NNTP protocol), for FTP archives, for WAIS data bases, for email addresses, etc.

Additionally, a common network protocol, namely the Hypertext Transfer Protocol (HTTP), allows native W3 servers to communicate with clients with high performance. Actually not hypertext per se is transferred, but information that enables hypertext-like jumps in an efficient way. The actual data may be plain text, pictures, video, or any other kind of information. HTTP is used to transfer data in many different formats. To achieve this, the client sends a request with an ordered list of formats it can handle. The server answers in an appropriate format. New formats can be added to the protocol at any time.

HTTP is stateless. That means, a connection between server and client is just held for the duration of one operation. Previous operations of the same client are not taken into account when evaluating the URI of any object. HTTP requests are composed of a method (an operation code) and an URI. Sample methods are “GET”, “PUT”, or “POST”, which are used for different purposes. The GET method is used to request objects. The PUT method is defined for front-end update, whereas the POST method attaches new documents to already existing ones. Servers are configurable to allow just a certain set of methods on certain objects. Thus the ”use” of documents can be restricted to meet certain needs.

To be able to exchange hypertext between servers and clients, W3 defines a basic interchange language HTML (hypertext markup language). HTML is compliant to the SGML standard. Still, the language is simple enough to be easily used by people to construct documents. HTML defines a set of constructs such as headings, lists, menus, etc., that can be used to produce on-line documents. HTML documents

may also hold portions of other kinds of media, such as graphics, video, or audio. HTML also defines link elements that point to parts of other documents. Whenever a link is selected by clicking on it, the client retrieves the referenced document from a server. Links can point to data of any format. Whenever the local site is capable of handling the data, it will make use of them. When not, it will ignore them. A current extension of HTML is HTML+, which includes more sophisticated features.

Started in 1990, the web project was introduced to the particle physics world in the Christmas 1991 CERN Computer Newsletter. Since that time, the access to the CERN server doubled every four months. This is a good indicator of the growing interest in the W3 project from all over the world. The same picture shows the statistics of traffic across the NSF backbone in the U.S.: Whereas in the beginning of 1993, HTTP traffic made up approximately 100 Megabytes, one year later, in the beginning of 1994, HTTP traffic counted for hundreds of Gigabytes. In the same time, the number of registered servers rose from below 50 to over 500. Also two conferences on W3 have been held in 1994, one in Genevea, and one in Chicago, with another one scheduled for next year in Darmstadt.

Originally intended for use within the different laboratories of CERN, the web has spread all over the world. There are servers out there providing information about particular institutes or particular fields, ranging from art to entertainment. An increasing number of servers are running in the forefield of commerce: as information providers companies (ranging from the "big players" in the information industries to small start-up companies) use the web to advertise their products or services. An emerging standard for charging the use of services will give the commercial use an even higher appeal. The traditional academic Internet user community follows this development with a rather skeptical view. The future will see if the "infobahn" infrastructure can scale accordingly to the demand of "commercial users".

Putting software together with the proposed standards for formats, protocols, etc., into the public domain made this tremendous success — in terms of users — possible. CERN provided the W3 developer community with the "libwww" code library. Based on this library, several institutions developed client software for various platforms. Among them NCSA's *Mosaic* browser software had probably the most influence on the rapid spread of W3. First developed for UNIX workstations, it soon became also available for Macintosh and PC compatible machines. This provided users throughout the computer field with a consistent user interface for the Web independent of the platform they used. The browser provides access to servers of all types using different protocols without explicit recognition by the user. The whole information available on the Internet becomes a unified information universe. The simple point-and-click paradigm enables even inexperienced computer users to "surf" the "information wave".

The aspect of decentralizing information sources and still being able to offer a central "access" point is the advantage of the linking paradigm. Via World-Wide

Web, distributed information providers can be brought together by setting up a "homepage" with pointers to several locations. Local information moderators are responsible for keeping their information up-to-date. This strategy enables a group of people to work together on a topic by adding their pieces of the whole picture. Moreover, existing information sources can be re-used and can be brought into new contexts by interlinking them in various webs with different purposes.

Is the Web the answer to all questions for hypermedia developers? Of course not. The "simplicity" of the overall approach induces — at least in its current state — some major constraints on the expressiveness of the resulting hypertext. For example, links are by definition uni-directional. There is no possibility to find documents that point on a certain document given that document. This hardens the process of maintaining documents. Because the physical location of documents is encoded in the current form of URIs, dangling links are results from moving resources from one place to another. There is no method to inform all links of the change in the destination address.

The future will show if extensions to the original proposal can overcome these limitations. One of these is the implementation of a name service that will allow referencing by name independent of the location. Furthermore, editors that actually allow changing rather than just viewing documents are needed to support the collaborative environment envisioned in the original proposal. Documents should also include some machine-readable semantic information to support navigation within and retrieving of interesting documents. We will see if the original W3 design is flexible enough to scale with these new requirements.

3.3 Hypermedia and Learning

Using computers as "teaching machines" has a long tradition (see [54] for a review of intelligent instructional systems). A lot of computer-aided learning (CAL) software is based upon the psychology of behaviorism, introduced by B.F. Skinner [94]. This rather outdated theory of learning reflects the assumption that learning consists of exhibiting appropriate behavior. Students are asked a series of short-answer questions. Giving the right answer let them proceed to the next question. Giving the wrong respond triggers the display of the correct answer. This kind of immediate feedback is very important for the conditioning process. Studies have shown that this approach is not very effective in a realistic setting. Students get never explained what's the source of their mistakes. Nor do they learn complex relations within their domain of studies. For behaviorists learning is reduced to conditioning. It is not important what students think as long as they react correctly to a given stimulus.

In contrast today's systems emphasize the active role of the learner. Learning is seen

> *'as a process of change in the way that the individual views something, rather than the mere aggregation of facts',*

as Peter Whalley states in [103]. The learner is put into some position of "control" regarding the strategy of knowledge acquisition. This is a rather process-oriented view of learning compared to the goal-oriented *instruction* approach that is based on pre-planning and formative evaluation. In contrast, Romiszowski [86] points out that

> *'there is much evidence to suggest that learners, when free to select their own learning strategies, do not always select wisely.'*

Hypermedia seems to be the ideal technology to implement interactive learning environments. Learning will occur simply from the solely use of any hypermedia document by implicit, incidental learning [68]. The question is how good hypermedia supports explicit learning through teaching. To answer this question we have to define the term "learning". Jonassen and Grabinger [49] define learning as the creation or reorganization of the learner's knowledge structures. In this understanding, hypermedia replicates knowledge structures that are integrated into the users cognitive structures through the process of learning. In contrast, Whalley [103] states that simple web structures of hypertext are not complex enough to model human semantic knowledge structures. Up to now there is no evaluation of this argument on the basis of empirical studies.

There are different pedagogical assumptions for the use of hypermedia in educational environments in the literature. Duffy and Knuth [23] propose the following taxonomy of uses of hypermedia in learning:

- *Exploring a large data base.* Assuming hypermedia as a large data base of nodes and links coupled with machine supported movement through this information, the main pedagogical benefit of this use of hypermedia in learning is the possibility to assist students in exploring relations. Learners get better access to information and to the structure of the information. The goal is to provide students with information on their finger tips.

- *Accessing elaborations on core information.* This use resembles mostly the aim of classical Computer-Based Training (CBT) packages. The emphasis lies on efficient learning. Students have the ability to ask for a variety of explanations on the core information that they have to understand.

- *Operating on a database.* Additionally to exploring the database, tools are made available that enable the user to manipulate the data base. The data base is not changed but rather personalized for the different users. Tools are made available on demand and used to abstract, compile, annotate, or juxtapose information nodes in the hypermedia network.

- *Building a database*. Compared to the previous uses the emphasis here is on extending the database. Learners are in the responsibility to author information. This raises new questions as how to keep the database in a consistent state, especially in a collaborative environment. Nevertheless, this approach is the most active one from the learner's point of view.

3.4 Conclusion

The analysis conducted in this chapter shows that hypertext methods are appropriate for both, learning and research aspects of architectural design investigations. We think that this supports our decision to base our approach towards modeling a design environment in the setting of academic research and education on hypertext techniques. The key aspects that hypertext provides for the solution of our problem are the support of evolving information structures, the scalability from single-user to work group environments, and the "open" character of the information presentation methodology.

Speaking in Conklin's taxonomy (see Section 3.1) we think that our intended system will qualify as *problem exploration tool* — a tool that supports unstructured thinking, problem solving, and designing. In the taxonomy of Leggett et al. (see also Section 3.1), the *explorative* class of systems promises to subsume the capabilities needed for our application — a collaborative environment that allows users to organize information elements and abstractions. If we look at the taxonomy for uses of hypermedia in learning by Duffy and Knuth (see Section 3.3) *Building a data base* is the most active class of systems from the learner's point of view. We think that adapting this notion for our approach will integrate the actual design activity with the hypertext interaction and learning activities.

The choice of the hypertext model is very important for the design of the system. The model often restricts the functionality of the resulting application. In the next chapter we will review this problem in detail when we talk about our experience with different models in the course of our prototype developments.

Chapter 4

Prototype Implementations

In the following we present our experience with prototype systems that we developed in the course of this study. We begin this chapter by describing our didactic framework for a collaborative environment in the realm of architectural design education. Subsequently, we discuss HySAT, the first prototype system that we have built based on our previous design considerations. After evaluating HySAT, we continue our discourse with presenting HySAT-W3, the successor system. Finally, we explain further needs and future plans.

4.1 Didactic Conception

Teaching architecture is not primarily an instructional process but rather a process of interaction and experience. Critique and discussion play an important role in this context. In the design studio, students learn by *doing*. The method of learning-by-doing is complemented by additional courses, where students learn about design methodologies, history of art and architecture, construction, functional requirements, etc. We want to investigate if we can integrate these rather contrasting learning-methodologies by providing the student with an environment where "experience" represented by design cases is tightly coupled with corresponding theoretical ideas.

To build a learning environment in the field of architecture using hypermedia as technology we have to embed the capabilities of hypermedia into a proper didactic conception. It is the active, exploration-oriented process of designing that sets this application area apart from others. Hypermedia has been used to construct systems that basically mimic "school books", leaving the student with a very passive experience. In such systems the interaction is reduced to "turn the page" functionality, presenting the student information in a linear fashion screen after screen.

The emphasis of our system lies on support of the design process itself. This can be achieved by engaging the user in active construction and transformation of the hyperbase, using the same metaphor of *exploration* for designing as well as for interacting with the system. Learning by exploring emphasizes the conversational format. Students are encouraged to take over control and direct the learning process by asking questions and finding answers within the system's base of expertise. This approach necessitates a proper organization of the knowledge intended to be transferred to the learner. The knowledge is useless if users cannot find what they are looking for.

As we have already pointed out in Section 2.2.2, creative architectural design is comparable to "puzzle-making" . There is no predefined set of design objectives, but rather an almost infinite number of possible ones. It is part of the design task to investigate the relationships between the different constraints and find a set of objectives that can be kept consistent under the given design problem. Architects differ not only in their choice of solutions but moreover in their choice of problems which they exploit within a given design program. Smithers and Troxell presented a formal description of the design process as exploration process (see Section 2.2.2 for an in-depth discussion of this formalism). In this formalism, an initially incomplete set of requirements is transformed into a final set of properties which defines the solution. During the design process, new requirements emerge as properties of the space of possible solutions change. This model of the design process emphasizes the structure of the problem space, defined by properties and operations on these properties rather than the "search problem" of former formalisms.

Teaching students of architecture should foster this change of perspective from mere "problem solving" to "problem finding" as part of the overall problem solving task. Of course, this paradigm shift is especially difficult for novice students having enough troubles to generate solutions to the most obvious problems related to a design task. The ability of problem finding requires a huge knowledge about the solution space. Exactly this knowledge can be supplied with the help of design precedents coupled with theoretical explanations. Design cases hold information on sets of design objectives and corresponding solution patterns. This information can be abstracted from these concrete examples and applied to new design problems. Examples of previous designs are not to be copied, but treated as artifacts of more abstract, theoretical concepts.

Conforming to the theory of case-based learning, cases aid learners to memorize facts that would be difficult to integrate into memory without context. Case-based learning draws on the idea that teachers are good storytellers. Storytelling exploits people's powerful natural learning mechanisms which are used to transfer experiences from person to person. Cases are repositories of experiences which are sources of knowledge to the learner. Many different access methods to these cases are needed to expose all the different knowledge that is inherent in each single case.

If we link design precedents to corresponding design theories we can support this dynamic process of solution space exploration, where new objectives emerge in the context of changing interpretations. The exploration process will lead back and forth between these different representations of design knowledge; and it is up to the users which trail to follow. Users can start with a set of objectives and — by constantly refining this set throughout the exploration process — end up with a complete different set of design objectives closer to their task at hand.

As an introduction to this conceptual framework students of architecture at the Building Theory and Design Department have to analyze design cases, applying their theoretical knowledge about design rules and methods. Formal, constructional, and functional aspects of a building are investigated. The result is a monograph about this building, containing text, pictures, plans, and schematic drawings. Our project is concerned with transferring these monographs into electronic documents. The main benefit of this new form of presentation is that students will work in an integrated information space, evolving over time, growing with the number of documents. While authoring their document, students can draw on previously done work by setting links to this material. In fact, integrating their document in the overall document space will be a major objective of their work.

Compared to the traditional paper work, electronic documents and electronic references have the main benefit of being immediately available on request. A seamless navigation between information items of different documents is possible. The whole document space is a single universe that can be restructured form all possible points of views. There is no single linear conception that is predefined once and for all. Besides this new understanding of documents, we expect a higher expressive power of these documents through inclusion of new media (e.g., video, CAD-drawings, audio) as well as a better accessibility both physically (e.g., over distributed network environments) and conceptually (e.g., searching and other retrieval techniques).

4.2 HySAT

Our first prototype HySAT (Hypertext System for Architectural Typology) was implemented as a HyperCard [40] application. HyperCard has been the first widespread "hypertext-like" program on Apple computers. This is due to the fact that in its first release HyperCard came for free with every purchased computer system. Its ease of use and appealing graphical capabilities soon earned it a huge user community. A lot of applications together with a growing range of external functions and extensions appeared in the public domain.

Although falling short in a number of aspects compared to "real" hypertext systems (as we will discuss later), HyperCard provides programmers and hypertext authors with an interesting environment. It has proved useful for rapid prototyping

(mainly because of its sophisticated user-interface building features) to evaluate first approaches to a problem, especially from the usability point of view. We were able to build a first prototype in a very short time and use this immediately to gain feedback from prospective users.

HyperCard comes with its own programming language, HyperTalk. HyperTalk is advertised as object-oriented language, but it just implements message passing between predefined classes of objects (buttons, fields, cards, backgrounds, and stacks) along a predefined inheritance path. HyperCard uses the metaphor of stacks of cards. Within a stack, cards share backgrounds (same type of cards) and contain fields (for text) and buttons (for actions). A card in HyperCard corresponds to a hypertext node and contains a certain amount of information. Buttons are used as anchors for actions, e.g., to follow a link. Fields are the main data containers. Buttons and fields may belong to a certain card, or may be shared by a number of cards via a common background. This composition defines the static organization of cards. The content of fields belongs to the card (if not explicitly shared between cards).

Messages trigger certain actions. Messages are generated whenever an appropriate event occurs. Events may be generated by certain user actions, or by the system. Events are passed according to the inheritance path from object to object. Whenever an object has a matching event handler defined it will react to the message by executing the event script (a HyperTalk program). New messages can be defined by the programmer and can be used to communicate with other objects. Whenever a message "reaches" the HyperCard program — the last instance of the inheritance path — without being processed before, HyperCard will generate a corresponding error message. Programming in HyperCard means to define a certain set of objects, a set of messages, and corresponding event handlers.

Our emphasis in the first prototype lay on the implementation of a case base. This case base should be able to capture the important information about buildings of interest. Students should be able to use that tool to produce interesting analyses of already existing buildings. The resulting case base can then be used as a knowledge base for future design tasks.

The basic information unit in HyperCard is a card. Cards of different stacks can be viewed simultaneously on the screen. We decided to divide the presentation of the material into three classes: one for text, one for pictures, and one for drawings (maps). This will enable users to view different representations of the same building concurrently, i.e., to view the corresponding picture to a certain passage in the text part. We designed three different stack prototypes, a text, a picture, and a plan stack. All of them equipped with functions tailored to the representation media, e.g., measuring of distances in the plan stack. Figure 4.1 shows a screenshot of the whole environment. In Figure 4.2 you can see the different types of stacks in more detail.

