INSTITUT FÜR INFORMATIK

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

HMT: Modeling Interactive and Adaptive Database-driven Hypermedia Applications

Dipl. Inform. Peter Zoller

Institut für Informatik
der Technischen Universität München

HMT: Modeling Interactive and Adaptive Database-driven Hypermedia Applications

Dipl.-Inform. Univ. Peter Zoller

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften

genehmigten Dissertation.

Vorsitzender:	Univ.-Prof. Dr. P.P. Spies
Prüfer der Dissertation:	1. Univ.-Prof. R. Bayer, Ph.D. / University of Illinois, Urbana
	2. Univ.-Prof. Dr. G. Specht Technische Universität Ilmenau

Die Dissertation wurde am 30.11.2000 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 25.4.2001 angenommen.

To my parents, Christa and Otto Zoller,
and Manuela.

Thank you for your support, patience and faith.

Kurzfassung

Angesichts der stetig wachsenden Größe und Anzahl von Anwendungen im World Wide Web spielt der Aspekt der Datenbankunterstützung für solche Systeme eine immer größer werdende Rolle. Zwar wurden die technischen Problemstellungen bei der Kombination von www und Datenbanken in den letzten Jahren weitgehend gelöst, es fehlt jedoch noch immer eine umfassende Modellierungstechnik zur Beschreibung von Anwendungen dieser Art. Etablierte Techniken und Methoden wie das Entity-Relationship-Modell oder die Object Modeling Technique bieten keine Unterstützung für typische Hypermedia-Strukturen wie z.B. Hyperlinks oder guided-tours, neuere Ansätze wie z.B. RMM oder OOHDM decken nicht alle relevanten Aspekte der Anwendungsentwicklung ab.

Die vorliegende Arbeit präsentiert die Hypermedia Modeling Technique, eine umfassende Methodik zur Modellierung datenbankgestützter Hypermedia Anwendungen. Der hier beschriebene Ansatz besteht aus einer Reihe von einzelnen Schritten wie z.B. dem ER-Design der Anwendungsdomäne, dem konzeptuellen Hypermedia Design, dem Design der Zugriffskontrolle oder dem temporalen Design, und behandelt dabei auch bisher unberücksichtigte Aspekte. Durch die strikte Trennung des HMT-Modells vom tatsächlich verwendeten Hypermediasystem kann die Hypermedia Modeling Technique nicht nur für Anwendungen im www, sondern auch für Anwendungen mit anderen Zielformaten wie z.B. XML oder PDF verwendet werden.

Zur Abbildung des HMT Modells auf eine geeignete Datenstruktur wird ein Metadaten-Repository entwickelt. Dieses Repository speichert die gesamte HMT Spezifikation einer Anwendung innerhalb des Datenbanksystems und garantiert dadurch maximale Integrität, Konsistenz und Wartbarkeit der Anwendung. Der HMT Ansatz wurde durch die prototypische Implementierung eines entsprechenden Case-Tools verifiziert, welches auf java servlets, JDBC und einem relationalen Datenbanksystem basiert.

Abstract

As applications in the World Wide Web are growing steadily regarding both size and functionality, database support plays a more and more important role. The technical questions of integrating databases and the WWW have been discussed and answered within the last few years, but the area of design methodologies still lacks a comprehensive framework for modeling these kinds of applications. Established modeling methodologies like ER or OMT cannot support hypermedia concepts like hyperlinks or higher level navigational structures like Guided Tours; new hypermedia modeling methodologies like RMM or OOHDM do not cover important aspects like personalization or temporal design.

This work presents the Hypermedia Modeling Technique (HMT), which has been developed in order to provide a comprehensive framework for modeling database-driven hypermedia applications. It consists of several design steps like ER design of the application domain, conceptual hypermedia design, authorization design (personalization) or temporal design. Due to the strict separation of HMT model and hypermedia system, HMT is not bound to the creation of WWW applications, but can also be used for other hypermedia platforms like XML or PDF.

For mapping HMT schemas to a computable format, a metadata repository has been developed. This repository allows storing the HMT specification of an application completely within the database system, thus ensuring integrity, consistency and easy maintenance of the application. The HMT approach has been verified by implementing a prototype CASE-tool. It is based on java servlets, JDBC and a relational database system.

CHAPTER 1

INTRODUCTION

A steadily growing interest in hypertext and hypermedia applications can be observed in recent years. Especially the rapid growth and acceptance of the World Wide Web has begun to turn this area of information technology into an aspect of every day's life. Current statistics estimate the number of web sites to exceed 73 Million[1] with more than 370 Million users worldwide[2]. What started as a distributed collection of mainly scientific information has emerged to a global marketplace with increasing economic importance.

Since the size of most web sites has been growing steadily, too, the shortcomings of web technology regarding the handling of large information collections have become evident. This refers to problems concerning overview and navigation in huge web sites, consistency of structure (dangling link problem) and content (outdated information), and costly maintenance of the application. In order to solve these problems, the integration of database management systems has been proposed and carried out successfully. Meanwhile every database vendor offers components and tools for connecting his database system to the World Wide Web.

But while the technical challenges of integrating World Wide Web and database systems have been solved already, the design process for the corresponding applications still lacks a comprehensive modeling methodology. A lot of applications are still created manually using programming languages or HTML extensions, at most supported by advanced editors or programming wizards. The following section will describe the importance of a structured design process for database-driven hypermedia applications and motivate our work.

1.1 Motivation

Modeling methodologies have proven to be useful and successful in a variety of IT areas, for example in object-oriented programming (Unified Modeling Language, UML) or relational database systems (Entity-Relationship Model, ER). Their simple and intuitive data models and the availability of corresponding CASE-tools are the fundamentals of their success. Both application development and maintenance benefit from the advantage of having a clear, well-defined syntax and semantics together with the possibility of automatic code generation by the use of CASE-tools.

[1] Source: www.glreach.com

[2] Source: www.isc.org

But existing modeling methodologies cannot be used for hypermedia applications, because they do not support hypermedia-specific elements like hyperlinks or complex access structures like Guided Tours.
As a consequence, a number of hypermedia design methodologies have been developed and proposed during the last decade, for example the Hypertext Design Model (HDM) [GPS93], the Relationship Management Methodology (RMM) [IKK98] or the Object-Oriented Hypermedia Design Model (OOHDM) [SRB96]. However, none of those has become a broadly accepted standard. One reason for this might be the fact that they all concentrate on specific aspects, but do not provide a comprehensive framework meeting all requirements of today's hypermedia applications. Typically unsupported aspects are:

- Interactive user interfaces
  In addition to interfaces for "passive" information presentation, a lot of applications also require interactive user interfaces. One example are search interfaces for directly querying the underlying data source. This approach guarantees higher performance and more exact query results than ordinary full text search on HTML pages. Another example are interfaces for manipulating the underlying data source. This is an important aspect for applications requiring distributed or mobile maintenance.
- Flexible hypermedia formats
  The very short life cycles of standards and techniques in the WWW demand great flexibility regarding the hypermedia target format of an application. This refers not only to the integration of new language extensions or versions, but also to the possibility of switching between completely different hypermedia formats with only minimal efforts. As an example, a hypermedia product catalogue for the World Wide Web might also be created as a PDF presentation that can be distributed on cdrom. If structure, navigation and layout of a hypermedia application are strictly separated, this migration can be achieved at limited costs.
- Access control
  More and more hypermedia applications contain not only information for the public, but also internal data that should be accessed only by selected users. Up to now, this aspect is usually handled on a physical level by the corresponding web server. Besides differences among the various web servers available, the main disadvantage of that approach is the distribution of application aspects among several components, which increases maintenance complexity and error probability. Integrating the aspect of access control into the central design and maintenance of an application would clearly improve this situation.
- Personalized information presentation
  Personalization of hypermedia information systems will be a central issue for a variety of future applications. This ranges from distance learning (with different presentations for different skills and interests) over intranet information systems (tailored to the needs of different departments) to e-commerce applications (providing individual user interfaces). Without support by the application design and maintenance process, personalization can hardly be achieved at reasonable costs.

- Temporal design
  Multimedia elements like graphics, sound or small video sequences become more and more popular as transmission rates in the internet are increased steadily. However, the synchronization and maintenance of these elements becomes rather labor intensive for large and complex applications. Although current web standards do not address the issue of temporal relations so far, a comprehensive hypermedia modeling methodology should provide support for the temporal design of the application.

Up to now, these aspects are not covered by existing modeling methodologies and corresponding CASE-tools, but have to be addressed manually using additional systems and components. For example, user authentication is usually still handled by the web server, whereas authorization is often done by the DBMS. This leads to costly development and maintenance of such applications with the risk of redundancies, inconsistencies and unintended side effects. A modeling methodology addressing these issues would improve development and maintenance of hypermedia applications significantly.

1.2 Objectives

This thesis will analyze the drawbacks and deficiencies of current hypermedia modeling methodologies and develop a new, comprehensive framework solving these problems. The Hypermedia Modeling Technique (HMT) is intended to support all important steps concerning the design of database-driven hypermedia applications. This includes not only the classical areas of information clustering and navigational design, but also new features like temporal design or personalization. The entire specification of complex hypermedia applications should be covered by this methodology without exception. The corresponding design steps, data models and mapping rules will be specified in detail.

The approach should be verified by the implementation of a prototype HMT CASE-tool allowing the specification and generation of database-driven web applications. Prerequisite for this tool is the development of a logical representation for the HMT schema that is created during the application design process. Based on this logical representation, the CASE-tool should generate the corresponding web pages. Important implementation objectives for the CASE-tool are flexibility regarding platform and database system and the development of techniques for optimizing the page generation process.

1.3 Outline

This thesis is organized as follows. Chapter 2 gives an introduction into current hypermedia modeling starting with a description of the Dexter Hypertext Reference Model and the Amsterdam Hypermedia Model. After presenting current hypermedia models and methodologies, the open issues and deficiencies of these approaches are identified.

Chapter 3 discusses technical issues regarding the integration of databases and the WWW. Besides the basic system architecture, possible approaches for page generation and user authorization are examined in detail. At the end, a short survey on commercial products is given.
Chapter 4 introduces the Hypermedia Modeling Technique (HMT). After an overview of the HMT design process, the core steps *conceptual hypermedia design*, *authorization design* and *logical design* and their data models are presented and discussed in detail.
A computable representation for HMT schemas is developed in Chapter 5. The HMT meta schema is used for storing all information about an application within a database repository. Besides the meta schema itself, the corresponding page and query generation process is specified.
Chapter 6 describes the prototype of a HMT CASE-tool. It focuses on the system architecture, specific implementation aspects like authorization or page generation, and identifies open issues.
This thesis ends with conclusions and an outlook on future work in Chapter 7.

CHAPTER 2

MODELING HYPERMEDIA APPLICATIONS

The use of software development models and methodologies is common in a variety of IT areas today. Two well-known examples are the Entity Relationship model (ER) for database design and the Unified Modeling Language (UML) for the design of object-oriented applications. Development models and methodologies help to build an abstract model of the application during the analysis and design phase, which can be used as a basis for both specification and implementation. This abstraction provides several advantages like rapid prototyping, reusability, distributed development or high extensibility.

The traditional life cycle model for software development [Boe76] consists of five subsequent phases with no or only few iterations between them: analysis, design, implementation, test and operation. While this linear approach is suitable for most traditional applications, the hypermedia design process needs significantly more iterations. The intensive use of multimedia elements and the central aspect of user interface design require a development team with a variety of distinct skills and abilities, which have to be integrated steadily. Together with the importance of rapid prototyping, this leads to a number of iterations in the design process of hypermedia applications. Additionally, new design goals gain more and more importance for hypermedia applications. One example is personalization of the application, which is subject to current research especially in the area of WWW information systems.
Several hypermedia design methodologies have been proposed especially during the last years, but so far no de-facto standard can be identified. These methodologies show considerable differences regarding number, order and content of their design steps, making it hard to describe a general life cycle model.

This chapter gives an overview on current hypermedia modeling techniques. The first section discusses the terms hypertext and hypermedia and their reference models. Afterwards, section 2.2 presents a very general specification of the typical hypermedia design process, and section 2.3 describes several design methodologies. Section 2.4 identifies open issues regarding hypermedia design methodologies, and section 2.5 ends this chapter with a short summary.