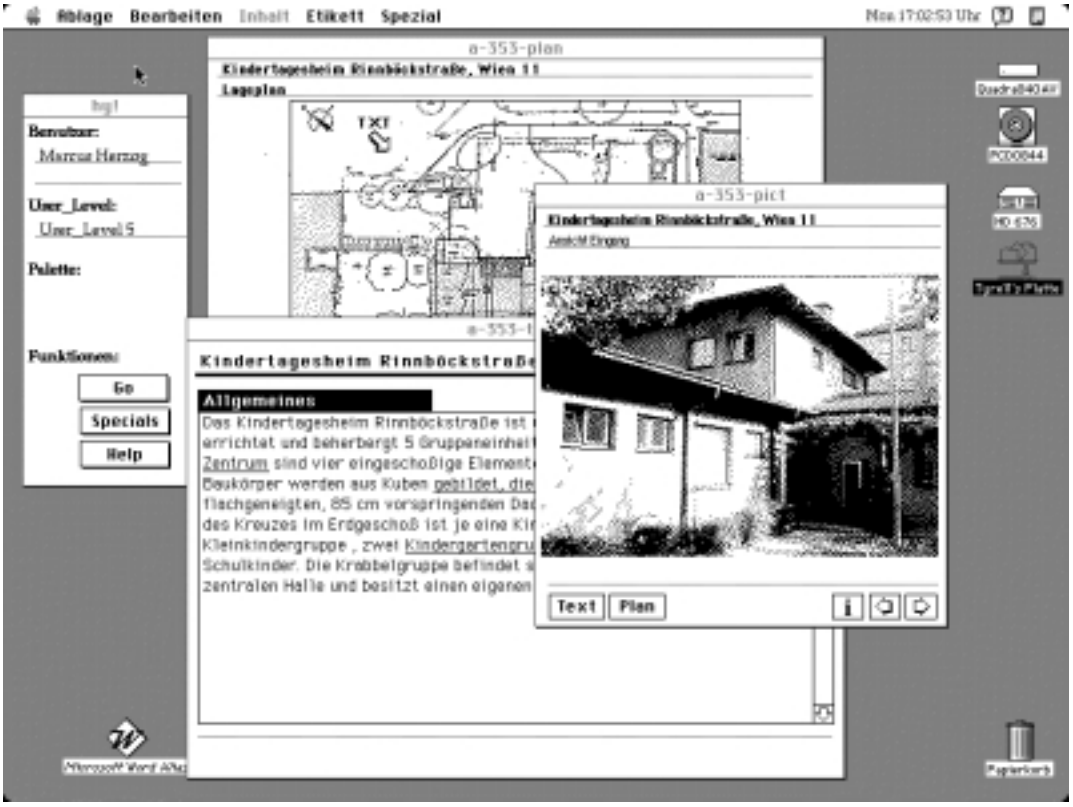

Figure 4.1: Screenshot of the HySAT environment

The main point of hypertext representations of information is the ability to associate chunks of information through the use of links. As HyperCard does not provide a system tool that allows users to generate links between arbitrary chunks of information, we implemented our own link mechanism. This tool, called the *linker* (see Figure 4.3), enables users to select some information (i.e., a text chunk) and link it to some other point within the universe of stacks. Links are bi- or unidirectional and typed. Different types of links have different purposes and help the user to predict the outcome of a "follow-link" operation, i.e., what kind of media the referenced material will be like.

During the design course, students were asked to "fill" these stack templates with their information. The use of templates imposed a certain structure on their compositions. Users, who will later browse through these stacks looking for some information, will know where to look because of the unified appearance. The combination of a predefined structure together with the simple user interaction model of point-and-click helps users to find their way without a long learning phase. Even inexperienced users can handle that kind of system after a short amount of time.

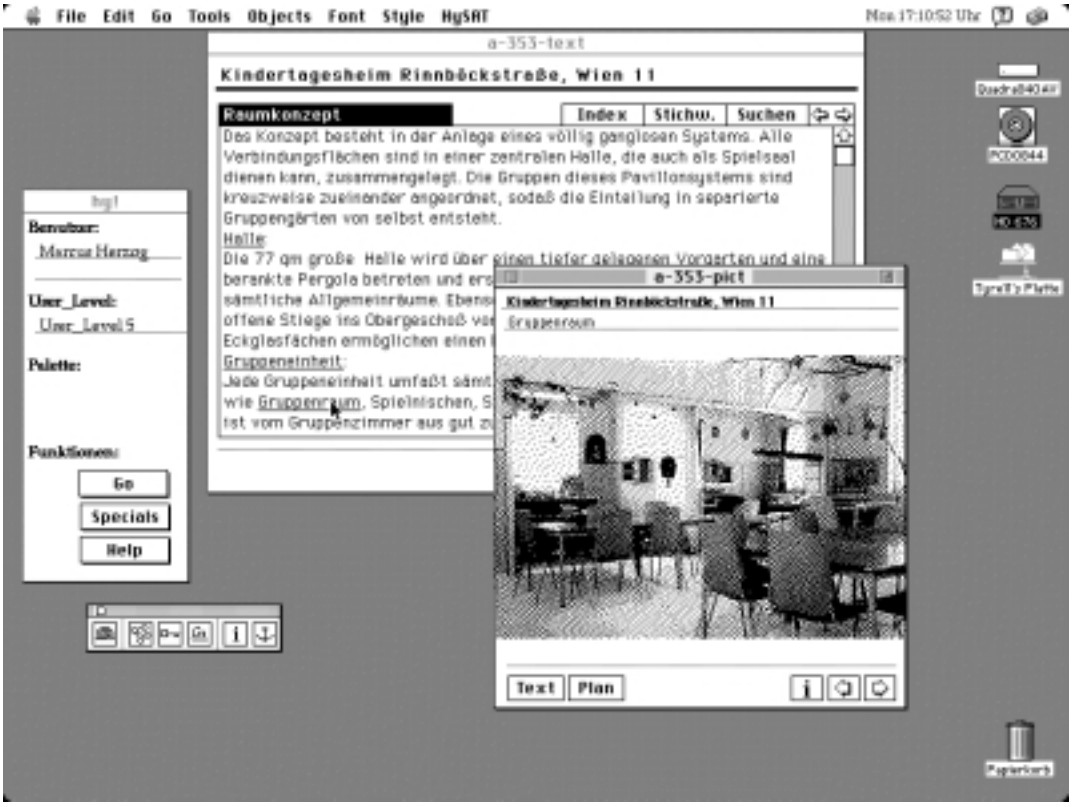

Figure 4.2: Pointer clicking at a link to a picture

4.3 Evaluation

Although HySAT proved viable from the conceptual point of view, the implementation in HyperCard has some major drawbacks that we found out during the on-going project. The main drawback of HyperCard is that it does not divide data from programs. That means, the content of fields, buttons, and scripts cannot be extracted (without a huge effort) from stacks and imported into some other stack. Because a stack is the definition of the user-interface, the functionality, and the appearance, there is no easy way to change already existing stacks.

Our problem is that we use a large number of similar stacks, but with different contents. Whenever we would like to enhance the functionality of the existing case base, we have to check automatically all stacks and change the affected parts. Still, sometimes this may not be possible because we might lose some customized information (i.e., added links, etc.). To overcome this problem, we tried to separate the user-interface from the actual functionality of stacks. We use a HyperCard feature that allows stacks to use message handlers of other stacks. Based on this program-

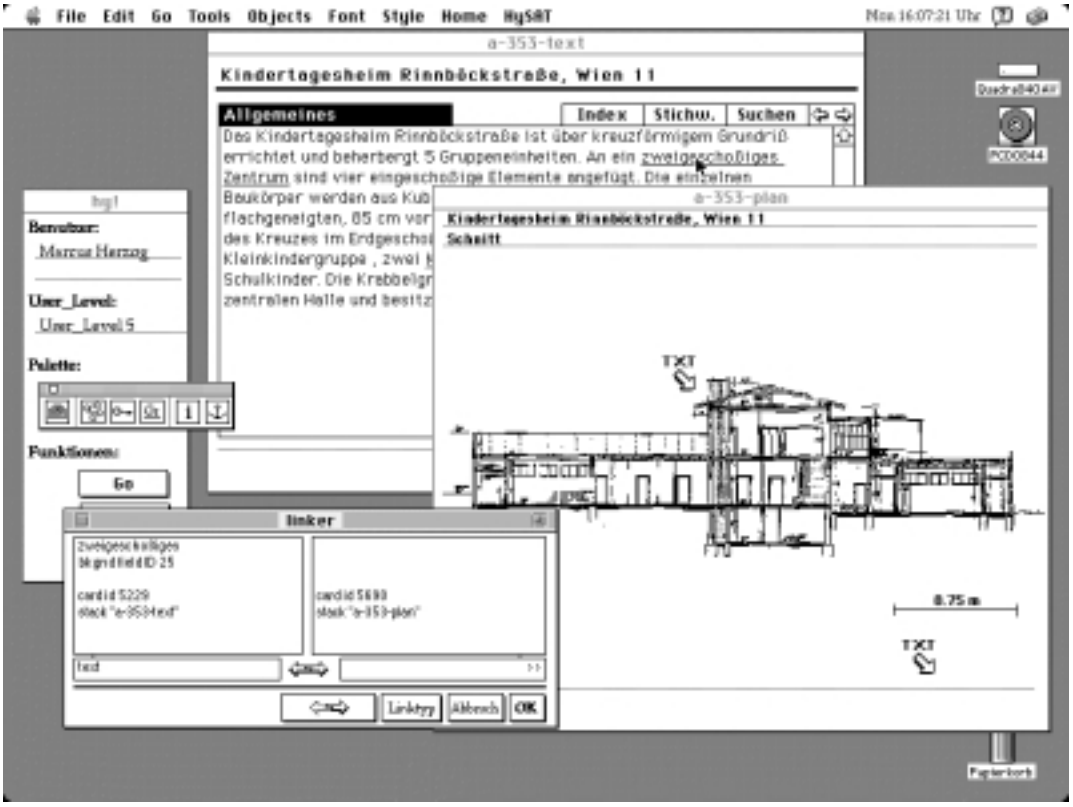

Figure 4.3: Using the link tool to generate links

ming trick we divided the original stacks into two stacks. One stack, the script stack, contains all scripts. The other stack defines the user-interface and is used as template for the actual stacks. The latter contains no scripts. Instead, the messages that are generated are passed along the inheritance path to the script stack.

Whenever one wants to change the functionality of all stacks, one just has to change the centralized script. In addition, all stacks do not carry the same redundant scripts, but use the shared script pool. On the other hand, this programming trick clearly violated the original programming intention of distributing programs to the objects that rely on these programs. This made the whole programming rather bulky and hard to maintain. The whole approach showed that while HyperCard is a nice prototyping environment, it is rather cumbersome to use it for large scale projects, that use a lot of copies of the same classes of stacks. It is nearly impossible to keep these stacks up-to-date and consistent over a longer period.

Beside this problem of maintaining the system, HyperCard lacked a number of features that we found essential for a successful system. HyperCard does not provide any kind of collaboration support. It does not even allow the concurrent use

of stacks, as long as these stacks are not locked (which means that they cannot be altered). This fact does not correspond to our vision of a knowledge universe that can be shared by all members of a design class. During the work with students this limitation proved to be very impeding for a smooth collaboration.

To summarize, HySAT did prove the usefulness of hypermedia representations of architectural design cases. We could show that this kind of knowledge representation is very liable and easy to understand for students. It is even possible for students to add their "pieces" without a large cognitive overhead. On the other hand, HySAT fell short in a number of technical details. It is the limitation of the underpinning technology that prevented the HySAT prototype from being the huge success we were hoping for. As a consequence we kept our main design considerations and changed the underlying technology.

4.4 HySAT-W3

Our second prototype investigated the possibility to use the W3 technology (see Section 3.2.2) for our purpose. We thought that W3 will resolve the problem of distribution of documents that we experienced throughout our first implementation. Based on W3, documents can reside on one server and can be read by a number of people at the same time. In W3, the basic information unit is a HTML document. We decided to emulate the appearance of our HyperCard documents by generating HTML documents that included the same basic features as our HyperCard documents. Figure 4.4 and 4.5 show two screenshots from the HySAT-W3 environment.

While it is possible to produce "similar looking" documents in W3 as in HyperCard, it is not possible to provide users with a similar environment as in HyperCard. This is due to the fact that at the current time in W3 a lot of powerful tools are missing that are available in HyperCard. To use the HyperCard environment, the user had just to fill in some information into predefined, structured "information containers". To the contrary in W3 users have to be capable of using the HTML language to write their documents. If we want to maintain a certain consistent document format, users have to be very disciplined to follow certain guidelines. There is no automatic document check for consistency.

Moreover, links in W3 are unidirectional. The whole linking schema of the HyperCard prototype was lost in the W3 application. The reason for this is that at the time no editors are in the public domain that support the attribute item of links which is defined in the HTML specification. This problem could be circumvented by writing one's own editor, but this was not possible in the scope of this project. Still the limitation of unidirectional links would remain as this is a consequence of how link management is defined in W3. Furthermore, the practice of using physical locations instead of logical names for documents and link specifiers makes the maintenance of growing document basis very complicated. Whenever a document

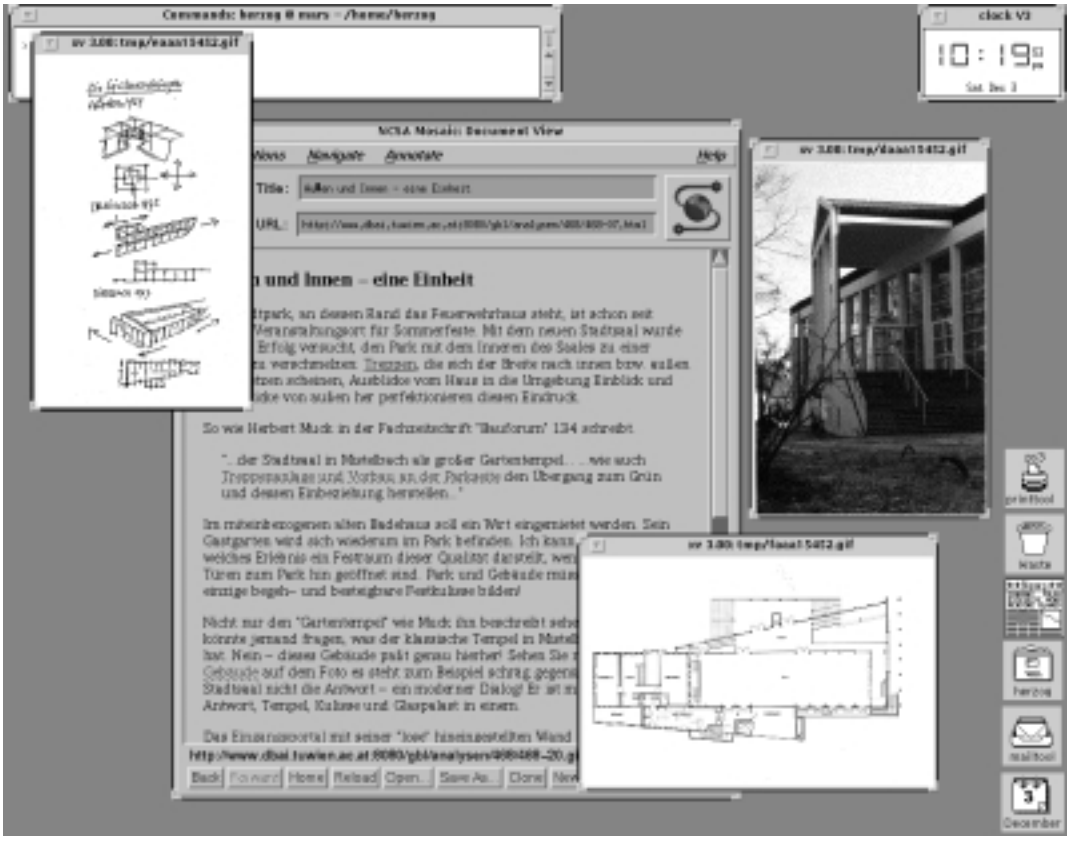

Figure 4.4: Screenshot of the HySAT-W3 environment on X Windows

is moved from a physical location to another, it is lost for all other documents that refer to it. By assigning a unique name to a document and installing a name service that will map names to locations, this problem might disappear. On the other hand, problems that might occur in such a system can be estimated from problems that occur at the time with the naming schema for Internet hosts, but on a much larger scale.

While the change from HyperCard to W3 gave us the possibility to access documents from remote locations and extend the hypermedia system across the boundaries of one machine, we lost a lot of features especially from the "user-friendliness" point of view. Encoding documents in HTML needs a much higher skill in computer usage than handling of HyperCard stacks. Writing HTML comes close to programming and is not suitable for computer illiterates; a term appropriate for at least some people of the intended user group. We found out that while it is possible to produce consistent documents it is not supported by the system. Consistent documents are the result of a very disciplinary approach towards hand-crafting these documents.

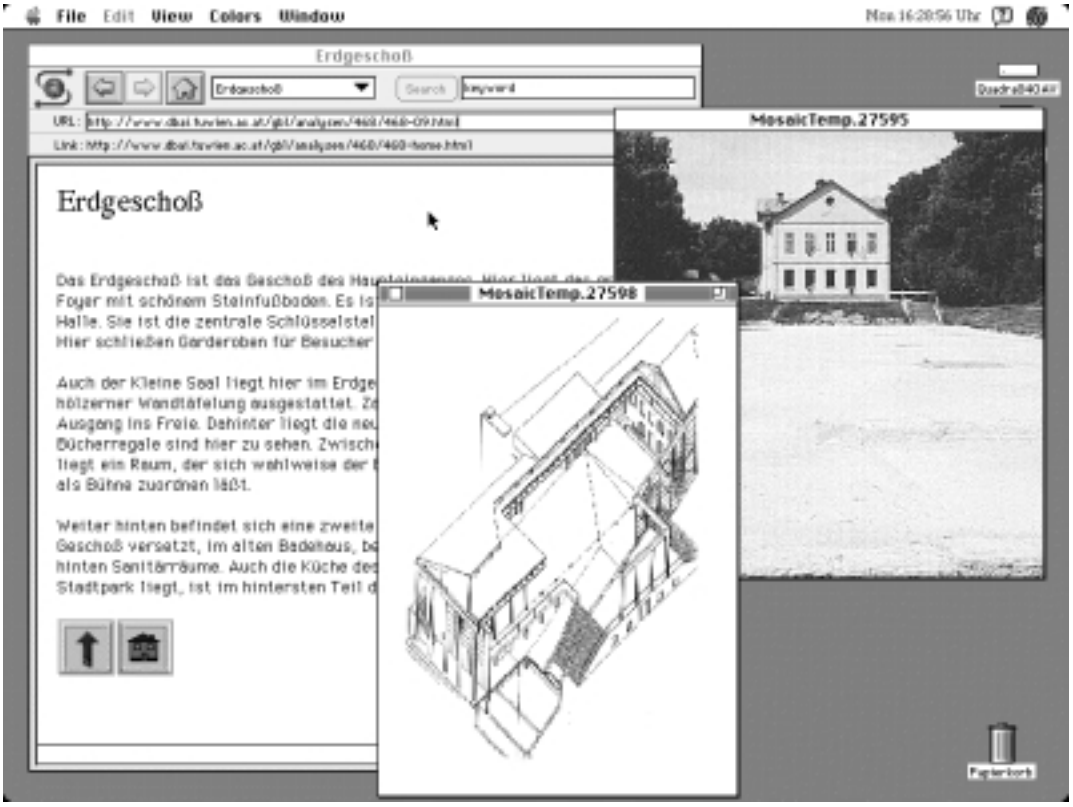

Figure 4.5: Screenshot of the HySAT-W3 environment on Apple Macintosh

A premise that cannot hold for the average user.

4.5 Future Plans

With the help of our two prototypes we could evaluate some of the key points of our conceptual framework. While the HyperCard approach was very interesting from the user interface point of view, the W3 application allowed us to investigate hypertext-in-the-large aspects of our work. Still we have not seen the unification of these features found in the two systems; neither had we the chance to investigate other key aspects such as collaboration or in-depth knowledge acquisition. We can report a high interdependency between the functionality of the system and the underpinning technology. As long as we do not base the system on technologies that support state of the art in hypermedia we will not be able to overcome these restrictions.

The choice of the hypermedia engine is an even more important task as long as there is no common interchange format for the content of hypertext systems

that will allow the transfer of hypermedia documents between different systems. We experienced that problem when we realized that there is just a very limited feasibility for converting HyperCard stacks to W3 documents. A stable document model is very important for the long term maintainability of the case base. The importance of this topic is well justified by the example of the Intermedia project [105]. When Intermedia was no longer supported because of some change in the A/UX operating system, all Intermedia course documents were more or less doomed to die, too.

We consider an approach based on the Dexter Hypermedia Reference Model for future work. The DeVise Hypermedia (DHM) system [42] provides an object-oriented framework with generic classes for all Dexter concepts. Based on these classes we can develop custom editors or integrate already existing ones to generate a Dexter-compliant hypermedia application. The use of these classes relieves us from starting from scratch again and allows us to inherit a lot of valuable concepts like the separation between storage and runtime layer, bi-directional, n-ary links, composites, and an interchange format. Furthermore, we can incorporate recent developments in the direction of CSCW and multimedia, with more to come. Drawing on the experience of the most valuable researchers in the domain of hypermedia eases the pain of reinventing the wheel and allows us to concentrate on the actual topic of our work: the design-supporting system.

What is most important is that with DHM there exists for the first time an object-oriented class hierarchy that can be used by other developers. Object-orientation is especially valuable in that context as it allows developers to re-use a certain behavior model while still being able to change the behavior were appropriate. This permits a short development time for applications based on the framework. To integrate new component types one just has to define subclasses of the appropriate generic classes that define the specialized behavior for the new component types. Common behavior can be inherited from the superclasses. This concept guarantees extensibility and tailorability.

4.6 Conclusion

The experience we gained through the use of our prototypical implementations is very valuable for further development steps. Problems that we encountered can be mostly attributed to shortcomings of the hypertext engines that we used. Again, this shows the importance of the right choice of the underpinning technology. The representation mechanism must be flexible enough to scale up with the growth of the knowledge base and has to have the potential to fulfil new requirements that emerge in the course of the development.

An urgent need that we experienced while working with our prototypes is the ability to cope with the emergent structure of the knowledge base. It is not possible

to define all potential views on the content of the knowledge base at the very beginning of the knowledge acquisition process. Instead, the actual use of the system discloses the different purposes of the knowledge base. The representation mechanism has to adapt to these different needs. We think that abstraction mechanisms within the knowledge model will facilitate a flexible presentation of knowledge. In the next chapter we will investigate intelligent methods to structure and retrieve knowledge within hypermedia systems.

Chapter 5

Intelligent Hypermedia

This chapter introduces the notion of *intelligent hypermedia*. We first give the motivation for incorporating methods that make hypermedia systems more intelligent. We then discuss two purposes of intelligent user support within hypermedia: *knowledge acquisition* and *knowledge retrieval*. We review different approaches towards information models that have been used so far. We finally examine the notion of an *index* in the context of hypermedia in more detail.

5.1 Introduction

As Halasz pointed out in his seminal paper [43], adding intelligence to hypermedia systems is one of the key research issues for the next generation of hypermedia systems. Today — seven years later — this topic is still an unresolved issue. There is no common model for hypermedia that supports computational inference, queries, and generation of hypermedia documents on the fly. Work that has been done [33, 34, 20, 62] has never come across the experimental stage of small-scale applications. We think that further work has to be done to augment the node-link model of hypertext with reasoning capabilities.

Why is there a need for hypermedia applications to be more intelligent? Without a formal representation of knowledge users are limited to browsing activities when interacting with a system. Of course, the ability to browse the information space is one of the great potentials of hypermedia systems. But what happens if the information space is just too large for effective browsing? Browsing is the appropriate method to locate interesting information when you already know where to look for it. By browsing you go from *where* to *what*. On the contrary, searching is appropriate if you know exactly what you want but do not know where to look for it. By making systems more intelligent users can be supported in their endeavor to locate appropriate information satisfying their information needs.

Hypermedia can be seen as a semi-formal knowledge representation and management technique. In the simplest case, nodes of multi-media contents are connected by one-to-one relations. The usual navigation support provided by this kind of systems is the ability to follow these links from node to node. If the system would utilize some knowledge about the contents of the nodes and the semantics of the links, it could support the user in navigating in a much more effective way. Documents could be restructured, links and nodes could be filtered, in short, the appearance of the information space could be tailored to the needs of the user. To achieve this, there must be some formal way to express information needs of users and furthermore some algorithm to map these needs onto the information space.

The literature shows two different classes of approaches to the problem of representing and processing information in hypermedia systems. On one end of the spectrum, information processing concentrates on *implicit* knowledge in documents that is utilized to derive links between related documents. The other approach is emphasizing *explicit* knowledge. Both approaches are of course not mutually exclusive and may be combined. In fact, most research work concentrates on single aspects of the overall problem. Integrating these results leads to a synergistic effect that may contribute to the solution of the problem in a new dimension. We will review these different approaches in the following sections.

5.2 Semantic Domain Models

A lot of work has been done in the field of hypertext network organization. It is a long recognized fact that hypertext structures mimic semantic networks [83]. Similar to hypertext, semantic networks are composed of nodes and links. Nodes represent concepts, links represent relationships between them. A hypertext system with arbitrary link types corresponds to a free semantic net. If the hypertext system allows just a limited number of link types, the underlying semantic net is restricted. Semantic nets are tightly coupled to the notion of associative networks. Jonassen [48] points out that psychological theories of memory and knowledge acquisition strongly support the claim that hypertext may reflect an expert's knowledge structure. Knowledge, as perceived today, is stored in the human mind in information packets which are interrelated in a network structure. This view of the organization of the human memory may be used as a model to organize the information in hypertext. The evolution of associative network theory provides the foundation for the development of hypertext structures.

The semantic network formalism can be used as general inferential representation mechanism of knowledge. In *spreading activation*, concepts surrounding a starting concept are visited in the course of problem solving. Starting from two different nodes is called an intersection search. Spreading activation fans out by following all links from the original two nodes. Then, all of these "second generation"

nodes are activated. If one node is activated from both directions, a conclusion is drawn. It is the intersection between the two spheres of activation. The resulting path indicates a potential relationship between the original two concepts. Other inferential mechanisms in semantic networks are *inheritance hierarchies* and *analogical structures* [84]. Both draw on the assumption that relevant facts about nodes can be inferred from neighboring nodes. An example for use of inheritance hierarchies in semantic networks is: If we know that Tracy is an elephant and an elephant is a mammal, we can infer that Tracy is a mammal. Analogical inheritance can be used in the following way: If we know that some concept x is the child of concept y, we can assume that concept x has an attribute x_1 that is related to the corresponding attribute y_1 of concept y.

Frame-based representation [69] adds *methods* to handle inference to the declarative knowledge representation in semantic networks. Each node in the semantic network (and in the corresponding hypertext structure) is represented as a single frame. Links are defined as slots of frames. Frame-based systems support inheritance, defaults of slot values, integration of truth maintenance, inference engines, and rule-based reasoning [13]. Through the use of *demons* (methods that are executed when triggered by some event) backward- and forward-chaining inference processes can be realized. Furthermore, inheritance is provided by a defined inheritance chain. Thus, if a slot value is not available, a value from a parent frame can be used instead. For a more detailed survey of frame-based representation techniques, see [29].

Koh, Loo, and Chua [51] describe a model of hypertext that augments the common node and link model with the structuring and deductive power of frames. They suggest to use deductive mechanisms to infer missing pieces of information from associated ones and to filter those pieces of the hypertext network that pertain to the users' information needs. In their article the authors examine many different possible interaction scenarios and outline how frames might be useful in these contexts. They propose to use frames to encode the users' information needs, frames to deduce missing information, as well as frames to monitor the user's behavior. Still, the article lacks a clear description how these frame representations are to be integrated in the node-link model and how the proposed inference algorithms work in detail. Especially an analysis how to manage the added complexity of generating and maintaining such a system is missing.

5.3 Purposes of Intelligent Hypermedia

We can identify two different main purposes of intelligent methods in hypertext. One is the support of the knowledge acquisition process during hypermedia authoring. The other is the retrieval of knowledge stored in hypertext networks. Research work done concentrated first on retrieval issues. The active support of the authoring

process is a rather new research topic.

5.3.1 Knowledge Acquisition and Structuring

In this section we concentrate on methods that support the knowledge structuring task within hypertext. This is especially interesting for the knowledge acquisition task as part of the difficult process of modeling the relevant parts of the real world for a given problem. We already discussed SEPIA in section 2.3.4, a system that supports the authoring process by providing certain semi-formal structures to help writers structure their argumentation. Another prototypical system is Aquanet by Catherine Marshall and her colleagues from Xerox PARC [64, 65, 66], which we will discuss in the following.

Aquanet is a hypertext tool that supports either individuals or groups in structuring and organizing their ideas. This includes analysis of information and building of organized structures from unstructured fragments. In contrast to the information presentation aspect emphasized in most hypertext systems through displaying nodes and their local connectivity, knowledge structuring tasks ask for a global view on the network. Grouping of nodes in the early stages of brainstorming is essential for the knowledge structuring task. That is why interaction in Aquanet revolves around a graphical network overview.

Aquanet lets people describe their knowledge domain in terms of *basic objects* (nodes) and *relations* (links). A collection of basic objects and corresponding relations between these objects is called a *schema*. An object is structured by slots that hold the knowledge about that object. All objects and relations have also a user-defined *graphic appearance* that is used to determine the appearance of these objects in the Aquanet information space. Aquanet supports the creation of multiple views of objects (called *virtual copies*) that can be placed at different locations on the display.

One of the basic goals of Aquanet was to provide users with the ability to create their own knowledge structures according to their specific tasks. Knowledge structures are controlled by schemes that define sets of allowable basic objects and relation types. The construction of schemes is supported by a type editor (for objects and relations) and a schema editor. Schemes can be changed over time. The long term evolution of schemes raises difficult issues about how to reconcile existing objects with schema changes.

When working with Aquanet, the system requires that a set of objects and relations is defined before one can proceed to build the knowledge base. This often leads to premature commitment to a schematic structure. Experience with Aquanet showed that users tend to work from concrete examples instead of following a top-down design approach for their representations. They do not identify the common

characteristics of their domain space but rather encode idiosyncratic properties. This often leads to awkward representations without proper abstractions.

Aquanet's notion of relations provides a much more expressive basis for knowledge structuring than the common concept of hypertext links. Observations during projects with Aquanet showed that this capability was much less exploited than expected. Instead, users preferred to use implicit relational structures through the use of spatial layout. Aquanet's virtual copy feature allows to organize the same objects from different points of view. Information relations like these are easily constructed for the user, but how they can be maintained in a multi-user environment is an open problem. A possible solution may be to induce schematic structure from the spatial layout.

VIKI [67], a spatial hypertext system derived from Aquanet, supports this emergent quality of structure. VIKI concentrates on active creation rather than reading and traversal of hypertext structures. It provides users with certain visual properties for denoting relationships between objects. These relationships may either remain implicit or be turned explicit by the system's spatial parser on demand, following a "structure and abstraction on demand" idea. The recognition algorithm allows users to interact with implicit structure before formally defining the structure. Marshall and her colleagues identified four classes of constraints enforced on authors by hypertext systems: *permissive*, *emergent*, *descriptive or meta-schematic*, and *prescriptive* structuring.

Permissive structure building allows authors to create idiosyncratic relationships between nodes. It does not provide means to express abstractions or assumptions underlying the building process. This class defines one end of the authoring spectrum. At the other end lies the prescriptive approach. Prescriptive structure-building imposes a certain methodology on authors that is embedded in the system. Descriptive systems provide users with certain abstractions to constrain their authoring. These abstractions are also expressed and manipulated by the authors. Finally, the emergent approach tries to overcome difficulties found in the descriptive approach: as a system's abstraction model becomes too rich, it becomes increasingly difficult to use. Emergent systems try to support gradual emergence of structure.