2.1 Hypertext and Hypermedia

The distinction between *hypertext* and *hypermedia* in literature is not always clear. Sometimes *hypermedia* is considered to be simply *hypertext* plus multimedia elements, and sometimes these expressions are even used as synonyms. Strictly speaking both interpretations are

incorrect. The following sections will give a clear specification for the terms *hypertext* and *hypermedia* and show their differences.

2.1.1. Hypertext – the Dexter Model

While written information is usually of sequential nature, the idea of hypertext is to allow non-sequential access for related portions of information. For this purpose, the information is divided into smaller units (called *nodes*), and relations between these units are represented by references (called *links*). A graphical user interface displays the information and allows activating these links, which usually originate and end at certain parts of a node (called *anchor*). This could be some keywords, sentences or images. A complete document can be regarded as a graph consisting of nodes and links.

In order to provide a standard hypertext terminology and to capture the important abstractions of typical hypertext systems, the Dexter Hypertext Reference Model has been published by Halasz and Schwartz in 1994 [HS94]. It consists of three layers named *run-time layer*, *storage layer* and *within-component layer*, the main focus is on the storage layer. Figure 2-1 shows the architecture of the Dexter Model with its three layers in an actual hypertext system.

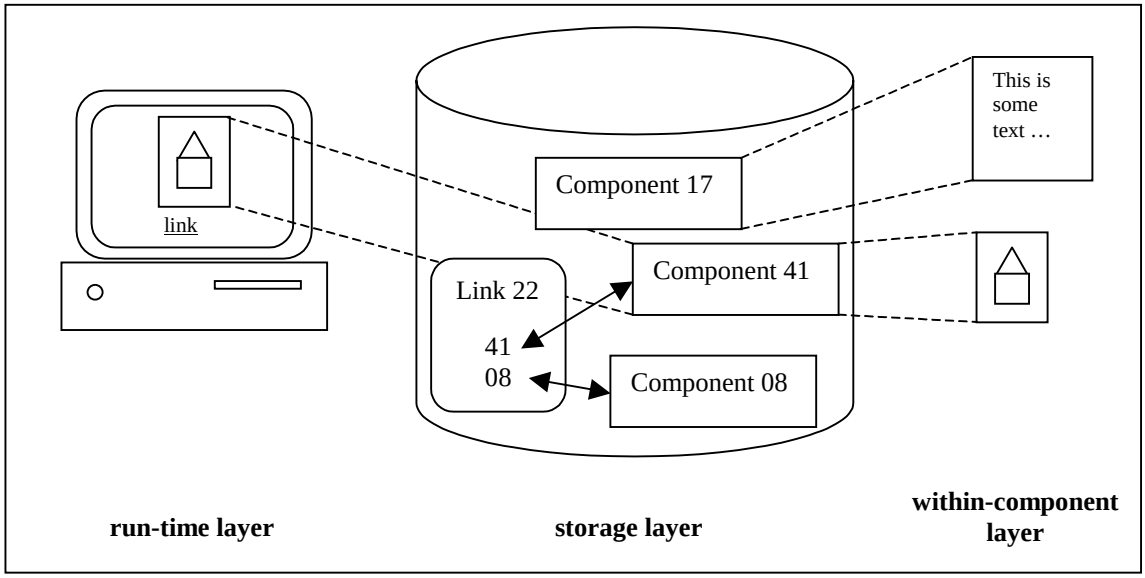

Figure 2-1: architecture of the Dexter Reference Model

The within-component layer is concerned with the structure and contents of the components of the hypertext network. The Dexter Model does not make any assumptions about these components; they may consist of data with arbitrary types.

The storage layer contains all information about links and anchors. The unit of addressing is a component, which can be an atom, a link or a composite entity. Atomic components typically correspond to nodes in a given hypertext system, composite components build a directed acyclic graph (DAG) and can be used to create tree structured documents. Each component has a unique identifier (UID) that is assumed to be uniquely assigned across the entire

universe of discourse. Components can either be referenced using this UID, or indirectly using component specifications (for example "all components containing the string *project*").
Links can be unidirectional or bi-directional, and reference either a component (*span-to-node* link) or an element within a component (*span-to-span* link). Since the link mechanism should not depend on the internal structure of a component (which is left to the within-component layer), the Dexter Model uses some kind of indirect addressing entity called *anchor*. An anchor consists of an *anchor id* and an *anchor value*, which could be regarded an offset within the component. The storage layer does not need to interpret the anchor value, it only uses the anchor id.

The run-time layer is responsible for the interaction with the user. The components and links of the storage layer are instantiated for the specific user, and a session entity is used to keep track of the user's actions. If the user leaves the hypertext system, the instantiated components are cleaned from the user specific cache and perhaps written back to the storage layer if changed by the user.

2.1.2. Hypermedia – the Amsterdam Model

At first sight, hypermedia simply seems to be a combination of hypertext and multimedia elements. Multimedia provides a variety of different presentation facilities, hypertext offers a convenient way of accessing the information in a content-based manner. But a closer examination shows that some hypertext concepts do not work very well with more complex, often time-based multimedia elements. Considering, for example, a short video sequence, how could we manage to provide a hyperlink pointing to different targets depending on the point of time within the video? And what happens when the user activates that link, does the video stop or continue? How can several time-based components be synchronized? These considerations show clearly that true hypermedia systems cannot only rely on hypertext concepts, but need additional support for the complex temporal relationships between hypermedia components.

The Amsterdam Hypermedia Model [HBR94] has been proposed as an extension to the Dexter Model adding multimedia and synchronization concepts.
Synchronization is provided for composite components on a coarse-grained and a fine-grained level. Coarse-grained synchronization is indicated by an offset symbol (see Figure 2-2) and specifies constraints between a composite component and its children, for example the relative starting time. Fine-grained synchronization allows specifying constraints between the children within a composite component, for example a synchronized termination of two elements, by using so called synchronization arcs (sync arcs).

Figure 2-2 shows a small sample scenario containing three composite components and six atomic components. The composite component labeled *comp_A* consists of another composite component (*comp_B*) and one atomic component (labeled *text*), which includes a hyperlink pointing to component *comp_C*. *Comp_B* consists of two audio tracks and a video sequence,

and the corresponding synchronization arc specifies that the second audio track should be started in time so that it terminates together with the video sequence.

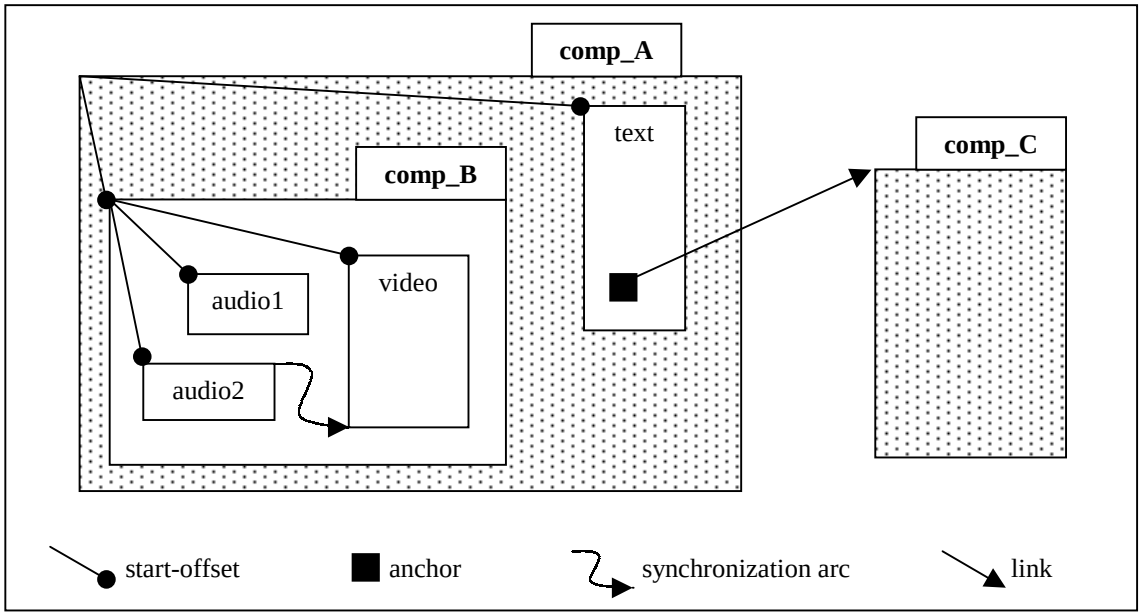

Figure 2-2: sample temporal specification using the Amsterdam Model

A second extension to the Dexter Model is the introduction of *link context*, which consists of a *source context* and a *destination context*. The *source context* is the part of the hypermedia application that is affected by the activation of a hyperlink. When the link operation is initiated, the source context can either be replaced by the *destination context* or be retained. In the second case, the author or the user may choose to continue or stop the source context presentation if it is a time-based component.

The third extension are so called *channels*, which are high-level presentation attributes. The Dexter model already allows defining presentation attributes, but these are only local (on the component level). The channels of the Amsterdam Model can be used to define application-wide presentation characteristics like font style and size, sound volume, scaling factors or preferred language.

As a summary, the Dexter Hypertext Reference Model and the Amsterdam Hypermedia Model provide an abstract framework for the development and comparison of hypertext and hypermedia models. Because they have been developed to cover all relevant theoretical concepts of hypertext and hypermedia systems regardless of technical limitations, currently no modeling methodology supports all their features. For this reason, we will simply speak of hypermedia systems from now on if there is no need to distinguish between hypertext and hypermedia applications.
One thing the Dexter and Amsterdam Model do not describe is the general modeling process itself, that means the sequence of design steps needed to develop an application. This aspect will be discussed in the following section.

2.2 Basic Hypermedia Design Process

Although there have been proposed several hypertext and hypermedia modeling methodologies mainly during the last decade, some kind of standardized modeling process has not emerged yet. Existing modeling techniques differ significantly regarding both number and content of their design steps, which makes it hard to identify a clear, universal structure. What they do have in common is some kind of meta structure describing a course sequence of design steps that are related to the traditional software life cycle [Boe76].

Requirements Analysis

This first phase identifies the requirements of the application regarding different aspects. As in every software development cycle, the application domain has to be specified by close investigation of the customer's business, and the layout of the user interface has to be developed.
A very important aspect for hypermedia applications is the identification and modeling of the target audience the system is developed for [Tro98]. Especially commercial applications can only be successful if tailored exactly to the user group identified as potential customers.

Conceptual Hypermedia Design

The conceptual hypermedia design covers the aspects of *information clustering* and *navigation* within the hypermedia application.
Information clustering is the process of grouping the information into meaningful units, often called nodes or documents. Several common guidelines are available for this design step. For example, it is considered to be important that conceptually equal elements are also presented in an equal manner [GMP95]. Additionally, a document should contain as much related information as possible, but it should be short enough to be displayed entirely on the screen without the user having to scroll the presentation.
The *navigational design* is responsible for the development of a user friendly, intuitive way of browsing through the application. A set of standard access patterns can be found in the related literature [IKK98] that covers the requirements of most applications: Index, Guided Tour and Indexed Guided Tour.