Hypertext is well suited for knowledge acquisition tasks [93]. Hypertext nodes can be used for a first informal description of ideas and concepts. These nodes can then be ordered and arranged according to evolving taxonomies. The experience gained throughout the Aquanet/VIKI project has shown the importance of supporting emergent structures within the hyperbase. The resulting structures in turn will be used to access the knowledge captured in hypertext networks. We will discuss the issue of knowledge retrieval in the next section.

5.3.2 Knowledge Retrieval

In this section we review the applicability of "knowledge retrieval" methods to hypertext structures. We first investigate data retrieval methods that have their roots in database theory. We then proceed to information retrieval queries that in contrast to data retrieval locate relevant rather than exactly matching information items. Methods based on information retrieval are the predominant class of retrieval algorithms applied to hypertext as found in the literature. After these system-tailored approaches we will present the concept of agenthood that focuses on interoperability between systems to retrieve knowledge.

Data Retrieval

Data or fact retrieval systems are used to find data items described by a number of formatted attributes. A query is evaluated by the exact match of the query attributes and the data item attributes. An item either matches a specific query perfectly or it fails to satisfy it. If the language of queries and data schemes is not unified at some point, a match cannot be made between queries and data items.

In hypertext, query and search mechanisms can be classified into structure search and content search. Structure search emphasizes the node and link structure of a hypermedia network. Halasz [43] gives the example of a circular structure containing a node that is indirectly linked to itself via an unbroken sequence of "support" links. This query could be used to find circular arguments. On the other hand, content search focuses on locating nodes that contain a certain information. Another example taken from Halasz, this time for content search, is: All nodes containing the string "hyper". This query will search over all nodes for the given string, independent of the network structure.

Afrati and Koutras [2] present a formal hypertext model that supports query mechanisms. They use a Prolog-like notation to describe the expressiveness of their approach. *Derived predicates* are introduced to describe predicates constructed from primitive definitions of the model. The model defines domain objects, information objects, a set of predicates, and a set of attributes. *Information objects* are instances of domain objects. Links, nodes, buttons, and regions are defined as information objects. *Scripts* are procedural attachments and are defined as properties of objects. Derived predicates such as POINT-TO-REGION(N, R, L) are true if link L points from node N to region R. Arbitrarily complex queries may be expressed using the power of predicate calculus.

GraphLog [18] is a visual query language based on G++. Structural hypertext queries are expressed by drawing graph patterns. The hyperdocument is searched for all occurrences of such patterns. GraphLog can express queries that are not expressible in conventional database languages such as relational algebra. An important

issue for a query language is its expressive power. As Consens and Mendelzon state, too little power limits the applicability of the language while too much makes efficient implementation difficult. For GraphLog, they use notions from deductive database theory and descriptive complexity to characterize the class of queries that can be formulated in the language. GraphLog turns out to be equivalent to piecewise-linear, stratified Datalog, where recursive rules are restricted to use the predicate being defined only once in its definition and negation is only allowed in a controlled way. Although GraphLog is based on the sound theory of logic programming, it avoids explicit use of logic formulae in favor of a visual user interface.

Beeri and Kornatzky [9] introduce a logical query language for hypertext systems that also permits the formulation of structural queries. The query language includes a notion of quantifiers appropriate for hypertext networks. Quantifiers are used to map quantified assertions expressed in natural language to logic expressions, e.g., for *most* paths from the current node, claim X holds. The query language consists of an extension of modal logics. The language operates on a labeled directed graph model of hypertext. Nodes and edges of the graph correspond to nodes and links of the document. Attributes and propositions are used to represent the contents of nodes and links. Link labels specify properties of links by asserting the propositions holding for the links. The language can be used to run queries to restrict the hypertext network to a certain view. Generalized quantifiers are used to tailor the network, e.g., view nodes where *many* of their direct or indirect links are attributed by a certain term.

There are just a few examples of systems in the literature that support solely data retrieval. Instead, most systems use some form of information retrieval to locate relevant items. We will review examples in the next section.

Information Retrieval

A lot of work emphasizing the usage of implicit knowledge for link discovery stems from the field of information retrieval [87, 8, 24]. Information retrieval (IR) is the process of searching for a specific information among a large number of information items. Users of IR systems expect to obtain items relevant to a certain query. The effectiveness of the system is usually judged by computing parameters such as *recall* and *precision*. Recall is defined as the retrieved proportion of relevant items whereas precision is the relevant proportion of retrieved items.

Queries are used to find information items relevant to an information need. Queries can be regarded as virtual items. The aim of the system then is to find real information items that are as similar as possible to these virtual information items. IR systems differ in how to compute this similarity. A variety of algorithms and data models have been proposed. The simplest model is based on a Boolean algebra with union, intersection, and negation operators. Retrieval is based on

matching procedures where items are either relevant or not relevant. This is due to the extreme character of the conjunction and disjunction. The problem of this model is this "all-or-nothing" response.

To overcome this problem, information items must be judged not only as relevant or irrelevant in respect to a query, but given a relative relevance. This allows for a ranking of information items. Relative relevance can either be achieved by some kind of weighting (index term weights, query term weights, etc.) or by use of fuzzy set theory to extend the Boolean model to a weighted system. In fuzzy retrieval, an item is no longer either relevant or not, but relevant to a certain degree. On the other hand, the use of a fuzzy model increases the complexity.

Another approach to overcome the problems of the Boolean model is the vector space model. In the vector space model, information items as well as queries are described as binary vectors whose components indicate the absence or presence of the nth indexing term. Query and information items are points in the vector space. Information items are ranked by the number of common descriptors in items and query. Mathematically, this is the inner product between the two vectors. Another well-known similarity measure in this model is the cosine measure, the cosine of the angle between the query and the item vector. The cosine measure ranks items according to their angle to the query vector. Vectors pointing in similar directions are thus taken to represent similar concepts. Clustering is the process of building groups of items that are close to each other according to some metric and can be represented by one most-typical item.

A third alternative how to extend the plain Boolean approach is to employ a probabilistic model. In this model, query evaluation is the computation of the probability of an item's relevance to a query. An important assumption of this model is that all descriptors of an item are conditionally independent. Furthermore, the model assumes that the probability measure of the event space is uniform. Another problem is that calculation of all probabilities of a large number of descriptors is a time-consuming task and not very practicable. The model needs relevance feedback by users and is thus highly dependent on the subjective judgment of previous users. Still, the model is based on a sound mathematical foundation.

Salton, Allan, and Buckley [88, 6] propose a system for structuring and retrieval of large classes of heterogeneous text as in newspaper files, encyclopedias, or library environments. In contrast to sophisticated conceptual representations of text (i.e., thesaurus, natural language understanding) their approach only requires a large collection of natural language text with no restrictions of subject matter and is based on statistical techniques. Links between documents or parts of documents are generated based on the similarity as defined in the vector space model. The pitfall of this approach is that coincidental use of common terms might rank two documents as highly similar even if their semantic contents differ. Also the different meanings of words can deceive the system and lead to incorrect high similarity. To minimize this

problem documents are checked for their local context. When a pair of documents has a high similarity in the overall structure, the documents are broken into sections and sentences. Then these sections are checked for similarity. If there is no sufficient similarity between these smaller units of the documents, the documents are taken not to cover the same topic.

Similarity is also used to structure the space of documents by relating documents with similar topic of discourse. After computing the global text similarity with the help of the vector model, promising texts are analyzed on a lower level, breaking documents into paragraphs and sentences. Similarities between documents and their subparts — be that sections, paragraphs, or sentences — are used to automatically generate links between these parts. The relationship between the parts of documents can be visualized in graphs where shaded portions of bars represent paragraphs of documents and edges represent significant similarity between these paragraphs. If there is not enough similarity between subparts of documents, similarity detected by the overall judging algorithm is the result of coincidence rather than because of a common topic.

The comparison of articles on the part level can reveal information about the nature of the relationship between documents. One type of relationship is the *topic-shifting* link. This kind of relationship links documents via one paragraph, changing the topic of discourse from one document to the other. *Scope-of-discourse* links relate documents that share common subtopics. This can be to various degrees of extend, balanced or unbalanced between the two documents. Some documents might broaden the topic of other documents or gather a number of subtopics under one topic heading.

While this approach is interesting for large amounts of text, it has some inherent shortcomings. Because of the use of solely statistical methods, associations between documents that are not represented through the use of a common vocabulary cannot be detected. This is a problem inherent in design subjects as the same artifacts may be discussed from completely different points of view, thus using different vocabularies. In contrast to deep knowledge acquisition, this method does not support the active creation of knowledge about a domain but rather uses surface textual analysis to create links. Finally, automatic text processing techniques are limited to this medium falling short of the multimedia character of hypermedia documents.

Autonomous Software Agents

While the methods discussed so far provide users with significant value when used in isolation, *software agents* [36] will exploit *agent communication languages* to interoperate and solve problems that cannot be solved by any single system. Agents can communicate complex information and goals. There are two different approaches towards an agent communication language standard — the procedural approach and

the declarative approach. The procedural approach emphasizes procedural directives to transmit entire programs. The declarative approach assumes that communication can best be modeled by exchange of declarative statements (definitions, assumptions, etc.).

ACL (Agent Communication Language) [72] is a declarative approach that consists of three parts: a vocabulary, a knowledge interchange format (KIF), and a knowledge query and manipulation language (KQML). The vocabulary is defined in an open-ended dictionary that contains multiple ontologies for any given application area. Each word has an English description for use by humans and formal annotations in KIF. KIF is a prefix version of first order predicate calculus. KQML provides a linguistic layer in which context is taken into account. This is taken to be necessary for efficient communication.

How can already existing software be integrated in an agent environment? There are three proposals: implementation of *transducers*, *wrappers*, or *rewriting* of the application. A transducer accepts messages from other agents and converts them into the program's native communication protocol. It then translates outgoing messages into ACL and sends them on to other agents. The wrapper approach directly injects code into existing software that guarantees the expected communication behavior. In contrast to the transducer approach, the wrapper can make use of intern data structures and may behave more efficiently. On the other hand, it requires that the source code for the program is available. The last approach, rewriting the software, is the most drastic one. But it also yields the highest efficiency beyond what would be possible in either the transduction or wrapper approach.

For the communication organization between agents two different approaches have been explored: *direct communication* and *assisted coordination*. In direct communication, agents supply other agents with their capabilities and needs. The recipients evaluate this information and reply with appropriate information. The disadvantage of this approach is the complexity of communication in large-scale settings such as the Internet, where millions of programs may communicate with each other. To avoid these communication costs a popular alternative is the *federated system* approach. In this assisted coordination, agents communicate with a special program called *facilitator*, and facilitators communicate with each other. Agents form "federations" and rely on their facilitators to fulfill their needs.

While ACL describes a declarative language for knowledge interchange, the ANSI/NISO Z39.50 Standard for Information Retrieval in the TCP/IP environment [63] defines an applications layer protocol within the OSI reference model for search and retrieval. The primary objective of the protocol definition was to encourage the development of many interoperable information services running over TCP/IP. The Z39.50 standard specifies encoding to produce a byte stream as well as connection control services. The Wide Area Information Server (WAIS) system [1] is an implementations that employs the Z39.50 standard.

An example application of agenthood in hypertext is given by Sanchez [89]. *Interface agents* are introduced to narrow the gap between the user and the complex information world. Interface agents are autonomous or semi-autonomous processes that act on behalf of the user by executing actions without direct user intervention. *HyperActive*, the resulting agent-aware hypermedia architecture, provides an *Agent Server*, an *Agent Manager*, and an *Agency Toolkit*. Any application can become an interface agent as long as it can communicate with the Agent Manager. Implementation of a common communication protocol is the only requirement. Agents may report to the user actions performed and decisions made. The architecture provides mechanisms to easily incorporate new agent classes. Rules on object attributes define conditions that trigger specified actions of agents.

5.3.3 Summary

Knowledge acquisition and knowledge retrieval are two sides of the same coin. While knowledge acquisition is concerned with defining an information structure in a domain, knowledge retrieval exploits this structure to locate relevant parts of a hypermedia network according to a given information need. We have seen that the support for an emergent structure is important in ill-defined knowledge domains such as design knowledge. The retrieval methods have to be robust enough to cope with the incompleteness and probable contradiction within the knowledge representation. In the following section we will examine an information structure that is in our understanding well suited for both, knowledge acquisition and knowledge retrieval.

5.4 Indices

Indices were originally introduced in the field of information retrieval. Everybody knows the back-of-the-book index and how to use it. It is a fruitful navigational aid when volumes of text get too large to be easily browsed. However, if the index is not designed properly it may actually reduce the accessibility of the information. Indices are approximations of concepts used in information items. The assignment of a descriptor (an index term) to an item means that the information item corresponds to the subject represented by the descriptor. Additionally, descriptors also relate to other descriptors. These relationships can be exploited when retrieving information items. We will discuss different systems using indices to retrieve hypertext nodes in the following.

5.4.1 Belief Networks

Frisse and Cousins [34] present a system that uses an index space to facilitate enhanced information retrieval, query refinement, and automated reasoning in the domain of medicine. They note that researchers in the biomedical field are confronted with millions of scientific articles from over 3000 scientific publications. In their Dynamic Medical Handbook Project they try to investigate effective methods for information retrieval from large-scale biomedical hypertexts.

Examining an information unit, activating a link label, and traveling to the new information unit is a local access method of hypertext navigation. Global navigation is facilitated by use of indices. There are several structures of index spaces. The simplest configuration is a flat document space combined with a flat index space. Documents are in a random sequence and the index space is usually in a lexicographic order. In this configuration location and meaning are not correlated. A hierarchical document space reflects the relationships between documents. Terms in the index space can also be organized as a hierarchy. Frisse and Cousins investigate how to represent index spaces as belief networks.

In their model index terms are arranged in a hierarchical manner and joined by probabilistic dependencies that encode the influences of index terms on each other. Probabilities assigned to nodes express the degree of belief that information units classified by this node will be of interest to the user. The latter are changed by user feedback via "like" or "do not like" buttons. This change in belief is then transmitted to all related information units in the index space. Calculating the degree of belief in a hierarchical index space is of polynomial order. Frisse and Cousins follow the method of Pearl [82] to compute the degree of belief in an index node. Ancestor and descendant terms in the index hierarchy influence the value of belief in an index node. At the same time, a change in the belief value of a single node through reader feedback is also propagated throughout the index space network. The intention of the algorithm is to increase belief values of nodes of interest to the reader while decreasing the belief value of others. Evaluations have shown that personal preferences are better represented by change of belief values than by absolute values of belief in index nodes.

Croft and Turtle [20] propose a similar information space configuration based on belief networks. In contrast to Frisse and Cousins they do not restrict the type of nodes of the belief network to index terms but rather support any combination of index terms and information units. Furthermore, a variety of link types are regarded as forms of evidence. Links between nodes represent "nearest neighbor" measures, structural hierarchies, and citation relationships. This even more complicated model requires compromises on the computational tractability. While conceptually very powerful, the model leaves a number of implementation issues open.

5.4.2 Fuzzy Set Theory

Lucarella introduces a plausible retrieval model based on fuzzy set theory [62]. In his model, the hypertext network is regarded as a two layer structure divided into concept network and document network. The terms of the concept network form an index to the document space. The concept network can be used for browsing as well as for the retrieval process. Searching in the hypertext network is perceived as filtering to locate interesting starting points for browsing. The intention is to reduce the hypertext network to a set of nodes that best matches the query and is manageable for browsing.

The concept network consists of a set of terms, a set of admissible relations, and a set of links. A link is a binary fuzzy relation, where the membership function indicates the strength of the semantic link between two concepts. The link relationship is defined as fuzzy transitive, where the membership function of the chain of links is the minimum of all membership functions. The strength of the chain linking nodes is given by its weakest link. The system works by spreading activation from the original query concepts throughout the network. Distance constraints stop the activation process at some specified distance from the original node. For each concept in the original query the system infers a set of semantically related concepts.

To support the retrieval process a number of inference rules are defined. The first is: If a query is about subject c and the document is about subject c then the document is relevant. By spreading across the network we can apply the following rule: Even if document d is not about subject c, it is still relevant if there exists a link between subject c and subject r; and d is about r. All different evidence values according to different subjects are aggregated and a single relevance factor for each document is computed. Documents with a greater relevance factor than a fixed threshold are selected for presentation to the user.

This approach combines effectively browsing and searching in a unified framework. The retrieval process is regarded as a process of inference. It can be carried out by the user (browsing), exploring the network, or by the system (searching), exploiting the knowledge base. During the query processing, the system makes inferences about the goals of the user, finding information that has not been explicitly requested but is likely to be useful. This emulates the reasoning of an expert. When asked about documents concerned with "expert systems", the expert will also retrieve documents about "rule-based systems", as these two topics are highly related.

Unsolved problems are the handling of the high cognitive overhead when every document has to be classified by certain index terms to a certain degree. How to justify the degree of membership to a certain concept set? Furthermore, the choice of functions to compute the aggregation of single membership function values is not unquestioned at all. As well as how to incorporate user feedback to adapt the response of the retrieval process to certain user groups? Finally, the question remains if this approach will scale up with the growth of the network?

5.4.3 Neural Networks

The guiding system proposed in the paper by Biennier, Guivarch, and Pinon [10] enriches the common hypertext structure with so-called *tag layers* that also function as index layers. In their model, the index and the hyperdocument level are associated by a bi-directional neural network that allows inductive retrieval. The links between index and content are weighted to express the description power for both directions, *direct* and *reverse*. *Reverse* represents the description power of the content for the tag definition, while *direct* measures the descriptive power of the description of a content by a tag. Both weights are usually not equal.

Queries are expressed by giving a set of tags and corresponding factors of importance. The neural network computes its activation level in response to this input signal. The selection process provides a set of semantically convenient cells together with their activation level. The cells can be ordered according to their activation level. A dynamic path can be built using these nodes. To do so, the activation level of the cells is corrected according to the hyperdocument structure. The distance between two nodes is important for the computation of the influence that two nodes have on each other. Dynamic paths are then built on the basis of semantic proximity between nodes. Nearest neighbors are defined as most active nodes.

Adapting to the user on the long term is a characteristic of neural networks. Users judge about the relevance of visited nodes according to the initial query. This either positive or negative feedback corrects the weights of the associations between nodes. That is how systems "learn" more adequate associations on the long term. Additionally, users are characterized by a profile that controls the retrieval process through parameters like a bound for the number of active tags, the number of selected documents, the specialization level for contents, and the precision level for tags.

5.5 Conclusion

We identified two main purposes for intelligent methods within the field of hypermedia: knowledge acquisition and knowledge retrieval. Methods for knowledge retrieval have a long tradition in computer science coming from different fields such as database theory, information retrieval, or knowledge-based systems. The field of knowledge acquisition is less exploited in the literature. We think that hypermedia is a promising technology to combine both, knowledge acquisition and knowledge retrieval, in a seamless environment. The support for emergent structures in the context of design is especially important. In the next chapter we will present our approach towards the formalization of design knowledge based on hypermedia technologies.

Chapter 6

Approach and Framework

We begin this chapter by reviewing the problems that we identified in the course of this study. We then give a model of architectural design knowledge that reflects our previous considerations. We elaborate in more detail on the structure and features of the resulting information model. We emphasize reasoning algorithms and how these will support the users in their various tasks. Finally, we illustrate our ideas by describing scenarios emphasizing different aspects of user interactions with the system.

6.1 Preliminary Observations

Before we present our conceptual framework for the representation of architectural design knowledge, we will summarize our findings so far and put them in a coherent presentation. Our starting point is the idea to "support the architectural design process by using the computer". We identify two different approaches towards computer-aided design: one approach emphasizes "automatic" design; the other draws on the notion of "augmenting man's intellect" by providing the human designer with an intelligent environment. We think that the latter has a high potential in the domain of architectural design.

We start our research by investigating the nature of design and identify properties that are important for supporting the process of design on the computer. We review which theoretical frameworks towards design are interesting in respect to a computer-aided support environment. Alexander's Pattern Language is interesting in terms of its holistic approach towards architectural design; unfortunately, it is not a formal language suited for direct implementation. *Exploration* is a formal representation of the design process that contrasts with the *search* paradigm of previous formal representations. We think that the notion of exploration suits the design process well. Case-based reasoning emphasizes the notion of "case" as self-contained

knowledge unit that captures all relevant knowledge for a specific design solution.

The review of already implemented design systems gives us new insight in the applicability of the various design theories. ARCHIE shows that a solely case-based reasoning approach is too rigid to capture the complexity of architectural design knowledge well. We hypothesize that hypertext may be a knowledge representation formalism well suited for design environments. The evaluation of three systems supports this assumption: JANUS uses hypertext to communicate design knowledge with the user. HENRY uses hypertext to structure design knowledge informally. SEPIA supports the design process by using semi-formal notations to structure the design knowledge.

In architecture knowledge about the synthesis of parts is deferrable from cases. We think that hypertext provides the expressiveness to represent case studies in architecture. It is open to an emergent structure in the domain knowledge and supports the acquisition of knowledge. Hypertext also supports the active transfer of knowledge. For our intended system both, knowledge acquisition as well as knowledge transfer, are essential. The first is interesting for research on how architectural knowledge can be organized while the latter is important for designers to elicit knowledge relevant to their task at hand.

Our prototypes can show partially the correctness of our assumption as given above. We can present architectural knowledge in the form of case studies. Our prototypes so far fall short in supporting active abstraction of the knowledge. We will propose a framework that integrates acquisition of knowledge, abstraction of knowledge, and retrieval of knowledge in a seamless environment. In the next section we discuss the influence of language on our proposed model. After this we present the resulting information model.

6.2 Language and Design

To build an access structure we need some kind of language that is capable to express the needs of the information seeker. Many different languages and representations are used to represent architectural design knowledge: natural language, diagrams, drawings, calculations, etc. All of these languages serve a certain purpose and are used in different stages of the design process. While natural language is mostly used to express the interpretation of design solutions, graphics and drawings are used to depict the actual design solution using a lot of conventions evolved over time.

Semantics is the study of meaning in language. The change between different semantics in different languages used during a design process is a crucial problem. The structure of the problem representation rarely corresponds to the actual structure of the problem. Christopher Alexander [4] tries to overcome this problem by introducing the idea of “constructive diagrams” which he claims are more abstract

means of representation. In his later work Christopher Alexander replaced the diagram concept by the less restrictive pattern concept. The pattern concept mixes again different languages (natural language, drawings, pictures), but integrates them in a coherent framework forming the *Pattern Language* (see Section 2.2.1). For a detailed analysis of the influence of language on the design process see also [56].

Formal languages used to access knowledge are usually more rigid. We have already discussed several different examples in Section 5.3.2. Most of these use some kind of abstraction of natural language, ranging from simple terms to whole sentences. The core problem for any kind of access structure is to find a way to abstract the knowledge without losing the "meaning" of that knowledge. The notion of "reference" as well as "sense" play an important role in that context. While reference copes with the relation between language and the external world, sense corresponds to the way things are in the world.

If we talk about simple words as "symbols" that refer to things, a popular view is that words *name* things: *words* $\longmapsto$ *things*. Proper names like *Daddy* or *Vienna* illustrate that conception. But the majority of words cannot be directly related to things. Another view of this topic denies a direct link between words and things and rather argues that links are made only through the use of our mind: *words* $\longmapsto$ *concepts* $\longmapsto$ *things*. The problem of this approach is to identify concepts for certain words and to guarantee that these concepts are shared by all people.

Meaning often depends on the specific context. Ludwig Wittgenstein refers to this saying:

> *'the meaning of a word is its use in the language'* [104]

The conclusion is that we cannot give exact definitions of concepts that we use every day. We can examine the meaning of words in certain contexts — but there is no meaning beyond that. The model of concept, as given by Dahlberg [21], is threefold and consists of: a term (symbol), a set of things (extension), and a set of attributes (intension). Most concepts cannot be defined using this model, because a complete enumeration of things referred to by a symbol is not possible and the choice of attributes is mostly arbitrary.

A model for concepts and the relationships between concepts has to reflect that inherent vague nature and still allow precise reasoning. Furthermore, incomplete and partially contradicting knowledge hardens the process of modeling the concept representation. Because the acquisition of large knowledge structures is time consuming, the complexity of the knowledge model should be as simple as possible. A high expressiveness of a model that allows a lot of different relations between concepts increases the complexity of the knowledge structure. We will reflect on this topic when we present our model in the next section.

6.3 The Information Model

In the following we present our information model that is the result of several iterative refinement steps. Some parts have already been evaluated in prototype implementations. Others are new proposals to overcome major shortcomings of our previous prototypes. We kept the idea of a large "case studies" base to model architectural design knowledge. We added abstraction mechanisms to structure, organize, and retrieve the knowledge stored in these case studies. The overall structure of the model can be described as three layered (see Figure 6.1). The main layer — the *information layer* — is the architectural design knowledge base; a collection of design cases. This draws on our experience with early prototypes as well as on theoretical considerations derived from case-based reasoning. To accomplish an easy access to the information stored in that layer, we provide the user with an abstraction level — the *concept layer* — that supports the structuring of the main level. The third level — the *user model layer* — stores user feedback and personal views of the knowledge base.

In the information layer we have decided to use the capabilities of hypermedia to encode architectural design knowledge in an informal way. Users, who add new material to the knowledge base, are requested to use certain types of information nodes (e.g., text, picture, drawing, etc.) and relations between these nodes to build a consistent knowledge base. Relations and types are used to describe the static, structural nature of the information representation. This will allow the selection of certain types of information within the total amount of information. The structures and types of information items are predefined, but can be extended by the user. In the taxonomy of Cathy Marshall's authoring constraints (see Section 5.3.1) this model will classify as prescriptive. The advantage of a prescriptive environment is that authors as well as readers can be supported based on the types of relations and nodes. Links between information items are also used to encode idiosyncratic relationships that cannot be abstracted.

While the structure of the knowledge base on the information level is made explicit through the use of node types, the contents of the information items are not as easily abstracted. The structuring method within cases is some "local" abstraction mechanism (e.g., a table of contents, a table of figures, etc.). The global structuring mechanism for the information layer is a more complex problem. Architectural design cases have no common predefined structure that can be used to index all possible design cases. Different aspects of design cases are important in respect to different points of view. Constructing an access structure to the contents of various building analyses is thus a much more tentative endeavor. The corresponding model that supports the construction of such an access structure has to reflect this character. Various versions of schemes will evolve over time, sometimes even contradicting. Still, the emergent structure must be usable for the intended purpose: guiding the information seeker. The main complication of the problem is

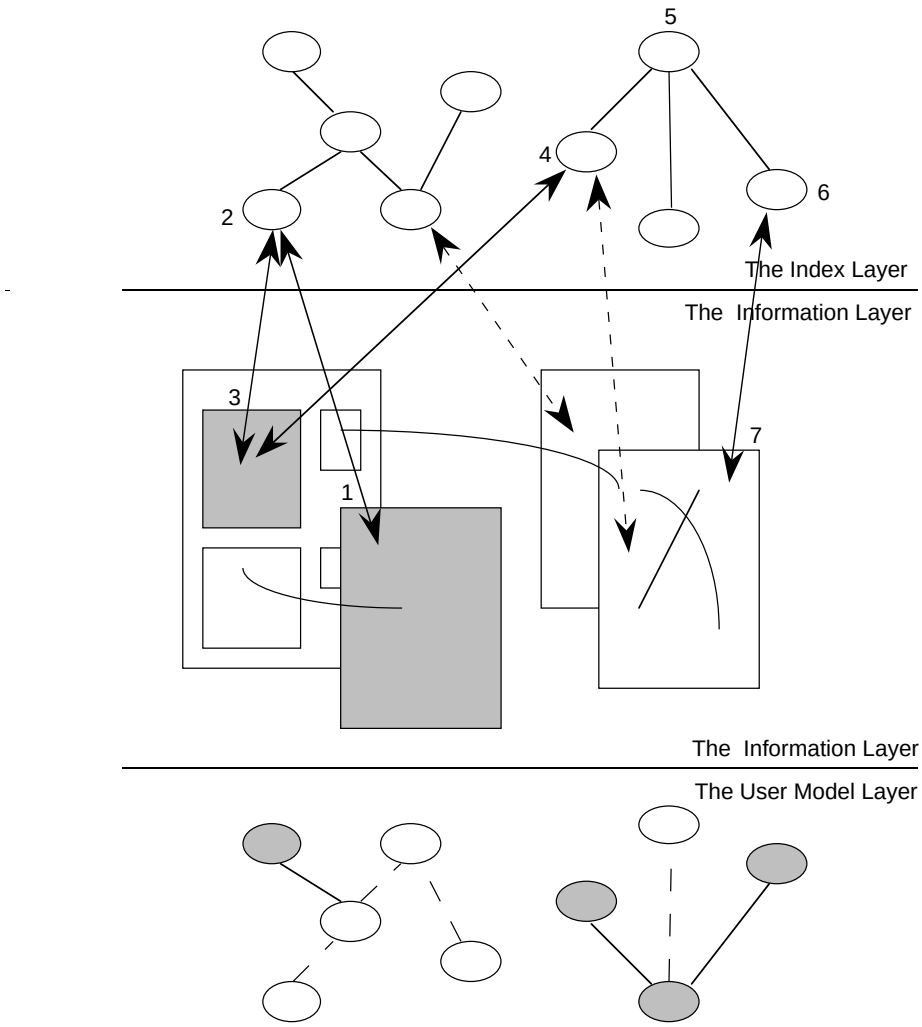

Figure 6.1: Structure of the Information Model

the evolutionary character of the information structure. There is always a possibility that new concepts, new arguments, new points of view emerge during the knowledge acquisition process. At the same time a lot of information may be already processed and integrated in the knowledge structure. The model of the structure has to be flexible enough to make a restructuring feasible without loss of all previous work.

We decided to base our access structure on the notion of an index. The index is a collection of concepts that abstract the knowledge in the information layer by associating concepts with information items. These information items might be whole building descriptions or just selected aspects. Concepts are represented in the concept layer by single terms. The relation between concepts and information items

is akin to "the information item is characterized by this concept". At the same time the meaning of a concept is characterized by the information items it refers to. This follows the understanding of Wittgenstein that concepts reveal their meaning in the context of their usage. We coined the term *language-game abstraction* (LGA) [57] for this abstraction mechanism, derived from Wittgenstein's language-games. A LGA consists of a set of concepts and a set of related information items.

The structure of the concept space evolves over time and reflects the inhomogeneous perceptions of various experts. Our model supports concurrent constructions of different collections of concepts following different intentions. The different approaches are then evaluated by user feedback applying an evolutionary conception. To help domain experts in structuring their collection of concepts that they use to abstract the information layer, our model provides four relations between items in the concept layer. The first two relations are *generalization* and *specialization* which are inverse to each other. These relations structure the concept space in a hierarchic manner, having the most general term at the root of the hierarchy tree and the most specific concepts at the leaves. The other two relations are *positive and negative association*; they express the affinity between two concepts. These relations can be weighted to express the degree to which they apply. We try to keep the complexity of the concept space rather low as otherwise the cognitive overhead would hinder the knowledge acquisition process of these abstractions. The reduction to four relations allows us to use a spatial representation of relations in the user-interface.

The user model layer finally holds information that is essential for using the model as framework for a design-support tool. The model must be capable of holding individual information for each user. Paths that one takes in the course of interaction while "exploring" a peculiar design problem are stored in the user layer. Also feedback of the user to the system is stored in this layer, e.g., the user judging the relevance of certain information units in respect to a given design problem. The content of the user layer evolves over time and represents a profile of each user. "Experiences" can be re-used when encountering a similar problem. The user layer enables the system to adapt the knowledge base according to a personal view.

In the following we summarize the key aspects of our knowledge model.

- The user model is divided into three conceptual layers: the information layer, the concept layer, and the user layer. Different types of relations are defined within and between these layers.
- The information layer models architectural case studies following the "classical" hypertext approach. Types of nodes and links formalize the structure of the information.
- The concept layer extends the hypertext model by providing abstraction mechanisms to structure and organize the knowledge within the information layer. Collections of concepts function as access structure on top of the information

layer. Different views of the information stored in the information layer are expressed using different collections of concepts.

- The user model layer represents a personal view of the knowledge base and provides mechanisms for adaptation and feedback.

In the following sections we will explain how this model supports reasoning over the semantic space spanned by the concept layer. We will compute some measurement of semantic similarity between concepts. We will give some algorithms to retrieve information chunks that are relevant to a given problem.