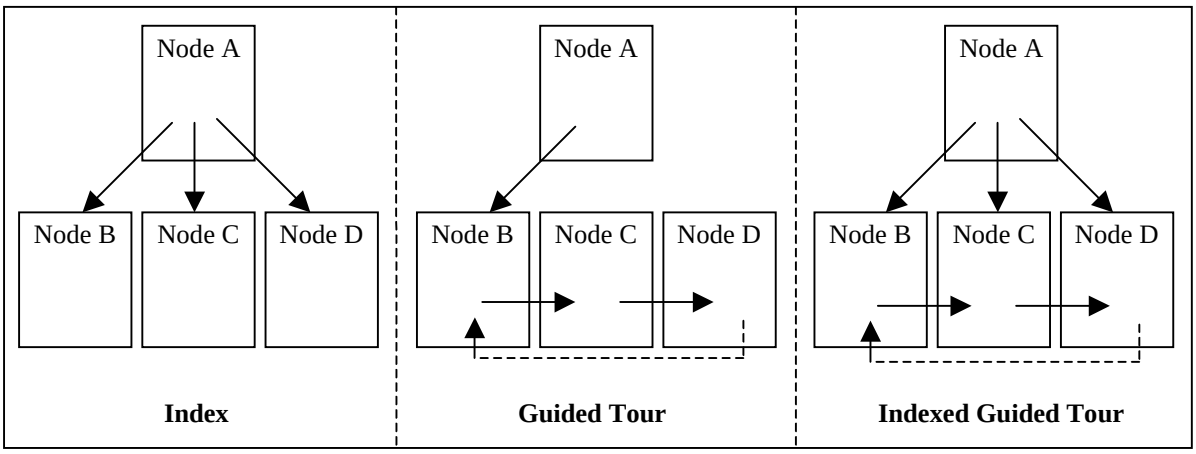

Figure 2-3: standard navigational patterns

An Index is a reference to a set of entities (nodes) where each entity can be accessed directly. In contrast to this, a Guided Tour provides only direct access to the first entity of a set, but each entity provides links to the next and perhaps previous element. In a cyclic Guided Tour, the last element points back to the first entity. An Indexed Guided Tour combines these two concepts. Figure 2-3 shows a graphical description of these three basic navigational patterns.
Similar to information clustering, also the navigational design should be consistent in a way that equal structures should be accessed by equal navigational patterns.

Implementation

The implementation step covers a variety of aspects, which are usually divided into several design steps in an actual methodology.

If a CASE-tool is used, the conceptual hypermedia schema is mapped to a logical representation that can be stored on the computer. In most cases, this mapping is not a task that has to be accomplished by the user, but is automatically handled by the CASE-tool. The data format for representing a conceptual hypermedia schema can be chosen freely, but in general two approaches exist: a specification language with a corresponding parser or a meta data repository. A lot of systems use the first alternative, but especially for database-driven applications the second approach might provide a number of advantages as described in Chapter 5.
Another typical step is the specification of additional information not contained in the conceptual hypermedia model. This refers to both functionality and layout. For example, a context-sensitive help system might require the specification of individual man pages, or a document displaying a list of items might be assigned a maximum number of elements being presented. Considering layout aspects, typical additional specifications are font properties or window and image sizes.
When using a CASE-tool, the application will be generated automatically. One approach is to generate and store the application initiated by the administrator (offline generation). As an alternative, the application can also be created dynamically at runtime when a user requests the corresponding information (online generation).

Evaluation

At the end of the design process, the hypermedia application has to be evaluated. Besides correctness and consistency, aspects like acceptance of the user interface, technical limitations regarding the execution of the application, or reusability play an important role in the evaluation of hypermedia applications [GMP95].

After this coarse and abstract description of the typical hypermedia design process, the following section discusses some actual modeling methodologies in detail.

2.3 Selected Models and Methodologies

A number of hypermedia modeling methodologies have been proposed and described, sometimes accompanied by prototype CASE-tools and reference applications. They usually provide a step-by-step description of the hypermedia design process, the corresponding data model(s) and additional design guidelines or mapping rules. Well-known examples are HDM [GPS93], OOHDM [SRB96], RMM [IKK98], WSDM [Tro98] and the ARANEUS design methodology [AMM98]. The newly proposed HDBM [Som00] is an extension to the RMM design process especially developed for database-driven applications.

Additionally, there are some research projects with a slightly different focus providing variations of these methodologies. Projects like STRUDEL [FFK+97], TSIMMIS [CGH+94] or SIMS [ACH+93] address the topic of integrating heterogeneous data sources into hypermedia applications. This is typically done by providing wrappers for each data source, which give a unified view on the information. A central mediator is used for distributing the queries on the different sources and integrating the corresponding results. Since these projects do not provide significant improvements regarding the aspect of hypermedia design, we will focus on the design methodologies mentioned in the first paragraph. The following sections will give a short introduction into HDM, RMM, ARANEUS and HDBM.

2.3.1. HDM

Presented in 1993, the Hypertext Design Model (HDM) builds the basis of various other design methodologies like RMM or OOHDM. It consists of five design concepts: *Entities* and *entity types*, *components*, *perspectives*, *units*, and *links*.

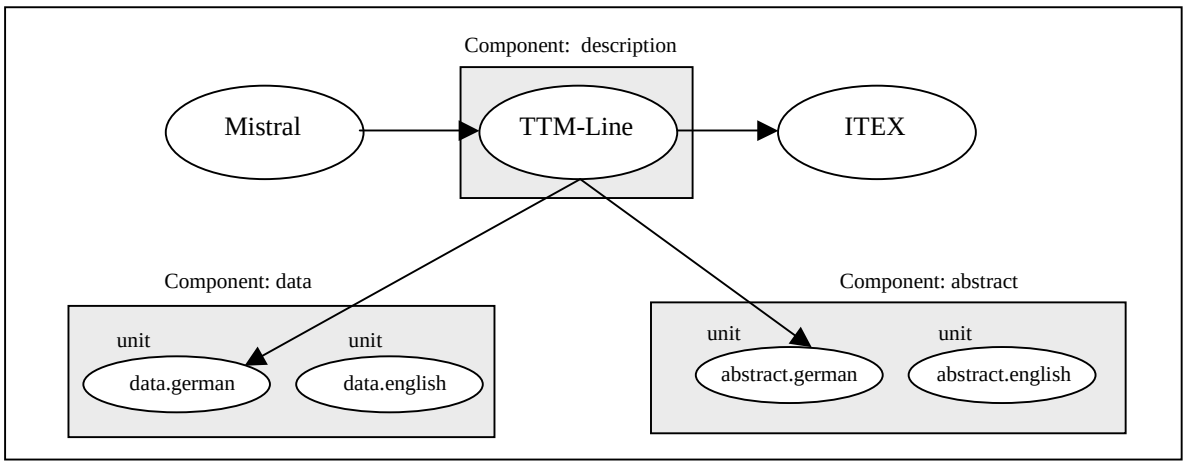

Figure 2-4: sample HDM schema

Entities and *entity types* correspond to the elements with the same names known from the ER model. In contrast to ER entities, HDM entities consist of complex hierarchical substructures (*components*) and are related to other entities by *application links* instead of relationships.

Components are abstract elements always related to a certain entity. They consist of a set of atomic elements called *units*. These units can be considered as hypertext nodes consisting of an identifier and a body, which contains the actual information.
Links in HDM are separated into *structural* and *application-links*. Structural links relate different components of an entity, whereas application-links reference other entities related to the current entity.
Perspectives allow presenting the content of an entity in different ways. This can be achieved by providing separate units in every component for each perspective. If a certain perspective is chosen, the components (and thus the entity) will be presented by the corresponding unit.

Figure 2-4 shows a sample HDM schema for the presentation of research projects. Each project entity (for example TTM-Line) contains a *description* component with two links referencing a *data* component (containing for example title, begin, end and manager of the project) and an *abstract* component (containing a textual description of the project goals). For two possible perspectives *German* and *English*, each component contains the corresponding unit that will be used to display the content for the perspective selected.

2.3.2. RMM

The Relationship Management Methodology (RMM) [IKK97a, IKK98] is one of the best-known hypermedia modeling techniques. Based on the Hypertext Design Model (HDM) and the Entity Relationship Model (ER), it provides a comprehensive data model and a seven-step design methodology that is defined as follows:

1. Requirements analysis: The designer begins with determining the information domain, the application functionality, the expected users and how the application will be used.

2. ER diagram: The information structure of the application is modeled based on the familiar principles of ER design.

3. Application diagram top-down: This step involves a top-down specification of the application diagram, that means the designer specifies a global view of the final application. This view contains only the names and descriptions of the application's top-level presentation units plus the basic hyperlinks between them. The corresponding design primitive is shown in Figure 2-5.

4. M-Slice design: The presentation units from the top-down application diagram are now decomposed into smaller units called *m-slices* (the "m" in "m-slice" is derived from the Russian *Matreyshka* dolls symbolyzing the nested nature of RMM). These m-slices are used for grouping attributes of a given entity into meaningful units, and relations between different m-slices are modeled with special access structs. This design step is the main part of the RMM design process and will be discussed more detailed later.

5. Application diagram bottom-up: By combining the m-slices created in the previous design step, the designer generates an application diagram in a bottom-up fashion and compares this to the top-down version of step 3. Mismatches are corrected by iterative application of the design steps 3 to 5.

6. User interface design: After having completed the m-slice and application diagram design, the user interface is created using arbitrary tools.

7. Implementation: Finally, the designer implements the application either manually or by using CASE-tools.

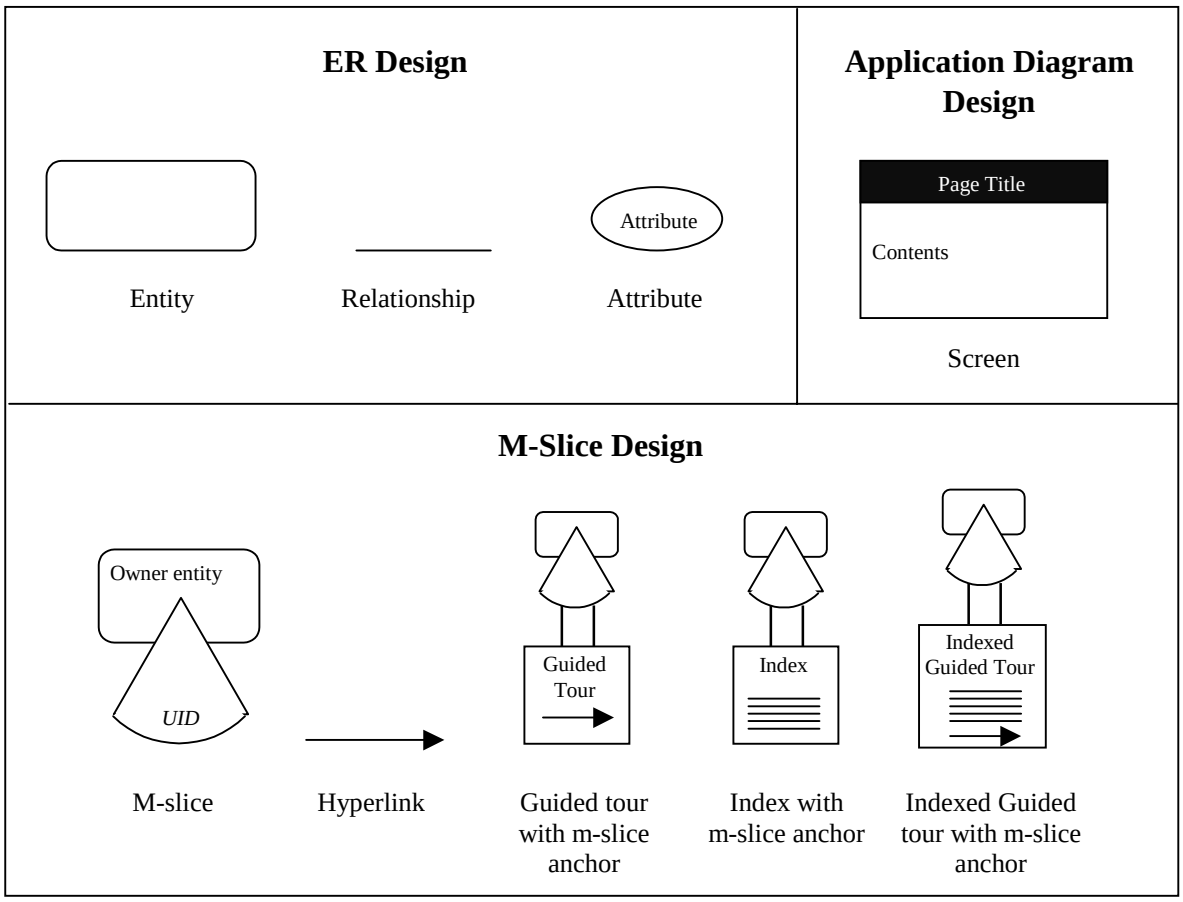

Figure 2-5: basic RMM design primitives

The Relationship Management Data Model (RMDM) consists of ER domain primitives, special RMDM domain primitives and a set of access primitives as shown in Figure 2-5. The ER domain primitives are entities, attributes, 1:1 and 1:N relationships; M:N relationships are mapped to two 1:N relationships.
An additional RMDM domain primitive is the m-slice, which is the central element of the Relationship Management Methodology. M-slices are used to group attributes of a given entity into meaningful units. Each m-slice has a unique identifier, a related entity type (called owner entity), and might contain attributes or other m-slices with the same owner entity.