6.4 Reasoning Strategies

We have pointed out in the conclusion of Chapter 5 that support for knowledge retrieval is one of the main purposes of intelligent hypermedia. In the following we present strategies to apply knowledge retrieval within our model. We do this by exploiting the relations defined in our information model.

The types of nodes defined in the information layer are used to provide a filter mechanism. Users can specify what medium of information (e.g., text, picture, drawing, etc.) is interesting to them. Filters can also be applied to several other basic characteristics such as date of creation, owner, etc. The filter mechanism can be extended to cover the contents of text nodes (i.e., full-text search). It does not need any kind of semantic knowledge but rather works on a pattern-matching basis — either the item is retrieved or not.

Besides these boolean type queries we want to support vague information retrieval. This is the case when information seekers cannot exactly state what they are looking for. We can support this by applying explicit knowledge that is stored in the concept layer of our model. This layer holds knowledge about the relationship between concepts within the concept layer as well as knowledge about the information layer through relations between concepts and information items. Within one collection of concepts we use the previous defined relation types, generalization, specialization, positive association, and negative association, to compute a similarity measure.

A semantic distance measure between two concepts that are not directly related to each other can be computed by exploiting the transitive character of relationships. The inference algorithm starts at a single term and calculates for all others the degree of similarity. The weights between concepts as well as the number of intermediate concepts influence the result of the similarity measurement. Techniques that can be applied to mathematically model these algorithms can be adapted from Section 5.4. The key point for our investigation is that we can compute similarity measures within one hierarchy of concepts under vague and incomplete settings.

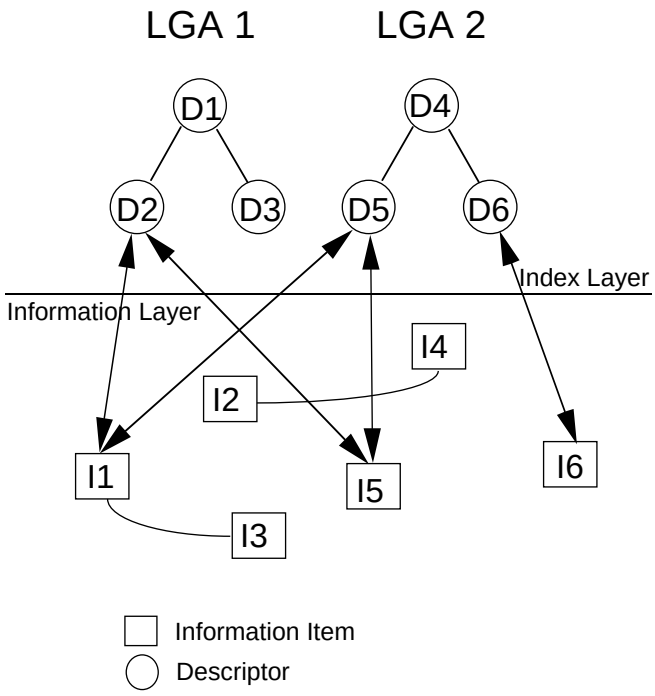

Figure 6.2: Computation of a similarity measure

Because concepts abstract the contents of information items, we can defer from the similarity measure between concepts the similarity measure between associated information items. Related information items can be retrieved by first locating concepts that are associated to the original information item. Starting from these concepts, related concepts are computed. Then the information items associated to these computed concepts are retrieved. The retrieved information items can be ordered according to their degree of similarity to the original information item. In Figure 6.2, the descriptors $D5$ and $D6$ are related to the descriptor $D4$ (via generalization). If we are interested in some "similar" case to $I6$ (referred by $D6$) we can use this relationship to infer cases $I1$ and $I5$ (referred by $D5$).

Our model supports multiple views of the information stored in the information layer via concurrent concept hierarchies. One of these views may be "construction", another "geometry". Each of these will have its own set of concepts, evolved over time in the course of abstraction and knowledge acquisition. Often it is interesting to defer related concepts across the boundaries of these views, e.g., what concept of "construction" relates to a certain concept of "geometry". Because these two sets do not share any concept, it is not possible to compute a similarity measure as we have described above. Instead, we have to retrieve all information items that are related to the concept in question. We then can retrieve all those concepts of the

other view that also refer to one or more of the retrieved information items. If one of those concepts predominates, we can infer a relationship to the original concept. In Figure 6.2, information item $I5$ is referenced by descriptor $D5$. If we want to find a related concept to $D5$ in LGA 1, all information items with descriptor $D5$ have to be retrieved and searched for a link to a descriptor of LGA 1. If one descriptor within this resulting set of LGA 1 descriptors predominates, we can infer a relation between these two descriptors. If we apply this strategy to descriptor $D5$, we will find descriptor $D2$.

In the next section we will show how we can apply these reasoning strategies to different tasks. We will do this by sketching several interaction scenarios and defining types of tasks that users will perform in the environment.

6.5 User Scenarios

In the following we give some examples of hypothesized user interaction scenarios. We highlight the different roles that users will play according to the task they want to perform. Common to all different roles is the use of the system as shared knowledge pool. We can categorize two major user groups: *knowledge providers* and *knowledge consumers*. Of course this partition is idealistic. In reality users will switch between these two roles or even play both at the same time. Within the role of knowledge providers we can subdivide into the class of users who build the information layer and those who construct the concept layer. Again, intersections between these two groups of users are possible.

Let us begin by describing the task of extending the information layer. The typical representative of this class is a student working on her building analysis. The task of modeling a building description using hypermedia is quite similar to writing a conventional document. The principal difference is that the gathered material is not presented in a solely linear fashion. The material is structured in self-contained information chunks which can be arranged in different sequences. There are certain types of nodes for all kinds of media (text, graphics, pictures, drawings, video, 3D models, etc.) which supply authors with corresponding authoring functionalities. Attributes like level of granularity, scale, etc., are idiosyncratic to the different types and are used to filter relevant information items according to a given profile.

The second main task is to organize the emergent collection of case studies. This is the responsibility of the domain expert or knowledge engineer. Our model supports several emergent indices that are used by different experts to structure the information space according to their domain expertise. These may range from *construction* via *form* to *function*. The aspect that architectural design integrates a lot of different professions and that an architect has to switch constantly between these different points of views finds it impact in this kind of parallel evolution of domain knowledge. The indices are built by introducing concepts and relating them

to corresponding information items. At the same time, the concept space is structured by using the four relations we introduced in the concept space. The index can be organized by moving terms on a plane in the graphical user-interface. The spatial layout is interpreted and translated into corresponding relations (e.g., distance between two concepts corresponds to their degree of association, etc.).

After discussing the roles of knowledge providers, we will now come to the role of knowledge consumers. Their course of action is usually tied to a certain design problem. This can be as vaguely defined as, e.g., "the usage of the material glass in building construction" or very precisely as "a detailed description of a security glass door for fire protection, scale 1:10". There are two different modes of interaction. One is undirected browsing in the knowledge base. The information seeker starts at some node in the knowledge base and travels from node to node using the usual hypertext navigation facilities. The other one exploits the use of indices to locate information relevant to the task at hand. The latter will use reasoning strategies as described in the previous section. While traveling through the knowledge space, the system will keep a record of the track in the user model layer of the information model. Feedback provided by the user about the relevance of certain nodes is stored to tailor the environment to the user's needs.

We will now describe the course of interaction for a hypothesized design example. We use the notion of *exploration* for both, the design process and the interaction process with the system. While the user explores the knowledge space looking for information relevant to her design problem, she also collects new requirements associated to her problem and gains new insight into the actual structure of her problem space. Figure 6.3 gives an example. The drawings in the figure abstract information items while the term hierarchies represent concepts in the concept layer. We use two architectural textbooks, *Precedents in Architecture* [16] and *Logic of Form* [101] to extract two possible LGAs. The numbers (1) to (11) order the exploration steps chronologically.

Using our information structure in a design-supporting system, the designer may start with the question: How to design a wide-spanned roof sheltering a gymnasium on a rectangular site? After retrieving some design cases described by the concept *Roof* of the LGA *Construction* (1-3), she gets interested in an example of a dome-shaped roof (4). Because she wants to know the consequences of a dome-shaped roof for the ground plan, she seeks for a corresponding example in the LGA *Formative Idea*. The descriptor in the LGA *Construction* for the selected roof is *Dome* (5). The system infers a relationship to the concept *Concentric* of the LGA *Formative Idea* by using exploration across the boundaries of a LGA (6). More examples retrieved for *Concentric* prove to be inconsistent with the rectangular site (7). To find similar solutions she may explore within the LGA finding *Double Center* (8) as related term concerning *Enclosure*. Examples (9) prove to be suitable for a rectangular site, e.g., two domes. By even more relaxing the constraints and looking for similar *Configuration Patterns*, she would find the concept *Binuclear* (10) and an example,

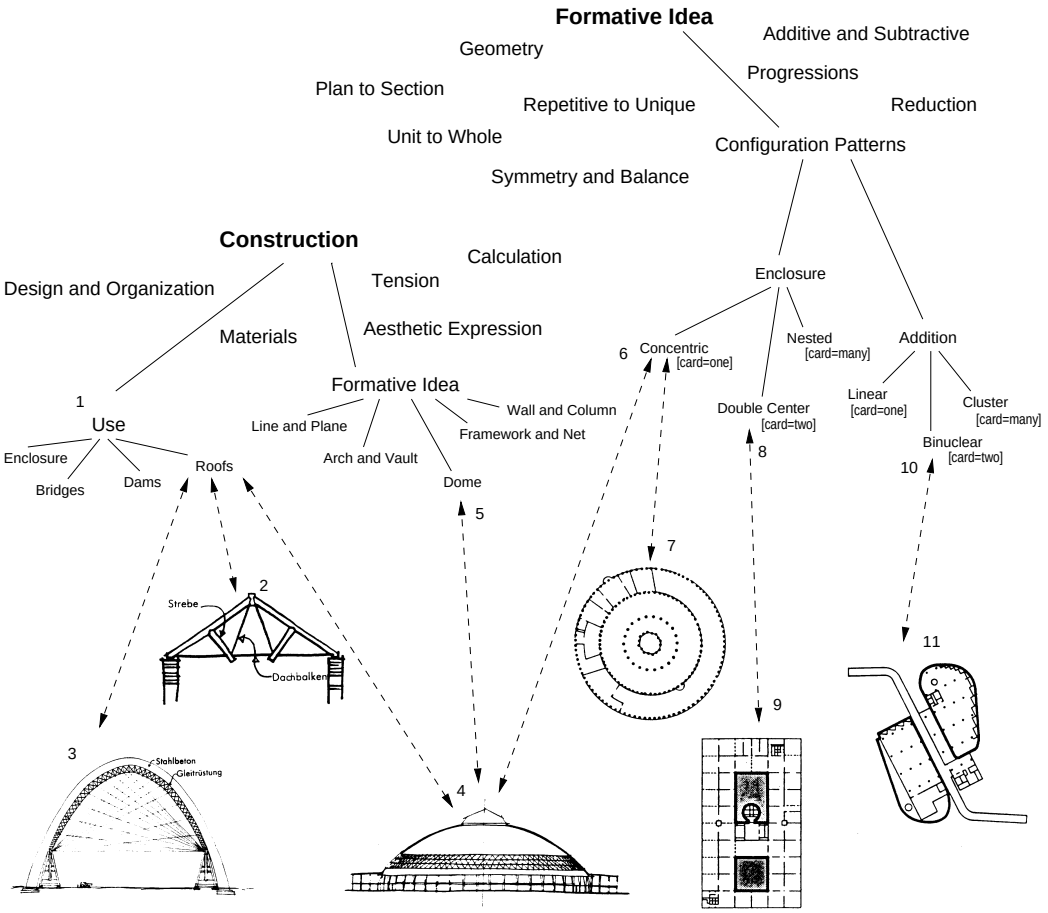

Figure 6.3: Design as exploration process

showing the addition of two related elements (11). The results of the exploration process are a list of examples, associated with a set of requirements. Thus, starting from an initial set of requirements, the designer gained new experiences during the exploration process and ended up with a refined set of requirements which will be closer to the description of the final design.

6.6 Conclusion

In this chapter we showed the use of intelligent hypermedia for modeling architectural design knowledge. We think that the proposed model is flexible enough to scale according to new requirements emerging in the course of changing purposes and intentions. This is important as much time and effort will be spent to formalize even

small parts of the whole "design knowledge space". From this point of view it seems to be permissible to compare our project to the CYC project of Douglas Lenat [61] that tries to formalize "common sense" knowledge. We think that our model supports an emergent knowledge space that is usable for knowledge structuring as well as knowledge retrieval under incomplete conditions.

Chapter 7

Epilogue

In this chapter we review our findings and point out interesting topics for future research

7.1 Summary of the Research

Support of architectural design through the use computers is an ongoing topic in various research projects. Most of these projects follow the paradigm of "automating" the design process. Our research work concentrates on the notion of "augmenting man's intellect" to increase the capability of human designers. In contrast to the "automatic design" approach, our intention is to support human designers by providing a tool that helps them to explore the problem space to gain insight into the actual problem setting.

Chapter 2 of this thesis reviewed different theories on design and their influence on actual implemented design-support systems. The notion of *exploring* rather than *searching* the space of potential design solutions seems to us a promising metaphor to characterize the design process. Alexander's *Pattern Language* is a set of patterns used by a process to generate artifacts. The process is carried out by the designer. The patterns hold all the knowledge necessary to compose design solutions with a certain "quality". The designer explores the design space — the pattern language — to learn about potential problems and corresponding solutions. We argued that hypermedia seems a promising technology to formalize this mechanism.

In Chapter 3 we discussed the technology of hypermedia. We reviewed taxonomies and conceptions of hypertext and investigated two rather contrasting models of hypertext — the Dexter Hypertext Reference Model and the World-Wide Web Model. We also investigated how hypermedia applications can be used to implement interactive learning environments. This aspect is interesting regarding our problem

setting as we intend to us the resulting system in the course of design studios, providing students of architecture with a tool that will support them in *learning by doing*.

Chapter 4 presented two prototypical implementations: HySAT and HySAT-W3. Both follow the same conceptual considerations but use different underpinning technologies. While HySAT uses HyperCard, HySAT-W3 is based on the W3 technology. The evaluations of both prototypes show the correctness of our conceptual framework but point out some shortcomings in the technologies used to implement these systems. We also observed a need to support the structuring of the emergent knowledge base. The use of techniques that scale with the growth of the aggregated information proved to be essential for the whole project.

Based on experiences gained from the prototypical implementations we saw the need to study the notion of *intelligent hypermedia* in Chapter 5. Intelligent hypermedia combines methods from artificial intelligence with ideas stemming from hypermedia research. We identified two key purposes for intelligent methods within hypermedia: the support of *knowledge acquisition* and *knowledge retrieval*. We discussed several knowledge models and proposals to incorporate "classical" artificial intelligence methods into hypermedia systems. We also proposed to use an *index* to solve both problems, knowledge acquisition and knowledge retrieval within hypermedia.

Chapter 6 finally brought together all previous considerations and presented a model to represent architectural design knowledge. Based on that model, we described reasoning strategies to retrieve relevant knowledge according to a given design problem and illustrated the usability of our model by giving some user interaction scenarios.

7.2 Future Work

HySAT and HySAT-W3 have been developed as testbed for use with students of architecture in the course of design studios. Future work will concentrate on implementing a more reliable platform based on hypermedia technologies that use a stable hypermedia document format. The maintainability of the emergent collection of documents is crucial for the success of the project. Furthermore, algorithms used to structure and to retrieve knowledge within the knowledge base have to be investigated in more detail. Especially the computational complexity as well as the cognitive complexity have to be taken into account in that investigation.

We think that the conceptual framework for using intelligent hypermedia in architectural design environments as presented in this thesis provides a useful focus in developing environments as envisioned by Vannevar Bush and Douglas Engelbart. We hope that one day we will reach our long-term goal of actually building such an environment.

Bibliography

NOTE: Order is by name of first author and year. Annotations are given where available.

[1] T. Addyman. WAIS: Strengths, weaknesses and opportunities. In *Proceedings of Information Networking 93*, London, GB, May 1993. Meckler.

[2] Foto Afrati and Constantinos D. Koutras. A hypertext model supporting query mechanisms. In N. Streitz, A. Rizk, and J. Andre, editors, *Proceedings of the ECHT'90*, pages 52–66. INRIA, Cambridge University Press, November 1990.

ANNOTATION: This model uses the power of predicate calculus to describe queries. Links, nodes, buttons, and regions are defined as properties of information objects. Both, structure and content queries are possible. See also [18, 9] for other approaches towards a query language in hypertext

[3] Robert M. Akscyn, Donald L. McCracken, and Elise A. Yoder. KMS: A distributed hypermedia system for managing knowledge in organizations. In *Proceedings of the ACM Hypertext'87*. ACM, 1987.

[4] Christopher Alexander. *Notes on the Synthesis of Form*. Harvard University Press, Cambridge, MA, 1964.

[5] Christopher Alexander. *A Pattern Language*. Harvard University Press, Cambridge, MA, 1977.

[6] James Allan and Gerard Salton. The identification of text relations using automatic hypertext linking. In James Mayfield and Charles Nicholas, editors, *Proceeding of the Workshop on Intelligent Hypertext, in conjunction with CIKM'93*, November 1993.

ANNOTATION: See [88] for an extended version of this paper

[7] John Archea. Puzzle-making: What architects do when no one is looking. In Y. E. Kalay, editor, *Computability of Design*, pages 37–51. John Wiley & Sons, New York, NY, 1987.

[8] Martin Bärtschi. An overview of information retrieval subjects. *IEEE Computer*, 18(5):67–84, May 1985.

ANNOTATION: Good beginner's introduction to the field of information retrieval. It covers the basic notions of informtaion retrieval

and gives an in-depth analysis of several models such as vector space models, fuzzy set models, and probabilistic models. See also [87, 24] for introductory books on information retrieval

[9] Catriel Beeri and Yoram Kornatzky. A logical query language for hypertext systems. In N. Streitz, A. Rizk, and J. Andre, editors, *Proceedings of the ECHT'90*, pages 67–80. INRIA, Cambridge University Press, November 1990.

ANNOTATION: This query language introduces the notion of quantifiers and permits the formulation of structural queries, e.g., for some paths from node X that is attributed with a certain term, claim Y holds. The logical query language consists of an extension of modal logics. See also [18, 2] for other examples of query languages in hypertext.

[10] Frederique Biennier, Michel Guivarch, and Jean-Marie Pinon. Browsing in hyperdocuments with the assistance of a neural network. In N. Streitz, A. Rizk, and J. Andre, editors, *Proceedings of the ECHT'90*, pages 288–297. INRIA, Cambridge University Press, November 1990.

[11] Jay David Bolder. *Writing space: the computer, hypertext, and the history of writing*. Lawrence Erlbaum Associates, Inc., Hillsdale, NJ, 1991.

ANNOTATION: This book covers topics such as the computer as new writing space, writing as technology, the electronic book, etc. It elaborates on the new quality of text gained through the use of hypermedia. A hypertext version is also available. See [59] for another book on this topic

[12] Vannevar Bush. As we may think. *Atlantic Monthly*, 176(1):101–108, 1945.

ANNOTATION: Seminal paper in the field of hypertext. Written in 1945 it covered for the first time the notion of a seamless information space to organize ones own thoughts and ideas. Reprinted in [41].

[13] David A. Carlson and Sudha Ram. Hyperintelligence: The next frontier. *Communications of the ACM*, 33(3):311–321, March 1990.

[14] Erran Carmel, Stephen Crawford, and Hsinchun Chen. Browsing in hypertext: A cognitive study. *IEEE Transactions on Systems, Man, and Cybernetics*, 22(5):865–883, 1992.

[15] Gianfranco Carrara, Yehuda Kalay, and Gabriele Novembri. Intelligent systems for supporting architectural design. In Gerhard N. Schmitt, editor, *Proceedings of the CAAD futures'91*, pages 175–186. ETH Zürich, July 1991.

[16] H.C. Clark and M. Pause. *Precedents in Architecture*. Van Nostrand Reinhold, New York, NY, 1985.

[17] Jeff Conklin. A survey of hypertext. Technical report, MCC, Austin, Texas, 1987.

ANNOTATION: Interesting paper that reviews many existing hypertext systems and introduces a taxonomy to categorize these systems.

It also explores fundamental features of hypertext and design options of underpinning technologies.

[18] Mariano P. Consens and Alberto O. Mendelzon. Expressing structural hypertext queries in GraphLog. In *Proceedings of the ACM Hypertext'89*, pages 269–292. ACM, November 1989.

ANNOTATION: GraphLog is a visual query language. Structural hypertext queries are expressed by drawing graph patterns. The hyperdocument is searched for all occurrences of such patterns. Other examples of query languages can be found in [2, 9].

[19] R.D.Coyne, M.A. Rosenman, A.D. Radford, M. Balachandran, and J. S. Gero. *Knowledge-based Design Systems.* Addison-Wesley, Reading, MA, 1990.

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

[21] Ingetraud Dahlberg. Die gegenstandsbezogene, analytische Begriffstheorie und Ihre Definitionsarten. In B. Ganter, R. Wille, and K.E. Wolff, editors, *Beiträge zur Begriffsanalyse*, pages 9–22. BI Wissenschaftsverlag, 1987.

[22] Eric A. Domeshek and Janet L. Kolodner. Towrds a case-based aid for conceptual design. *International Journal of Expert Systems*, 4(2):201–220, 1992.

[23] Thomas M. Duffy and Rany A. Knuth. Hypermedia and instruction: Where is the match? In David H. Jonassen and Heinz Mandl, editors, *Designing Hypermedia for Learning*, volume 67 of *Nato ASI Series F*, pages 199–225. Springer-Verlag, New York, NY, 1990.

ANNOTATION: The authors describe a project to design, develop, and implement an Enriched Learning Environment (ELE). They discuss the role of hypermedia in the context of learning activities. They state that if one understands the effective use of hypermedia as a database to explore, one must give more attention to the assignments that accompany that explorations. Although the authors question the benefit of hypermedia in regard to the development of new instructional goals and nonlinear thinking, they believe that hypermedia can make more information available to the learner. Learning must be an active process. The learner must come to see it relevant to his context. The focus of ELE is on facilitating the transfer of what is learned in the classroom to its application in everyday life. See [49] for related work.

[24] David Ellis. *New Horizons in Information Retrieval.* Library Association Publishing Ltd, London, GB, 1990.

ANNOTATION: This book provides an introduction to novel developments in the field of information retrieval. It covers statistical and probabilistic retrieval as well as expert intermediary systems and

hypertext systems. It is a good review of existing technologies and brings together all possible notions of information retrieval.

[25] Douglas C. Engelbart. A conceptual framework for the augmentation of man's intellect. In P. W. Howerton and D. C. Weeks, editors, *Vistas in Information Handling*, pages 1–29. Spartan Books, Washington, D.C., 1963.

ANNOTATION: First paper by Engelbart that introduced the idea of augmenting the human intellect. See also [26, 27, 28] for later updates on this topic. Reprinted in [41].

[26] Douglas C. Engelbart and William K. English. A research center for augmenting human intellect. In *Proceedings of FJCC 33(1)*, pages 355–410. AFIPS Press, 1968.

ANNOTATION: Reprinted in [41].

[27] Douglas C. Engelbart. Towards high-performance knowledge workers. In *Proceedings of the 1982 AFIPS Office Automation Conference*, pages 279–290. AFIPS Press, 1982.

ANNOTATION: Reprinted in [41].

[28] Douglas C. Engelbart. Authorship provisions in augment. In *Proceedings of the IEEE Compcon Conference*. IEEE, 1984.

ANNOTATION: Reprinted in [41].

[29] R. Fikes and T. Kehler. The role of frame-based representation in reasoning. *Communications of the ACM*, 28(9):904–920, September 1985.

[30] Gerhard Fischer and Raymond McCall. Janus: Integrating hypertext with a knowledge-based design environment. In *Proceedings of the ACM Hypertext'89*, pages 105–117. ACM, November 1989.

ANNOTATION: This paper presents the JANUS system. JANUS is a design-support system for kitchen design. It employs the method of design critics as presented in [31]. A summary of the whole critics approach can be found in [32]

[31] Gerhard Fischer and Kumiyo Nakakoji. Making design objects relevant to the task at hand. In *Proceedings of AAAI-91, Ninth National Conference on Artificial Intelligence*, pages 67–73. AAAI Press/The MIT Press, 1991.

[32] Gerhard Fischer, Kumiyo Nakakoji, Jonathan Ostwald, Gerry Stahl, and Tamara Sumner. Embedding critics in design environments. *The Knowledge Engineering Review*, 8(4):285–307, 1993.

ANNOTATION: The authors use a methodology called *critics* [31] to support designers specifying a problem and constructing a solution. Critics follows the human critiquing paradigm where interjections of reasoned opinions trigger reflections on the artifact being designed. This article focuses on applying critics on domain-oriented design environments. HYDRA-KITCHEN, a residential kitchen design environment built by the authors, is used as an example to demon-

strate the capabilities of the critics approach. Three embedded critiquing mechanisms are presented: generic, specific, and interpretive critics. Generic critics reflect knowledge that applies to all designs. Specific critics are related to situation-specific requirements. Interpretive critics provide alternative views to given design solutions.

[33] Mark E. Frisse. Searching for information in a hypertext medical handbook. *Communications of the ACM*, 31(7):880–895, July 1988.

ANNOTATION: This paper discusses the possibilities for a query language in hypertext in medical settings. A key point is how to find information in a dynamic medical textbook. Structural and contextual relationships are exploited to specify retrieval algorithms. Later work on that project can be found in [34]

[34] Mark E. Frisse and Steve B. Cousins. Information retrieval from hypertext: Update on the dynamic medical handbook project. In *Proceedings of the ACM Hypertext'89*, pages 119–212. ACM, November 1989.

ANNOTATION: Building on the work presented in [33], this paper attempts to provide a more complete theory on information retrieval from hypertext documents. Two general classes of navigation methods are compared: local and global. The authors argue that a global navigation aid necessitates some form of index space. Various architectures of index and document spaces are presented and associated computational complexities discussed.

[35] Richard P. Gabriel. Pattern languages. *Journal of Object-Oriented Programming*, 6(8):72–75, January 1994.

[36] Michael R. Genesereth and Steven P. Ketchpel. Software agents. *Communications of the ACM*, 37(7):48–53, July 1994.

ANNOTATION: There are more articles on this topic in the same issue of the Communications of the ACM

[37] John S. Gero and Mary Lou Maher. Mutation and analogy to support creativity in computer-aided design. In Gerhard N. Schmitt, editor, *Proceedings of the CAAD futures'91*, pages 241–249. ETH Zürich, July 1991.

ANNOTATION: This paper describes the computational model for a creative process in design. Mutation and analogy algorithms are used to introduce new design variables in certain design situations. Dependency networks are used to implement prototypical mutation operators. Design prototypes are used to represent function, behavior, and structure of design examples.

[38] A. Goel, J.L. Kolodner, M. Pearce, R. Billington, and C. Zimring. ARCHIE: A case-based architectural design system. Technical Report GIT-CC-91/18, College of Computing. Georgia Institute of Technology, Atlanta, GA, 1991.

[39] C. F. Goldfarb. *The SGML Handbook*. Oxford University Press, 1991.

[40] D. Goodman. *The Complete HyperCard Handbook*. Bantam Books, New York, NY, 1987.

[41] Irene Greif. *Computer-Supported Cooperative Work: A Book of Readings*. Morgan Kaufmann Publishers, San Mateo, CA, 1988.

ANNOTATION: A very good collection of papers on the topic of CSCW. The collection consists of three parts. The first is on visions and first steps towards CSCW. The second discusses new technologies for CSCW. The third finally introduces CSCW design theories.

[42] Kaj Gronbaek and Randall H. Trigg. Design issues for a dexter-based hypermedia system. *Communications of the ACM*, 37(2):41–49, February 1994.

ANNOTATION: Gronbaek and Trigg turn the Dexter Hypertext Reference Model into a design specification and discusses an object-oriented implementation of important Dexter concepts. Together with other articles [47, ?, ?] this issue of the Communications of the ACM provides a good introduction to the concepts of the Dexter Hypertext Reference Model as introduced in [45].

[43] 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.

ANNOTATION: Although presented 1987, Halasz's paper is still of interest. Most issues are not resolved so far and still topics of research work.

[44] F.G. Halasz, T.P. Moran, and R.H. Trigg. Notecards in a nutshell. In *Proceedings of the 1987 ACM Conference of Human Factors in Computer Systems*, pages 45–52. ACM, April 1992.

[45] Frank Halasz and Mayer Schwartz. The dexter hypertext reference model. In *Proceedings of the Hypertext Workshop*, pages 95–133, Gaithersburg, Md, January 1990. National Institute of Standards and Technology, NIST Special Publication.

ANNOTATION: First presented at the NIST Hypertext Workshop, this model still has influence on current hypermedia developments. There are several research projects working on refinements or extensions to the original Dexter model. Recently, a collection of reports on on-going work has appeared in the Communications of the ACM, see [42].

[46] K. J. Hammond. Chef: A model of case-based reasoning. In *Proceedings of AAAI-86*, pages 267–271, 1986.

[47] Linda Hardman, Dick C.A. Bulterman, and Guido van Rossum. The amsterdam hypermedia model: Adding time and context to the dexter model. *Communications of the ACM*, 37(2):50–62, February 1994.

ANNOTATION: See [42] for related articles.

[48] David H. Jonassen. Semantic network elicitation: tools for structuring hypertext. In Ray McAleese, editor, *Hypertext: State of the Art*, pages 142–152. Intellect Limited, Oxford, UK, 1990.

[49] David H. Jonassen and R. Scott Grabinger. Problems and issues in designing hypertext/hypermedia for learning. In David H. Jonassen and Heinz Mandl, editors, *Designing Hypermedia for Learning*, volume 67 of *Nato ASI Series F*, pages 3–25. Springer-Verlag, New York, NY, 1990.

ANNOTATION: This is an interesting collection of papers on the topic of learning and hypermedia. All papers are commented by all authors, thus providing a hypertext-like appearance. Different link types support the construction of a conceptual space. Unfortunately, the limitations of the printed version prohibit a real "hypertext feeling". The following papers are part of this collection: [103, 68, 86, 23, 60]

[50] Terry W. Knight. Designing with grammars. In Gerhard N. Schmitt, editor, *Proceedings of the CAAD futures'91*, pages 19–34. ETH Zürich, July 1991.

ANNOTATION: While most shape grammars are used to analyze architectural or other styles of designs, these paper introduces shape grammars for the creation of new and original styles of designs. It reports about the experience with shape grammars in the design studio. Students can design their own vocabulary elements and geometric relationships. Sometimes this leads to difficulties for students to translate emergent forms into architectural forms that fit particular programs.

[51] Toh-Tzu Koh, Peing Ling Loo, and Tat-Seng Chua. On the design of a frame-based hypermedia system. In Ray McAleese, editor, *Hypertext: State of the Art*, pages 154–165. Intellect Limited, Oxford, UK, 1990.

[52] Janet L. Kolodner. An introduction to case-based reasoning. *Artificial Intelligence Review*, 6:3–34, 1992.

ANNOTATION: This paper gives a good overview of the technique of case-based reasoning. It uses examples to illustrate the applicability of this method to solve design problems such as planning a meal. It explains the model of case-based reasoning and discusses the case-based reasoning cycle.

[53] J. L. Kolodner, R. L. Simpson, M. Pearce, and K. Sycara. A process model of case-based reasoning in problem solving. In *Proceedings of IJCAI-85*, pages 284–290, 1985.

[54] Hinny P. Kong. An intelligent, multimedia-supported instructional system. *Expert Systems With Applications*, 7(3):451–465, March 1994.

[55] Herbert Kramel and Chen-Cheng Chen. Bau: A knowledge-based system for the investigation of a basic architectural unit. In Gerhard N. Schmitt, editor, *Proceedings of the CAAD futures'91*, pages 307–322. ETH Zürich, July 1991.

[56] Christian Kühn and Marcus Herzog. A language game approach to architectural typology. In Gernot Pittioni, editor, *Proceedings of the ECAADE 91*, October 1991.

[57] Christian Kühn and Marcus Herzog. Modeling the representation of architectural design cases. *Automation in Construction*, 2(1):1–10, March 1993.