Access primitives are used to define the navigation between the m-slices of an application. Available in RMM are link, index, Guided Tour and indexed Guided Tour. A description of these access structs can be found in section 2.2.

A sample RMM schema specifying a project overview m-slice is shown in Figure 2-6. The sample ER scenario consists of the two entity types *projects* and *employees* and a relationship type *works_in*. The *project_overview* m-slice has the owner entity *projects* and contains the attributes *title*, *abstract*, *begin*, *end* and *image*. The set of employees working in that project is determined using the *works_in* relationship type and an index with the anchor m-slice *name*. This anchor m-slice simply displays the name of the employee and a hyperlink pointing to the employee's *homepage*.

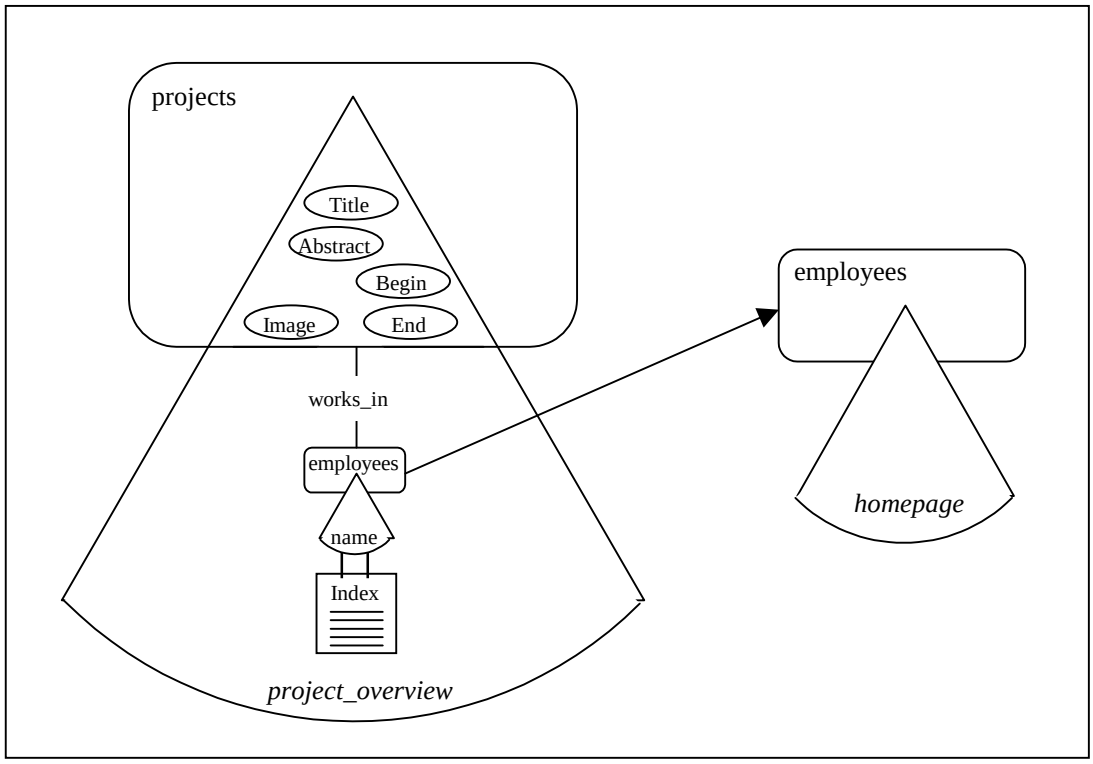

Figure 2-6: RMM sample schema

A more detailed description and examples of the RMM slice design process can be found in [IKK97a, IKK98], the description of the prototype CASE-tool RMCase is available at [DI95].

2.3.3. ARANEUS

The ARANEUS web design methodology [AMM97, AMM98] focuses on the creation and maintenance of database-driven web applications. It is based on HDM, RMM and OOHDM, and offers a design process (see Figure 2-7) consisting of two main tasks: database design and hypertext design. Each of these two tasks is divided into a conceptual and a logical design step.

The first step is the design of a conceptual database schema describing the application domain. For this purpose, the well-known Entity-Relationship model is used. Based on this schema, the logical and physical database design is performed according to standard rules and techniques [BCN93, EN94]. In parallel to the logical database design, the hypertext design is performed. Similar to RMM, the ARANEUS hypertext design is based on the ER schema developed during the conceptual database design step, but here it is divided into three different levels: *hypertext conceptual design*, *hypertext logical design* and *presentation design*.

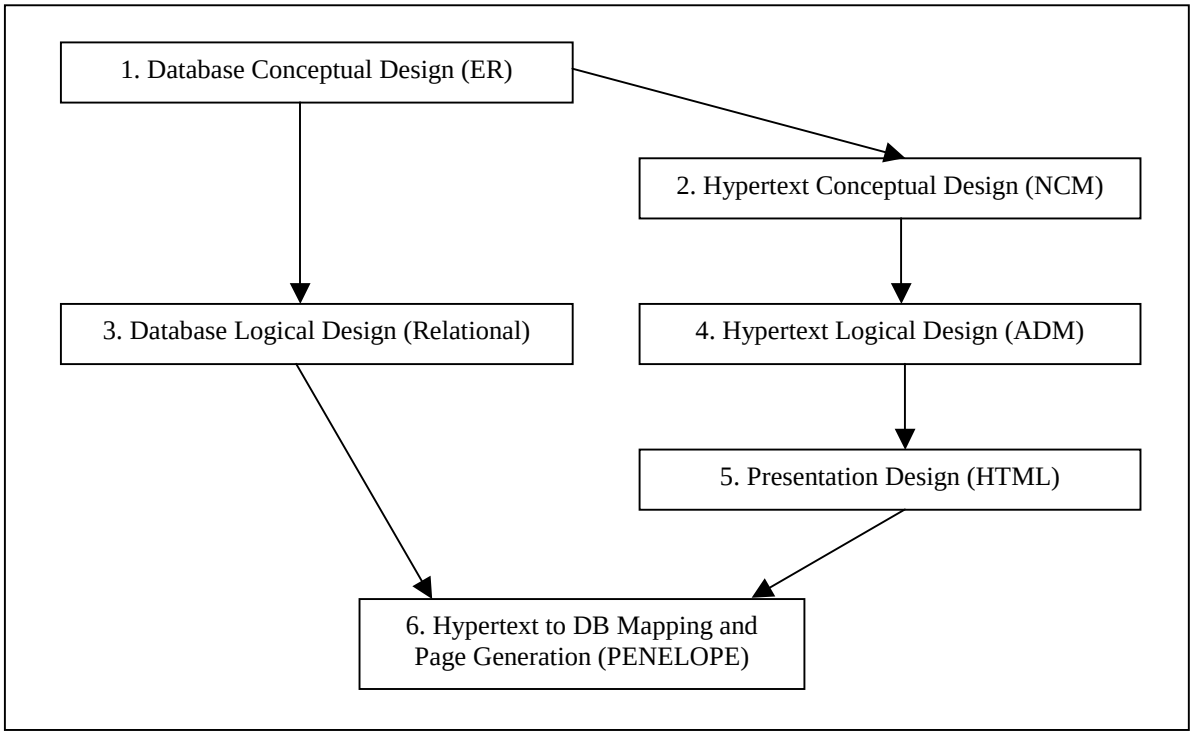

Figure 2-7: the ARANEUS design process

The conceptual design level abstracts the typical hypertext features like nodes and navigation, and defines how entities are to be aggregated. The corresponding data model used for this design step is called *Navigation Conceptual Model* (NCM) and is based on the RMM data model described in the previous subsection. We will discuss the NCM more detailed later.
The logical level is committed to the World Wide Web and describes how *pages* of the web site are organized. It ignores all physical aspects of these pages and concentrates on features like grouping relevant information and organizing their structure (for example (nested) list or flat page). For this purpose, the hypertext conceptual schema is mapped to a logical presentation based on the ARANEUS Data Model (ADM). Entities and aggregations are translated into so called *page-schemes*; directed relationships are mapped to links.
The presentation design step deals with all aspects regarding the layout of the web application. The ARANEUS design methodology does not investigate that topic further, but assumes that the output of this design step is an HTML page template.

Finally, the *Hypertext to DB Mapping and Page Generation* step generates the HTML pages from the database content. In order to fulfill this task, the mapping between hypertext and database has to be accomplished first by generating database views containing all information necessary for a specific page. After that, the PENELOPE language is used to generate the HTML pages either online or offline (more information on the generation of web pages is given in section 3.2).

After this short overview, we will take a closer look at the Navigation Conceptual Model (NCM) of ARANEUS. A NCM schema is derived from the ER schema of the application domain and is built from six design constructs: *Macroentities*, *Union Nodes*, *Aggregation Nodes*, *Directed Relationships*, *Symmetric Relationships* and *Aggregation Links*. Their corresponding graphical representations are shown in Figure 2-8.

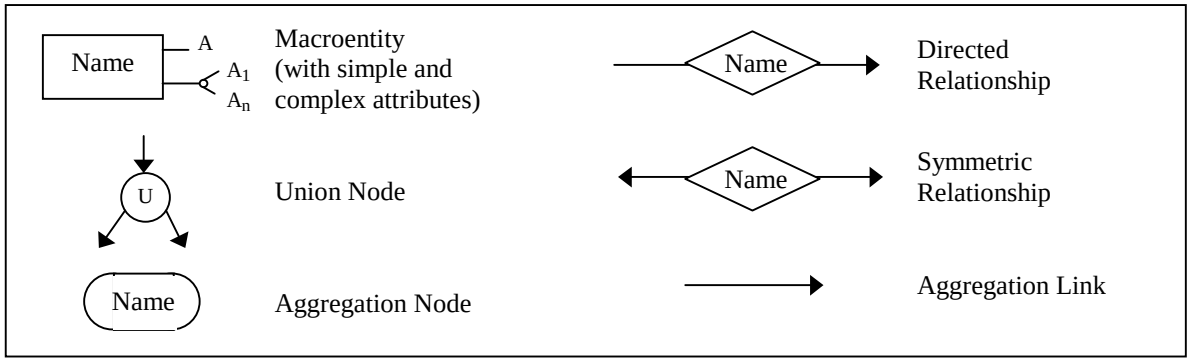

Figure 2-8: basic NCM design primitives

Macroentities are used to represent independent hypertext objects. They are derived either from ER entities or from views over entities and relationships, and their name is usually identical to the name of the corresponding entity. Since views are allowed, the attributes of a macroentity might be multivalued. *Union nodes* are used to model types representing the union of different macroentities. These union nodes are typically obtained when mapping ER hierarchies or modeling navigations involving different macroentities.
Directed relationships are usually derived from ER relationships and describe navigations between macroentities. *Symmetric relationships* are considered to be composed of two asymmetric directed relationships. Aggregations occur as the counterparts of ER hierarchies or can be explicitly introduced in order to organize the hypertext information. *Aggregation links* are used to describe what macroentities are used to form an *aggregation node*. Partial aggregations can be defined by attaching labels to the links that are associated with predicates on the instances of the destination node.

Figure 2-9 shows the NCM specification of a small application based on a sample ER scenario including the entities *research project*, *manager*, *scientist* and the relationship *project_staff*. The aggregation node *Department* acts as the entry point to the hypertext application and provides links to the *Project* macroentity and another aggregation node called *Staff*. The *Staff* node is built from the *Scientist* and *Manager* macroentities, and the set of

people working in a certain project is specified by the *Project_staff* relationship referencing a union node built from *Scientist* and *Manager*.