ANNOTATION:

[58] Rainer Kuhlen. *Hypertext.* Springer-Verlag, Berlin, GER, 1991.
ANNOTATION: Introductory book to the field of hypertext in German language. Large bibliography.

[59] Georg P. Landow. *Hypertext: the convergence of contemporary critical theory and technology.* The Johns Hopkins University Press, Baltimore, Md, 1992.
ANNOTATION: Landow examines the new rhetoric of hypertext. Reconfiguring the text and reconfiguring the author are two important notions in this context. Landow conducted several classes on English literature using the Intermedia system at Brown University's IRIS lab. He is the author of *Dickens Web*. Compare this book to [11]

[60] John J. Leggett, John L. Schnase, and Charles J. Kacmar. Hypertext for learning. In David H. Jonassen and Heinz Mandl, editors, *Designing Hypermedia for Learning*, volume 67 of *Nato ASI Series F*, pages 27–37. Springer-Verlag, New York, NY, 1990.
ANNOTATION: See [49] for related work.

[61] Douglas B. Lenat, Ramanathan V. Guha, Karen Pittman, Dexter Pratt, and Mary Shepherd. Cyc: Toward programs with common sense. *Communications of the ACM*, 33(8):30–49, August 1990.

[62] Dario Lucarella. A model for hypertext-based information retrieval. In N. Streitz, A. Rizk, and J. Andre, editors, *Proceedings of the ECHT'90*, pages 81–94. INRIA, Cambridge University Press, November 1990.

[63] Clifford Lynch. Using the Z39.50 information retrieval protocol in the internet environment. Technical report, Internet Engineering Task Force INTERNET-DRAFT, 1994.

[64] Catherine C. Marshall, Frank. G. Halasz, Russell A. Rogers, and William C. Janssen Jr. Aquanet: a hypertext tool to hold your knowledge in place. In *Proceedings of the ACM Hypertext'91*, pages 261–275. ACM, November 1991.
ANNOTATION: This paper introduced the Aquanet project. Aquanet emphasizes knowledge acquisition and structuring within hypertext. Several up-dates on this project have been published (see [65, 66, 67]).

[65] Catherine C. Marshall and Russell A. Rogers. Two years before the mist: Experiences with Aquanet. In *Proceedings of the ECHT'92*, pages 53–62. ACM, November 1992.

[66] Catherine C. Marshall and Frank M. Shipman III. Searching for the missing link: Discovering implicit structure in spatial hypertext. In *Proceedings of the ACM Hypertext'93*, pages 217–230. ACM, November 1993.

[67] Catherine C. Marshall, Frank M. Shipman III, and James H. Coombs. VIKI: Spatial hypertext supporting emergent structure. In *Proceedings of the ECHT'94*, pages 13–23. ACM, September 1994.

[68] Terry Mayes, Mike Kibby, and Tony Anderson. Learning about learning from hypertext. In David H. Jonassen and Heinz Mandl, editors, *Designing Hypermedia for Learning*, volume 67 of *Nato ASI Series F*, pages 227–250. Springer-Verlag, New York, NY, 1990.

ANNOTATION: See [49] for related work.

[69] Marvin Minsky. A framework for representing knowledge. In P. Winston, editor, *The Psychology of Computer Vision*, pages 211–277. McGraw Hill, New York, NY, 1975.

[70] William J. Mitchell, Robin S. Liggett, Spiro N. Pollalos, and Milton Tan. Integrating shape grammars and design analysis. In Gerhard N. Schmitt, editor, *Proceedings of the CAAD futures'91*, pages 1–18. ETH Zürich, July 1991.

ANNOTATION: This paper draws on the notion that design can be represented using a shape grammar to generate design alternatives and analyses procedures to test them. A vocabulary for a "primitive hut" is presented. Testing procedures use standard engineering structural analysis. The application of this approach in teaching is discussed in the later part of the paper.

[71] K. Nakakoji. *Increasing shared knowledge of design tasks between humans and design environments: The role of a specification component.* PhD thesis, Department of Computer Science, University of Colorado at Boulder, 1993.

[72] R. Neches, R. Fikes, T. Finin, T. Gruber, R. Patil, T. Senator, and W. Swartout. Enabling technology for knowledge sharing. *AI Magazine*, 12(3):36–56, March 1991.

[73] Theodor Holm Nelson. *Literary Machines.* The Distributors, South Bend, IN, 87.1 edition, 1987.

ANNOTATION: This book was first published in 1981 and has undergone various revisions throughout the years. It reports about the Xanadu project, a vision of a world-wide hypertext system, where anybody can create documents and put it into Xanadu's data base. Royalties would be paid according to the time a user was reading a specific document. Whoever met Ted Nelson once will know this is a book worth reading it.

[74] Susanne Neubert and Gabriele Schmidt. Hypertext und Künstliche Intelligenz, Aktuelle Trends 1993. Technical Report 289, Institut für Angewandte Infor-

matik und formale Berschreibungsverfahren, Universität Karlsruhe, January 1994.

[75] Steven R. Newcomb, Neill A. Kipp, and Victoria T. Newcomb. The HyTime hypermedia/time-based document structuring language. *Communications of the ACM*, 34(11):67–83, November 1991.

[76] A. Newell and H. A. Simon. GPS: A program that simulates human thought. In E. Feigenbaum and J. Fieldman, editors, *Computers and Thought*. McGraw-Hill, New York, NY, 1963.

[77] Jakob Nielsen. *Hypertext and Hypermedia*. Academic Press, San Diego, CA, 1990.

ANNOTATION: This is one of the classical introductory books to the field of hypertext. It explains all basic notions of hypertext and is easy to read and understand. It contains a rich annotated bibliography which serves as starting point for further reading. A second edition will appear soon.

[78] Jakob Nielsen. The art of navigation trough hypertext. *Communications of the ACM*, 33(3):311–321, March 1990.

[79] ISO/IEC. *Information technology - Open Document Architecture (ODA) and Interchange Format - Temporal relationships and non-linear structures, Draft International Standard*. ISO/IEC DIS 8613-14:1993, 1993.

[80] Tim Oren. The architecture of static hypertexts. In *Proceedings of the ACM Hypertext'87*. ACM, 1987.

[81] Rivka E. Oxman and Robert M. Oxman. Refinement and adaptation: Two paradigms of form generation in CAAD. In Gerhard N. Schmitt, editor, *Proceedings of the CAAD futures'91*, pages 291–306. ETH Zürich, July 1991.

[82] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgen Kaufmann, San Mateo, CA, 1988.

[83] M. Ross Quillian. Semantic memory. In Marvin Minsky, editor, *Semantic Information Processing*, pages 227–270. MIT Press, Cambridge, MA, 1968.

[84] Roy Rada. *Hypertext*. McGraw-Hill, London, UK, 1991.

ANNOTATION: This book brings together topics from human-computer interaction, information storage and retrieval, CSCW, as well as artificial intelligence. The content is organized in four topics: small-volume hypertext, large-volume hypertext, collaborative hypertext, and intelligent hypertext. The book is also available in several hypertext versions.

[85] H. Rittel. On the planning crisis: Systems analysis of the 'first and second generations'. In *Bedriftsokonomen*, number 8, pages 390–396. 1972.

[86] Alexander J. Romiszowski. The hypertext/hypermedia solution — but what exactly is the problem? In David H. Jonassen and Heinz Mandl, editors, *Designing Hypermedia for Learning*, volume 67 of *Nato ASI Series F*, pages 321–354. Springer-Verlag, New York, NY, 1990.

ANNOTATION: See [49] for related work.

[87] Gerard Salton and Michael J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, New York, NY, 1983.
ANNOTATION: Standard introductory book to information retrieval. Covers important aspects of information retrieval. See [24] for novel developments in the field of information retrieval.

[88] Gerard Salton, James Allan, and Chris Buckley. Automatic structuring and retrieval of large text files. *Communications of the ACM*, 37(2):97–108, February 1994.
ANNOTATION: The authors propose a system that retrieves information from large collections of text without a sophisticated text representation. Instead, they use the vector processing model to handle information retrieval operations. Text comparison algorithms are used to automatically structure the text. Sufficiently similar texts are linked, allowing joint access to related text excerpts.

[89] José Alfredo Sánchez. HyperActive: Extending an open hypermedia architecture to support agency. Master's thesis, Texas A&M University, December 1993.

[90] Gerhard N. Schmitt, editor. *Proceedings of the CAAD futures'91*. ETH Zürich, 1991.

[91] Shen-Guan Shih. Case-based representation and adaptation in design. In Gerhard N. Schmitt, editor, *Proceedings of the CAAD futures'91*, pages 279–290. ETH Zürich, July 1991.

[92] Mario J. Silva and Randy H. Katz. Henry: a hypertext based VLSI design and documentation system. In Kasper Osterbye, editor, *Proceedings of the Hypertext'93 Workshop on Hypertext in Engineering*. Aalborg University, November 1993.

[93] L. Simon, editor. *4. Workshop "Hypertext und Künstliche Intelligenz", Einsatzgebiet Wissensakquisition*, Erlangen, München, Passau, GER, 1993. Bayerisches Forschungszentrum für Wissensbasierte Systeme FORWISS.

[94] B.F. Skinner. Teaching machines. In A.A. Lumsdain and R. Glaser, editors, *Teaching Machines and Programmed Learning*, pages 137–158. Nat'l Ed. Assoc., Washington, D.C., 1960.

[95] Tim Smithers and Wade Troxell. Design is intelligent behaviour, but what's the formalism? *AI EDAM*, 4(2):89–98, 1990.

[96] John B. Smith and Stephen F. Weiss. Hypertext. *Communications of the ACM*, 31(7):816–819, July 1988.

[97] G. Stahl. Supporting interpretation in design. *Architecture and Planning Research*, 1993.

[98] Norbert A. Streitz, Jörg Hannemann, and Manfred Thüring. From ideas and arguments to hyperdocuments: Travelling through activity spaces. In *Proceedings of the ACM Hypertext'89*, pages 343–364. ACM, November 1989.

[99] N. Streitz, J. Haake, H. Hannemann, A. Lemke, W. Schuler, H. Schütt, and M. Thüring. SEPIA: A cooperative hypermedia authoring environment. In *Proceedings of the ECHT'92*, pages 11–22. ACM, November 1992.

[100] Norbert A. Streitz, Jörg Geißler, Jörg M. Haake, and Jeroen Hol. DOLPHIN: Integrated meeting support across liveboards, local and remote desktop environments. Arbeitspapier 828, GMD, D-53754 Sankt Augustin, February 1994.

[101] E. Torroja. *Logik der Form.* Verlag Georg D. W. Callwey, München, GER, 1961.

[102] Janet H. Walker. Document examiner: Delivery interface for hypertext documents. In *Proceedings of the ACM Hypertext'87*. ACM, 1987.

[103] Peter Whalley. Models of hypertext structure and learning. In David H. Jonassen and Heinz Mandl, editors, *Designing Hypermedia for Learning*, volume 67 of *Nato ASI Series F*, pages 61–67. Springer-Verlag, New York, NY, 1990.

ANNOTATION: See [49] for related work.

[104] Ludwig Wittgenstein. *Tractatus logico-philosophicus.* Vienna, AT, 1918.

[105] N. Yankelovich, B. Haan, N. Meyrowitz, and S. Drucker. Intermedia: The concept and the construction of a seamless information environment. *IEEE Computer*, 21(1):81–96, 1988.

Acknowledgements

I would like to thank my friends and companions throughout the years that it took me to finish my studies, in particular Gerald Hammerl, Doris Dobersberger, Andreas Lang, Sebastian Dietrich, Sandor Nagy, and Bernhard Krüpl, whom I am also indebted for commenting on preliminary versions of this thesis.

My family never hesitated to support me in my endeavors and gave me the backing that is necessary to achieve ones goals. Thanks also to my Muse for inspiration.

I especially wish to thank the people who shaped my work through valuable critique and comments. Christian Kühn spent many hours discussing ideas and contributed to many architectural issues. Paolo Petta always provided me with the latest news in hypermedia research and thoroughly commented on drafts. Wolfgang Slany, my supervisor, always encouraged my research work and supported me in all stages of my studies.

Finally, I sincerely thank Prof. Gottlob, who gave me the possibility to work in the scientifically challenging environment of the Christian Doppler Laboratory for Expert Systems.

Vienna, Austria — November, 1994 Marcus Herzog

Curriculum Vitae

Personal Data:

Name:	Marcus Alexander Herzog
Born:	May 14, 1969, in Vienna, Austria.
Parents:	Arch. Dipl. Ing. Gerhard and Anna Herzog.
Sister:	Ariane Herzog.
Nationality:	Austria.
Address:	Promenadegasse 57 D2/2, A-1170 Wien, Austria.
Email:	m.herzog@ieee.org, herzog@vexpert.dbai.tuwien.ac.at

Education:

1974–1979:	Volksschule Knollgasse, Vienna, Austria
1979–1987:	AHS BG XVI, Maroltingergasse, Vienna, Austria; passed with distinction.
since 1987:	Student of Architecture and Computer Science at the Technical University of Vienna.
since 1993:	Research assistant at the Christian Doppler Laboratory for Expert Systems c/o Department of Information Systems under Prof. G. Gottlob, Technical University of Vienna.

Work Experience:

Feb 1990:	Probationer at Siemens AG Austria, EK 41
Summer 1990:	Probationer at Siemens AG Austria, EK 41
1991:	Freelance programmer for Siemens AG Austria, EK 41
since 1993:	Research assistant at the Christian Doppler Laboratory for Expert Systems c/o Department of Information Systems under Prof. G. Gottlob, Technical University of Vienna.

Scientific Activities:

1991: Participant at the CAAD futures'91 International Conference for Computer Aided Architectural Design, Education, Research, and Application; ETH Zürich, Switzerland

1991: Participant at the ECAADE'91 International Conference, Munich, Germany

1993: Student volunteer at the 8th Austrian Artificial Intelligence Conference, FLAI'93, Linz, Austria

1993: Talk about 'World-Wide Web and related Information Services' in Linz at the Workshop for Doctoral Students in Fuzzy-Based Systems, organized by Hans Gamper and Bernhard Moser at the 8. Austrian Artificial Intelligence Conference about 'Fuzzy Logic in Artificial Intelligence'.

1993: Talk about 'Exploring Architectural Design Cases' at the Workshop on Intelligent Hypermedia, in conjunction with CIKM'93, Washington, D.C., USA

1993: Student volunteer at the ACM Hypertext'93 Conference, Seattle, USA

1994: Talk about the 'Department Information System at the Information Systems Department' at the Eurographics Symposium on Multimedia/Hypermedia in Open Distributed Environments, Graz, Austria

1994: Talk about 'Large Hypermedia in Architectural Design Environments' at the ED-MEDIA'94 — World Conference on Educational Multimedia and Hypermedia, Vancouver, Canada

1994: Student volunteer at the ACM ECHT'94 — European Conference on Hypermedia Technology, Edinburgh, Scotland

1994: Organization Committee member for the ICDT'95 — International Conference on Data Base Theory, Prague, Czech Republic

Scientific Affiliations: Member of ACM (Association of Computing Machinery), IEEE (Institute of Electrical and Electronics Engineers; Computer Society; Systems, Man, and Cybernetics Society; Communications Society),

List of Publications:

[I] Christian Kühn and Marcus Herzog. A language game approach to architectural typology. In Gernot Pittioni, editor, *Proceedings of the ECAADE'91*, October 1991.

[II] Marcus Herzog, Riccardo Peratello, Christian Kühn, and Wolfgang Slany. Exploring architectural design cases. In James Mayfield and

Charles Nicholas, editors, *Proceeding of the Workshop on Intelligent Hypertext, in conjunction with CIKM'93*, November 1993.

[III] Christian Kühn and Marcus Herzog. Modelling the representation of architectural design cases. *Automation in Construction*, 2(1):1–10, March 1993.

[IV] Christian Kühn and Marcus Herzog. On the role of hypermedia in architectural design education. In Tom Maver, editor, *Proceedings of the ECAADE'94*, September 1994.

[V] Eva Valsky, Marcus Herzog, Riccardo Peratello, and Wolfgang Slany. The Department Information System of the Information Systems Department at the Technical University of Vienna. In Werner Herzner and Frank Kappe, editors, *Proceedings of the Eurographics Symposium on Multimedia/Hypermedia in Open Distributed Environments*, Graz, Austria, June 1994.