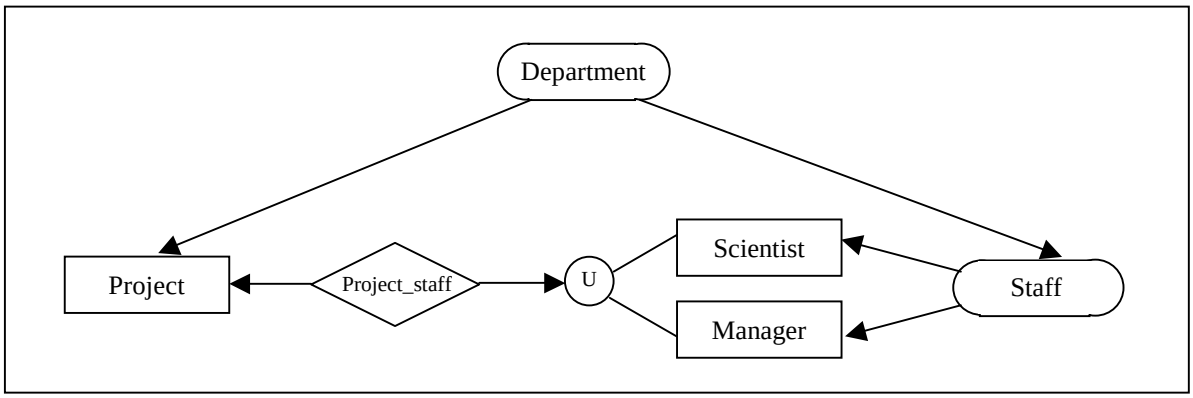

Figure 2-9: sample NCM schema

In the hypertext logical design step, NCM schemas are mapped to ADM schemas. The available design constructs of an ADM schema are shown in Figure 2-10. The *page schema* is the fundamental design primitive that represents a set of web pages with common structure and features. A special case is the *unique page schema*, which has only one instance, that means no other page has the same structure. The contents of a page schema are defined by the attributes they contain. Besides *text attributes* and *image attributes*, also *link attributes* can be used. Links are modeled as pairs of *anchors* and *references*, where the reference is the URL of the destination page and the anchor is a simple text or image element.
Similar to the NCM schemas, ADM schemas also offer a *heterogeneous union type*. Additionally, *Lists* can be used to represent (ordered) collections of tuples originating from multivalued attributes. Closely related to these lists are the so-called *form types*, which represent HTML forms. Such HTML forms are interpreted as a virtual list of tuples and can be used, for example, to restrict long lists.

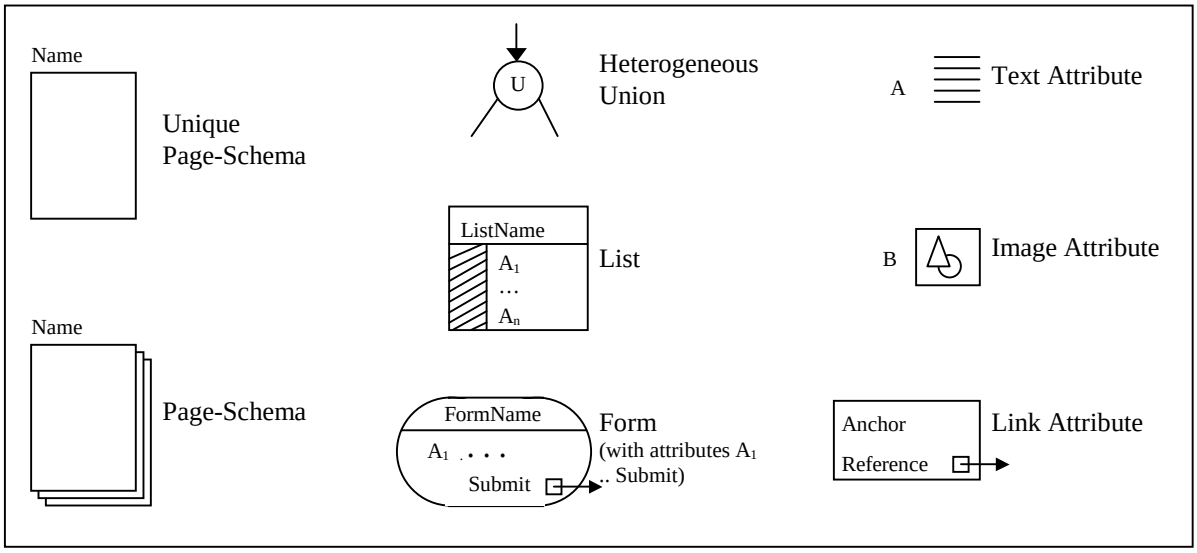

Figure 2-10: basic ADM design primitives

When mapping of NCM schemas to ADM schemas, the following rules apply: Macroentities are mapped to page-schemas, directed relationships are mapped to links between page-schemas, and aggregation nodes are mapped to unique page-schemas. Unions are mapped to their corresponding ADM primitive and relationships are translated into links between page-schemas.
These ADM schemas are used in the final design step for generating the HTML application by use of the PENELOPE language. Information about this language and a more detailed description of the ARANEUS design methodology together with several example schemas can be found in [AMM98].

2.3.4. HDBM

The Hypermedia Database Methodology (HDBM) [Som00] is based on well known conceptual and logical data models from the area of database and hypermedia systems. It offers a modeling process designed especially for database-driven hypermedia applications by using a slightly modified version of RMM for the hypermedia design step and the traditional design process for relational database systems. Figure 2-11 gives an overview of the basic HDBM design process.

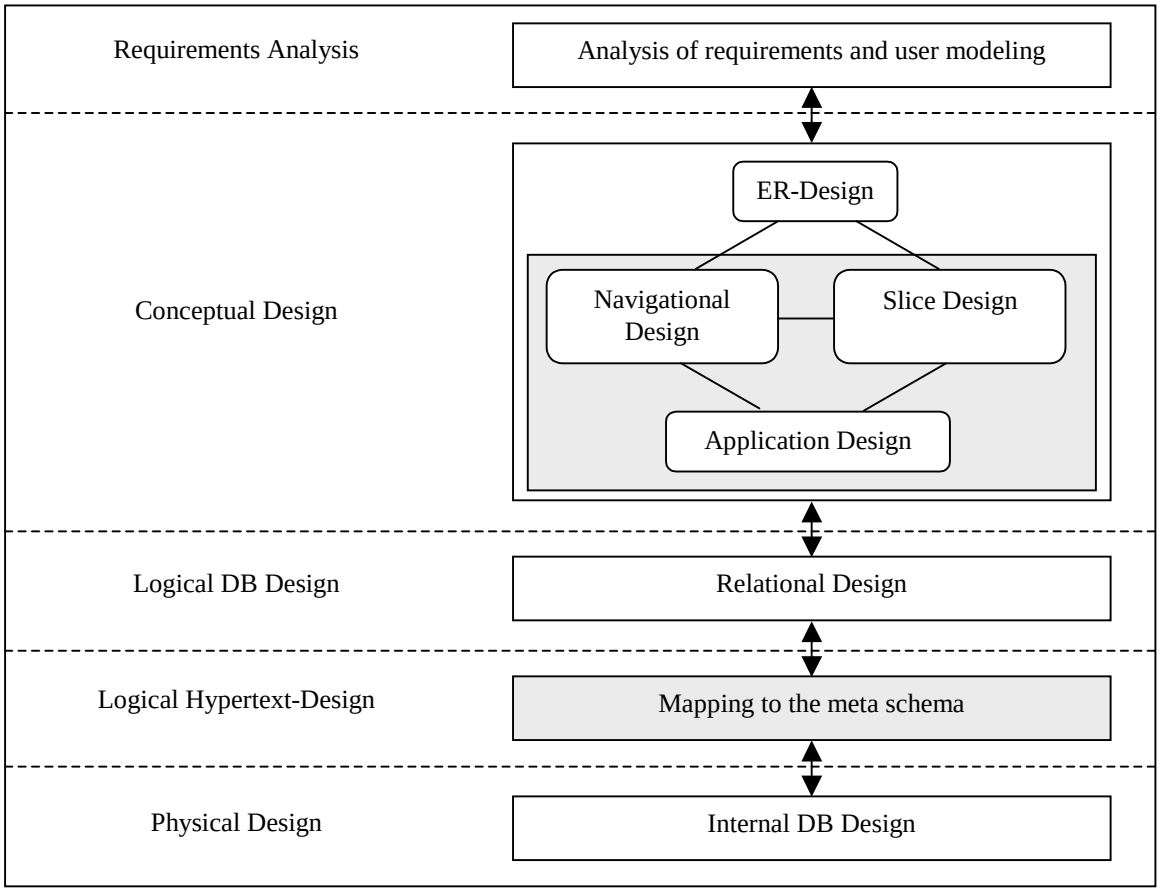

Figure 2-11: HDBM design process

The first step in the HDBM design process is the requirements analysis, followed by the conceptual design step, which can be divided into two parts. The first part is the specification of an ER schema of the application domain that will be used to create a relational database in a later design step. This ER schema is the basis for the second part of the conceptual design step, which consists of slice design, navigational design, and application design. The application design provides a global view of the hypertext structure on the level of HTML pages and corresponds to the application diagrams of RMM, slice design and navigational design correspond to the m-slice design step of RMM.

The logical DB design step transfers the ER schema of step 2 into a relational database schema using known mapping rules. During the logical hypertext design step, the layout of the application is defined and, together with the conceptual hypertext schema, mapped to a relational representation, which is called the HDBM meta schema. Finally, the physical design step deals with all aspects of database administration and tuning, often also called physical database design.

For the HDBM conceptual design, three additional elements are added to the set of design primitives of RMM: *presentation unit*, *head slice*, and *external index*.

- *Head slices* are used to handle the HTML generation for SQL queries that are not known at design time (ad-hoc queries). In order to be able to produce sensible HTML output for these queries, a default slice has to be specified for each entity, because there might be several slices available for a certain entity. This default slice is called *head slice* and will be used for the presentation of entities resulting from adhoc queries.
- The second extension to the slice primitive is the *presentation unit*. In RMM, m-slices might be nested and there is no distinction between top-level and lower-level m-slices. If the application has to be materialized, the top-level m-slices have to be identified. In order to avoid the costly computation of the relevant m-slices, the corresponding elements are modeled as *presentation units* during the conceptual design phase.
- The third extension refers to the RMM access primitives. The *external index* allows referencing a set of related entities by providing a link to a new page that contains the list of these entities. With the ordinary index of RMM (now called *internal index*), the referenced entities are always displayed directly on the current page.

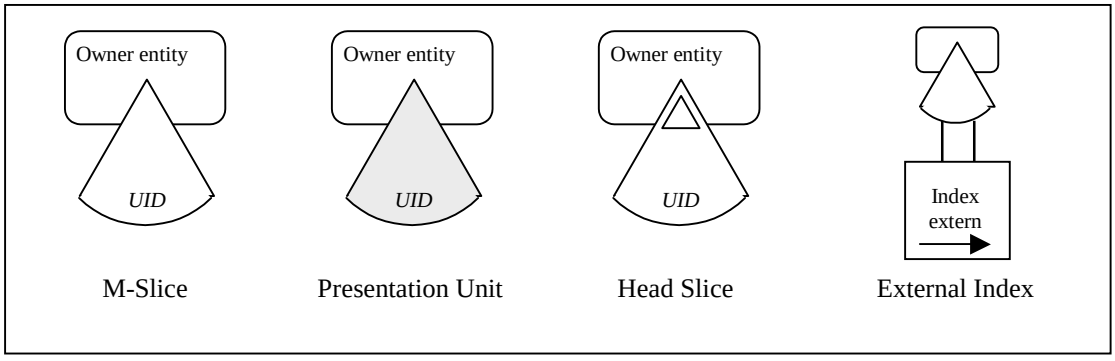

Figure 2-12: new HDBM design primitives

The original m-slice primitive and the three new design primitives of HDBM are shown in Figure 2-12. For a sample HDBM diagram, the m-slice primitives of Figure 2-6 have to be replaced with presentation unit primitives.

2.4 Deficiencies and Open Issues

Independent of the individual advantages and disadvantages of each approach described in the previous sections, there are some essential aspects that are not covered by any of these methodologies. This refers to interactive user interfaces for querying and updating the underlying data source, personalization of the application, authorization aspects or the specification of temporal relations and dependencies. The following sections will discuss these open issues and point out why they are important for a variety of hypermedia applications. This will show the motivation for the introduction of the Hypermedia Modeling Technique (HMT) in Chapter 4.

2.4.1. Interactive Interfaces

Current modeling methodologies focus only on the aspect of passive information presentation, but ignore the need for (inter)active interfaces for querying or updating the application's data source. Especially the growing number of World Wide Web applications contains a significant amount of systems requiring more than just pages for passive information presentation. Examples are the online information systems built by FORWISS [Ab00, Ttm00] or the web site of the Deutsche Bahn AG [DB00], which offer multiple search interfaces. Most hypermedia document formats like HTML or XML are not suited for efficient searching; querying the information source would be the better alternative, especially if a database system is used to store the information.
Another important aspect especially of WWW applications is the availability of interfaces for maintaining the underlying data source. This provides not only the advantage of needing only one interface for both retrieval and maintenance, but offers also the possibility of distributed, mobile administration of the system.
Of course not all hypermedia platforms support these interactive interfaces, but the hypermedia modeling methodologies should provide appropriate concepts for handling these aspects.

2.4.2. Authorization

Before hypermedia applications provide interfaces for manipulating the underlying data source, a solid authorization concept is necessary in order to protect the system's information from unauthorized access. Since no modeling methodology supports user authorization so far, this task has to be accomplished separately. For example, database-driven web applications might be protected by using the authorization features from the web server or the DBMS as

described in section 3.3. But this approach does not only increase the efforts needed for creation and maintenance of such a system, but also embodies the danger of inconsistencies and errors, which often are hard to detect.
Integrating the authorization design for an application into the modeling methodology eases the development and maintenance of such systems and provides a central, consistent administration environment.

2.4.3. Personalization

As soon as hypermedia applications are developed for a large and heterogeneous audience, the aspect of application personalization becomes an important issue. For example, a lot of search engines in the web meanwhile offer personalization components where the user can customize basic search and presentation options. In many other places, the users are not even aware of the fact that personalization is used. As an example, a large internet bookstore collects information about a user's preferences when ordering books. The next time this user visits the store, automatically books of the corresponding genre are offered on the starting page. Besides www information systems and stores, also other hypermedia applications might benefit from personalization. Typical examples are hypermedia presentations on large fairs where the user might choose between different languages or levels of knowledge like *layman*, *advanced* or *expert*.

Two different forms of personalization are distinguished [MG+87]: *Information filtering* and *document adaptation*.
Information filtering describes the process of selecting the appropriate information for a user from a pool of data sources. This can happen by correlation between the content of an object and the user's preferences (cognitive filtering), by analyzing the likes of other people with similar tastes (social filtering), or by using cost factors like network bandwidth (economic filtering). In other words, information filtering restricts the set of documents presented to the user.
In contrast to this, *content adaptation* tailors the content of a document to the interests and knowledge of the requesting user, for example due to cultural or geographical background. For this purpose, the information has to be available in a highly structured representation.

Since hypermedia modeling techniques are used to design documents and the access paths between them, information filtering is not relevant at this level. However, content adaptation within a document is an aspect that could be addressed during the hypermedia design process, but no existing modeling methodology covers this issue so far.

2.4.4. Temporal Design

Considering the definition of hypermedia from section 2.1.2, it becomes evident that every true hypermedia design methodology has to support some kind of temporal design. Although the area of temporal relations and temporal design has been subject to research for some time

[All83, RJM+93, SDK96, Jou97], this aspect is still not covered by current hypermedia modeling methodologies. The main reason for this situation could be the fact that most of the modeling techniques developed lately are focused on the World Wide Web. Since this platform does not support temporal specifications yet, there has been no need to care for this aspect so far. But considering the rapid changes of standards and techniques in the World Wide Web together with the increasing use of hypermedia elements for www applications, the capability of modeling temporal aspects might be an essential feature of future www design methodologies.

2.5 Summary

This chapter introduced the basic concepts and techniques for modeling hypermedia applications. At the beginning, the differences between hypertext and hypermedia have been identified by describing the Dexter Hypertext Reference Model and the Amsterdam Hypermedia Model. The essential result of this comparison was the finding that hypermedia is more than simply hypertext plus multimedia.

The next section provided a coarse description of the general modeling process more or less common to all existing hypermedia design methodologies. The basic steps are requirements analysis, conceptual hypermedia design, implementation and evaluation. Depending on the actual methodology used, the implementation step is usually divided into several other design steps like logical hypermedia design or layout design.

Afterwards, some important and interesting hypermedia modeling methodologies have been discussed in detail. Among others, the Hypertext Design Model (HDM) and the Relationship Management Methodology (RMM) have been described as both early and important approaches in that area.

The chapter ended with a precise identification of deficiencies and open issues regarding hypermedia design methodologies. Currently no approach in that area covers aspects like interactive user interfaces, authorization, personalization or temporal design. The relevance and importance of these concepts for hypermedia modeling methodologies has been described as a motivation for the development of the Hypermedia Modeling Technique (HMT).

Before introducing the HMT in Chapter 4, we will first discuss some basic architectural issues regarding the integration of databases and the World Wide Web in the following chapter.

CHAPTER 3

DATABASE-DRIVEN HYPERMEDIA APPLICATIONS IN THE WWW

With the rapid growth and acceptance of the World Wide Web, the size of most websites has grown significantly. The traditional approach of storing a website as a collection of HTML pages in the local file system becomes more and more insufficient, because it offers no support for managing large information systems and leads to a number of problems like dangling links (links with an invalid URL), work intensive and error prone updates, inconsistencies regarding layout and structure, and outdated or redundant information.
Having been designed to support efficient storage, integrity, consistency and easy maintenance of large information sources, database systems can solve these problems for most web applications. This refers not only to the public WWW, but also to intranets of companies and organizations where often database systems have already been used for years. Also extranets and B2B portals (non public www areas for business partners) are a typical application scenario for the usage of databases connected to the World Wide Web.

This chapter investigates various aspects concerning the connection of databases and the WWW. Different architectures are presented and compared, the page generation techniques are discussed, the access control concepts of both technologies are examined, and a short survey on commercial products in this area is presented.

3.1 Architectures

The connection of databases and the web can be classified according to several characteristics and different points of view. One approach is to classify the applications according to conceptual characteristics like type of access (read/write), security level, user identification or session management [Loe98]. Another possibility is to investigate the basic technical aspects like the connection type (connection on server-side or client-side), the document specification technique (HTML templates, binaries, scripts) or the document generation technique (on the fly or materialized HTML views) [Zol96].
We consider the first type of classifications (the conceptual characteristics) to be higher-level classifications, because they depend on technical characteristics. For example, session management or user identification of an application is probably different depending on whether a client-side connection or a server-side connection is used. We therefore distinguish

the various approaches of connecting databases to the World Wide Web primarily according to the connection type.

3.1.1. Connection on Server Side

The traditional approach of connecting databases to the web is the server-side connection. The web server handles all the communication with the database system and converts information from the database into HTML pages. The client (the web browser) only communicates with the web server; it has no direct connection to the database system. This approach is used in most database-driven web applications, for example the three online information systems developed at FORWISS [Ab00, Mli00, Ttm00] or the online timetable of the Deutsche Bahn AG [DB00]. The main advantages of this technique are:

- No special requirements on client-side
  Arbitrary web browsers can be used to access the database, because only standard HTTP is used for communication. This is a very important aspect especially for applications aiming at maximum availability.
- Easier development, maintenance and optimization
  The application has to be developed only for one platform and does not have to care about the various possible client platforms or browser versions. This eases development and maintenance, and allows optimizing the application according to the local situation.
- Seamless integration into security frameworks (firewalls)
  Firewalls are used in most companies and organizations in order to protect the local network from foreign intrusion and attacks. Usually these firewalls are configured very restrictive and often allow only HTTP and a few other protocols to pass. Databases connected on server side can be integrated in such environments without problems. A client-side connection with a proprietary communication protocol would either require to change the security configuration to a lower level or to use a host outside the secured network for this application.
- Standard access statistics
  Analyzing the web server log files is important for improving a web site's structure and contents according to the users' preferences and behavior. If databases are connected on server side, the web server's access statistics also contain all database requests. Client side connections bypass the web server and the corresponding user requests were either lost or had to be logged and analyzed separately.

But concentrating all communication and application logic on server-side has also some disadvantages:

- Server becomes bottleneck
  The server has to handle requests for both static HTML pages and dynamic pages generated from database contents. Especially the latter can be complex and time consuming requests slowing down the server. A distributed architecture with multiple web

and application servers should be used for high traffic websites with server-side database connection.

- Costly session and transaction management
  Due to the stateless nature of the HTTP protocol, sessions and transactions across more than one HTML page require the implementation of a complete session and transaction manager on server side. If the client were connected directly to the database system using a stateful protocol, the session and transaction management would be handled by the dbms itself.
- No information processing on client side
  The server side approach restricts the implementation of the user interface to the capabilities of HTML and perhaps JavaScript. One drawback of this technique is the lack of possibilities to check the user's input before it is sent to the server. Errors or missing information are not detected at once but can only be reported with the next HTML page after the user has submitted his data. It is also not possible to offer context sensitive menus where the options of one menu depend on the user's selection in another one.

From a technical point of view, four different interfaces for connecting a database to the web on server-side can be distinguished: The Common Gateway Interface (CGI), proprietary web server APIs, scripting language extensions or the java servlet API. Each approach has its specific advantages and disadvantages that will be discussed in the following subsections.

3.1.1.1 The Common Gateway Interface (CGI)

The common gateway interface (CGI) [CGI00] is a standard interface implemented by every web server. It was the first interface that allowed to execute programs on the server and to produce dynamic HTML pages (opposed to static HTML pages read from the file system). Every time a request is received, the CGI program is started, the operations are carried out, the HTML code is produced, and the program exits. All early applications and still a lot of current implementations use this interface, because it is standardized, flexible and easy to use.

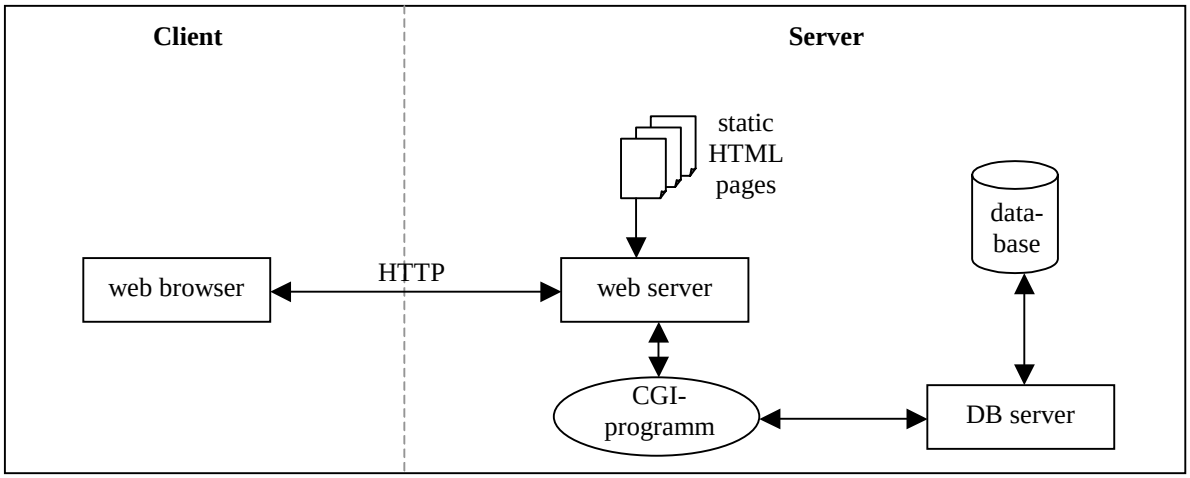

Figure 3-1: CGI based connection

The only condition for a program used with the CGI interface is that the program has to produce HTML code on the standard output, which is then transferred to the client. There are no restrictions regarding the language, any programming or scripting language can be used to write CGI programs. Very popular are shell scripts, perl or the c programming language. Figure 3-1 shows the architecture of a CGI based connection.

The main advantage of CGI is that it is a standardized interface, easy to use and very flexible regarding the programming languages used. Additionally, by running the CGI program as a separate process, the web server is not affected by errors or malfunctions of the program. But this feature is at the same time the main disadvantage of the CGI approach: the CGI program has to be started and the connection to the database has to be opened for each request, which is very inefficient. One solution for this problem is the use of an application server holding one or more connections to the database permanently open. The CGI program is then only used to pass the necessary parameters to the application server. Another possibility is to use FastCGI [Op96], an extension to the CGI standard where the CGI program is not terminated after having served the request, but remains activated in order to answer the next query.

3.1.1.2 Proprietary web server APIs

An alternative to the CGI approach is the use of proprietary application programming interfaces (APIs) offered by most web servers, for example NSAPI [Net00] or ISAPI [Mic00]. These interfaces allow write and link extensions to the web server, which are executed within the server process. Figure 3-2 shows the architecture of such a connection.

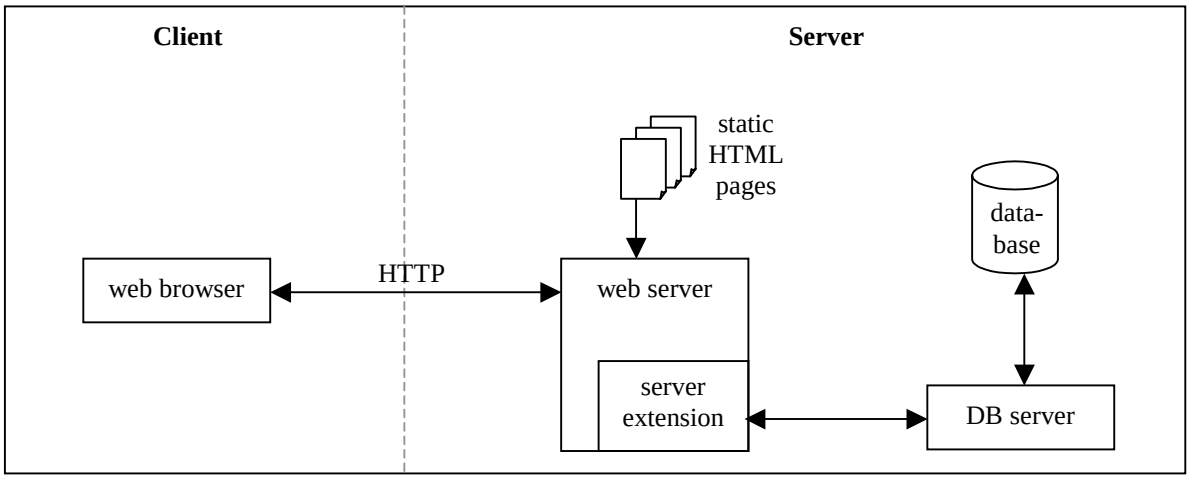

Figure 3-2: web server API connection

The advantages of this approach are a better performance because of less communication and process management overhead, and the possibility of directly accessing and modifying web server functions. The main disadvantages are higher implementation efforts (the code has to be thread-save), possible side effects on the server (for example crashes because of errors and malfunctions), and the commitment to a specific server and platform (whereas CGI programs are at least web server independent).

3.1.1.3 Scripting language extensions

This classification covers a rather huge and heterogeneous group of non-standard interfaces often supplied by third party vendors. Examples are modules allowing the execution of perl [WCO00] or php [RG00] scripts, the Microsoft active server pages (asp) extension [ASP00], the cold fusion toolkit with its cold fusion markup language (CFML) [DM00], or the OMNIS gateway used by former version of the WebCon toolkit [Zol96, ZS98, SZ99]. These extensions are connected to the web server either by CGI or a proprietary server API. They offer some kind of higher level programming interface to the user, often an HTML extension. The following example is a small template for the OMNIS gateway:

```
<HEAD>
<TITLE>sample template</TITLE>
</HEAD>

<BODY>
<H1>List of Projects</H1>
<!-- OMNIS BT -->
<!-- OMNIS SQL select Title, Begin, End from Projects -->
      <P>
      Project &(table[0])   (from &(table[1]) to &(table[2]) )
      </P><BR>
<!-- /OMNIS -->
<!--OMNIS CT -->
</BODY>
```

The standard HTML code is simply passed on by the gateway, only the instructions beginning with <!-- OMNIS … are interpreted and executed. In this example, a transaction is initialized (OMNIS BT) and the title and duration of each project in the database are retrieved and displayed (this happens within the <!-- OMNIS SQL … --> … <!-- /OMNIS -> loop, the &(table[x]) variables correspond to the current tuple's attributes). At the end, the transaction is committed (OMNIS CT).

One advantage of using scripting languages is the fact that these languages (mostly HTML extensions like WebCon and ColdFusion) are easier to use than traditional programming languages and do not have to be compiled. Additionally, the risk of causing server malfunctions is much lower compared to the API approach. The main disadvantages are the proprietary nature of this approach, the reduced functionality of these scripting languages compared to traditional programming languages, and the lower performance compared to compiled code.

3.1.1.4 Java servlet API

The java servlet API is an interface combining most advantages of the other three approaches discussed so far. Servlets are java programs running on server side (opposed to applets, see section 3.1.2) using sun's servlet API [Goo99]. A lot of web servers directly support this API,

for example apache, lotus domino, Netscape enterprise server, or IBM internet connection server. Most other web servers can be extended by servlet engines available for various platforms and vendors, for example WAICoolRunner, JRun, or ServletExec. Depending on the web server and the servlet engine, the servlets can either be executed within the server process or within a separate servlet server.

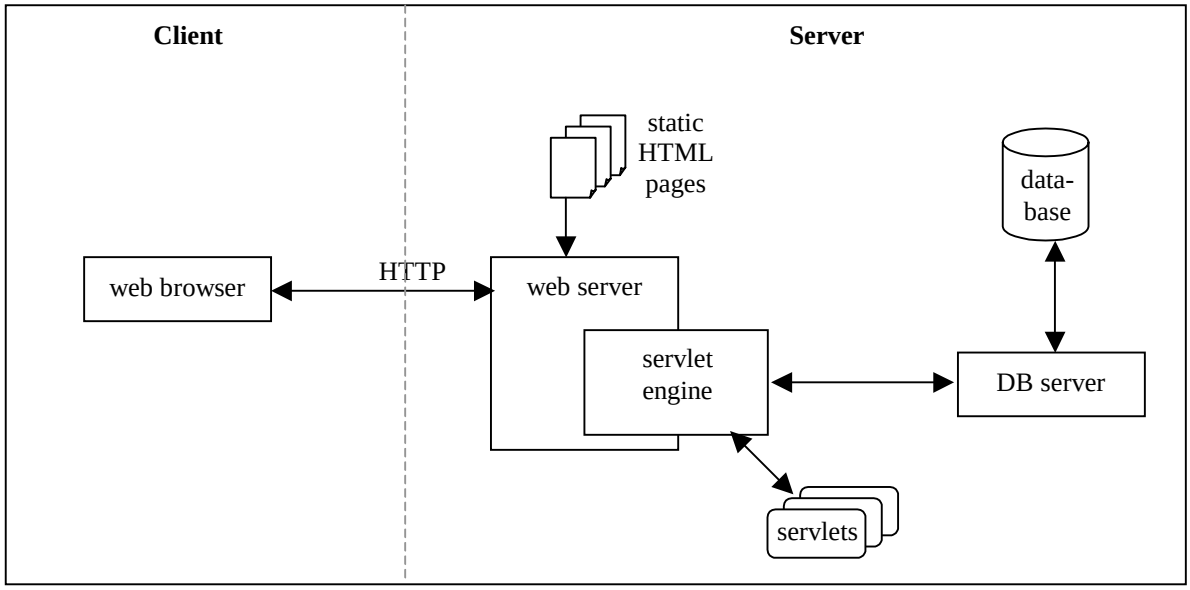

Figure 3-3: servlet API connection

The java servlet API is a standardized interface similar to CGI, but avoids the problem of starting a new process for each request, because servlets are only initialized once. This happens either at web server startup or when the first request is received. Additionally, this approach has the advantage of allowing to use a full featured programming language similar to the server API approach, but avoids the drawback of risking server stability, because servlets are not compiled into the web server. In contrast to scripting language extensions, servlets are not bound to a specific platform or web server, because they use the java programming language and the standardized servlet API.
The main disadvantage of the servlet approach is the lower performance of interpreted code (java) compared to compiled programs.

3.1.2. Connection on Client Side

Connecting a database to the World Wide Web on client side means that the web browser communicates with the database system directly and not by contacting the web server. For this purpose, the browser has to be extended by an additional application component. This concept shows several advantages originating mainly from avoiding the stateless http protocol:

- Easy transaction and session management
  Arbitrary protocols may be used to connect the browser to the database system, for example the proprietary protocol of the database vendor. These protocols are usually stateful and offer already an integrated session and transaction management.
- Information processing on client side
  The browser extension allows performing information processing on the client. This can be useful for checking whether the user's input matches a given specification, for example a valid email address or upper case letters only. It can also be helpful for executing costly (in terms of time) calculations which otherwise had to be done on server side.
- Server load reduction
  Establishing a direct connection between browser and database clearly reduces the web server load, because now the web server only handles requests for static HTML documents and does not have to execute expensive database queries.

On the other hand, shifting application logic to the client side causes new problems:

- Higher development costs
  In order to achieve a high degree of availability, the application has to be implemented and tested for various browser types, browser versions and platforms. Even if standardized techniques (java) are used, the behavior of the application may differ significantly on client side due to bugs and incomplete implementations of standards.
- Proprietary solutions
  Not using HTTP as the transmission protocol often leads to proprietary solutions. On the one hand, this results in huge migration efforts in case of changing the database system used. On the other hand, existing security concepts (like firewalls) have to be adapted in order to treat the new protocol type properly.
- Lower acceptance
  The acceptance of a client side solution is clearly lower than that of a server side connection. The reasons are additional efforts for installing the browser extensions or performance and security objections, because this approach requires some foreign code to be executed on the user's host.

Two techniques for connecting databases to the web on client side are known: Plug-ins and java applets. Plug-ins are extensions which are compiled and linked to the browser using a browser specific API, for example ActiveX from Microsoft. Because these extensions are platform and browser specific and have to be installed on client side, this approach is not very common.
Java applets are small java programs, which are sent from the web server to the browser where they are executed. The special security concept of the java programming language (the *sandbox* model, see for example [Ber99]) ensures that applets are unable to address any resources on client side like hard disk or floppy, unless the applet is *trusted*. Trusted applets are either signed or come from a trusted source, for example the local hard disk. In order to be really platform independent, the applet has to use a pure java driver for connecting to the

database system. For this purpose, most database vendors provide a driver conforming to the Java Database Connectivity Standard (JDBC) [Dic97].

Figure 3-4 shows the architecture of a client based connection using applets.

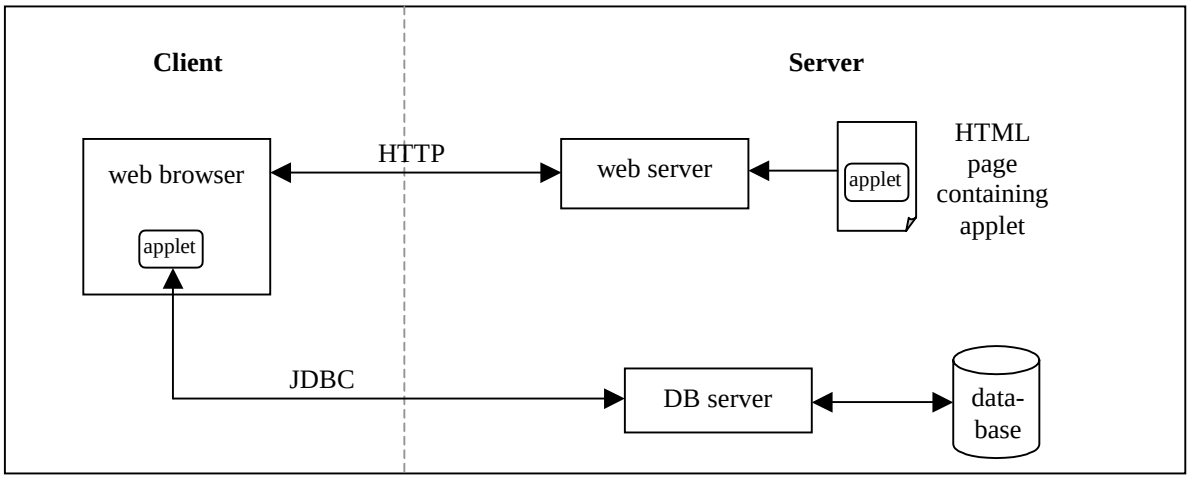

Figure 3-4: client side connection using applets

Applets have the advantage that they do not have to be installed on client side and are browser and platform independent. The disadvantages are higher download times for the HTML pages containing the applets, higher development costs compared to ordinary HTML pages and possible execution errors due to incomplete support of standards by the web browsers.

3.2 Page generation

In addition to the connection type discussed in the previous section, the HTML page generation technique is also a classifying feature of database-driven web applications. When talking about page generation techniques, we do not refer to the programming language and environment used to create the application, but distinguish whether the HTML pages are created at runtime (online) or materialized (offline). Since this distinction refers to the generation of HTML pages, it is only valid for server side database connections.

3.2.1. Online generation (dynamic page generation)

Online generation of HTML pages means to create the documents at runtime when the user sends a request. The application has to contact the database server, execute the corresponding queries, receive the results and convert these into HTML documents. The main advantages of this approach are:

- The dynamically created HTML pages always show the latest information from the database. Therefore, this technique is especially suited for applications where the database

contents change frequently and where it is important for the users to get up to date information. Typical applications are online brokers, auctions or ticket booking centers.

- Online generation of HTML pages also allows offering structured query interfaces directly querying the database instead of executing unstructured full text search on HTML pages. This provides the user with more powerful search interfaces and more efficient query execution especially for large-scale information systems.
- Personalization becomes a more and more important aspect of modern web sites. HTML pages can only be adapted to the user's preferences if they are created at runtime. Materializing all variations of every HTML page of an application is impossible.
- The administration of applications using this technique is very simple, because changes to the database become automatically visible in the web pages at once; the documents do not have to be updated explicitly.

The most important disadvantage of online generation of HTML pages is the lower performance compared to the delivery of static HTML pages, because the database has to be queried for each document requested. However, the performance can be improved significantly by using internal caches for the tuples requested most, the page structure or even for whole HTML pages.

3.2.2. Offline generation (materialization of HTML pages)

Creating HTML pages from database contents and storing the whole pages is called offline generation or materialization of HTML pages. The pages can either be stored in the file system so that they are directly accessible by the web server, or in the database system as a binary large object (blob). From a database-centric point of view, this technique could also be regarded as some kind of page cache. The main benefits of this solution are:

- Higher performance
  The online generation of HTML pages requires a certain amount of database queries for retrieving the application data and algorithms converting the results into HTML pages. Fetching the whole page in one piece needs only one access, which is clearly more efficient. If the page is stored in the local file system, the web server even doesn't have to contact the DBMS.
- More independence from the database system
  Materialized pages are completely independent from the database system. This can be very useful if the database system is offline for some time, for example because the database schema is changed. Another advantage is the ability to store and distribute the application, for example on cdrom, without having to include the database system.
- Higher database security
  The users only address static HTML pages instead of a database gateway. This excludes the possibility of an attack on the database system from the web. Especially the CGI interface is known for providing some risks if the corresponding programs are not implemented carefully.

Besides the lack of support for personalization and advanced search concepts (see 3.2.1), the main disadvantage of this approach is the complex and costly administration of the materialized pages. Small changes in the application data may require the re-generation of numerous pages. As an example, changing the name of a person must not only lead to a new generation of the person's page, but also to a reconstruction of all pages referencing this person, because the corresponding links could contain the person's name. This could be project pages where this person is listed as a participating scientist, department overviews, or bibliographic references.
Always generating the whole application whenever the data source has been changed is a very inefficient solution, especially for large scale web sites. Only generating the pages affected by the change in the data source is the better approach, but requires an efficient algorithm for the detection of the dependencies among the generated pages as proposed in [Som00].

3.3 Authorization

When combining databases and the World Wide Web, the selection of the appropriate authorization concept is another classifying feature of database-driven Web applications. Both database systems and web servers have developed their own access control standards and concepts, which are tailored to the specific requirements and technical possibilities of the corresponding area of operation.
If a client side connection of the DBMS is used, only the authorization mechanism of the database system can be used if the application does not implement its own security concept. Server side approaches have several options for choosing an appropriate authorization concept, which will be discussed in the following subsections.

3.3.1. Pure web server authorization

Web server authorization strategies have been designed for securing sets of HTML pages lying in the local file system. Their application granularity are whole directories, single files cannot be secured separately.
Access control can be applied on two levels: domain level and user level. On the domain level, a group of hosts can be specified which are allowed to access the corresponding HTML directories. For example, the specification **.tu-muenchen.de* accepts requests from all hosts and subdomains within the *tu-muenchen* domain. By providing only hostnames or IP addresses instead of domains, access can even be restricted to single hosts. For example, an access restriction of a directory looks like this (when using an apache web server):

```
<DIRECTORY /home/proj/www/internals>
      <LIMIT GET POST>
      option deny,allow
      allow from *.tu-muenchen.de
      </LIMIT>
</DIRECTORY>
```

The second level of application is the user level. Web servers allow specifying a list of users who are granted access to a specific directory. For this purpose, the web server does not use the login name specified in the http protocol, because this information can easily be manipulated. The corresponding user account has to be created on the web server first, that means the login and the (encrypted) password are stored in a configuration file. If a user tries to access a restricted area, he is prompted for login and password, which is then compared to the entries in the configuration file. For an apache web server, such an access restriction can be specified as follows:

```
<DIRECTORY /home/proj/www/internals>
      <LIMIT GET POST>
      require user bayer, sommer, zoller
      </LIMIT>
</DIRECTORY>
```

Database-driven web applications can use this technique for controlling access to their HTML pages. For HTML pages materialized in the file system (see section 3.2.2), this is the only solution available.
For online generation of web pages, this technique can only be used if the web server directly accesses the local file system in order to generate dynamic HTML pages. In this case, the corresponding file (for example, an extended HTML page or a CGI binary) can be placed within the source tree of the web server and secured with one of the strategies described before. This technique can be used with every CGI and API approach and with most scripting solutions, if the scripts are directly called by the web server and can be stored within its source tree. Authorization happens only within the web server source tree. For connecting the DBMS, a fixed login is used independent of the requesting client.
Figure 3-5 illustrates the architecture of an application with pure web server authorization for a CGI based connection.

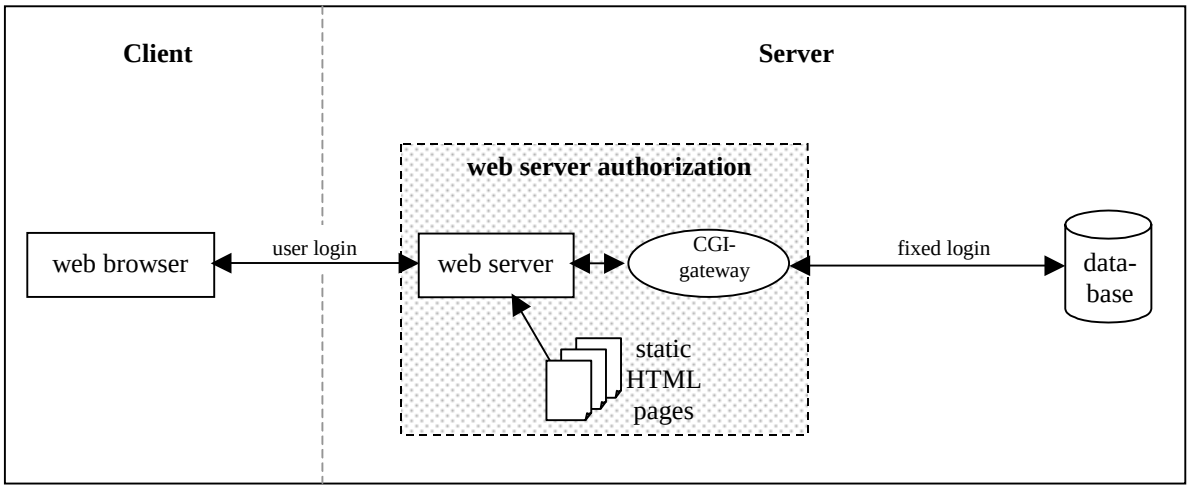

Figure 3-5: pure web server authorization for a CGI based solution

If such a CGI connection is used, this approach has the big disadvantage that only access to the whole application can be restricted unless different CGI binaries in different directories are used. While this may not be a big problem for small applications, it is real drawback for large and complex information systems.
For API and scripting solutions, this approach integrates seamlessly into the security concept used for static HTML pages, but offers only limited functionality (access restrictions on directory level, no different levels of authorization like read/update/write).

3.3.2. Pure DBMS authorization

Instead of using the web server's authorization techniques, a database-driven web application may also leave it to the DBMS to permit or restrict access to the system. When choosing this approach, a database account has to be created for all users who are granted access. The application has to provide a web page where the user can specify login and password, which are then used by the database gateway for connecting the DBMS. Figure 3-6 shows the architecture of an application using pure DBMS authorization.

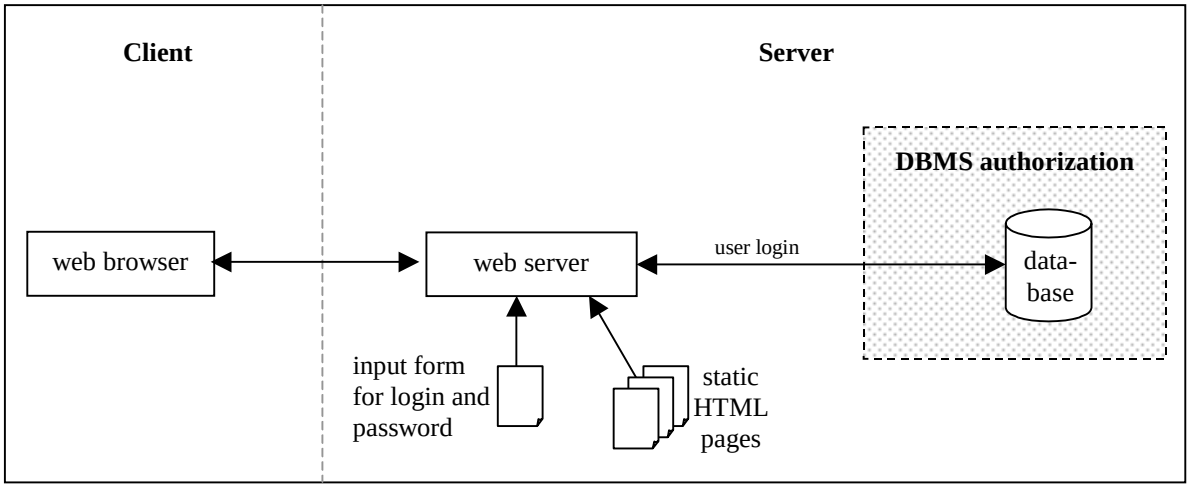

Figure 3-6: application using pure DBMS authorization

The advantage of this solution is the more powerful and flexible authorization concept of the DBMS compared to that of a web server. Access can be granted for single tables instead of whole directories of HTML pages, and different levels of access (for example *read only*, *update* or *delete*) can be used. Additionally, this approach is independent of the connection technique used (CGI, Servlets,).
The main disadvantages arise from the stateless nature of the HTTP protocol. Since the user is not permanently connected to the web server, he had to authenticate himself for every new HTML page requested. To avoid this unacceptable behavior, the application has to implement a complete session management where some unique ID, which is assigned upon the first request, identifies the user. This can be achieved, for example, by using cookies [Net99] or parameters exchanged with each HTML page. If web server authorization would be used instead, web browser and web server handled this issue automatically.

Another drawback is the fact that also an error management has to be implemented by the application, because authorization errors from the DBMS have to be processed and mapped to HTML error responses. And last but not least, the database administrator has to care for authorization issues of the web application, which should be the task of the web administrator.

As a summary, this approach offers a more powerful and flexible authorization concept than pure web server authorization, but requires significantly more efforts for implementing and maintaining a web application due to the stateless nature of the HTTP protocol.

3.3.3. Hybrid approaches

Hybrid approaches combine web server authorization and DBMS authorization. This is typically done by offering a central static HTML page from which dynamically created pages can be reached. If access to this central HTML page is restricted by the web server on user level, each client accessing this page has to provide username and password. This information can then be passed to the gateway to be used for database login as shown in Figure 3-7.

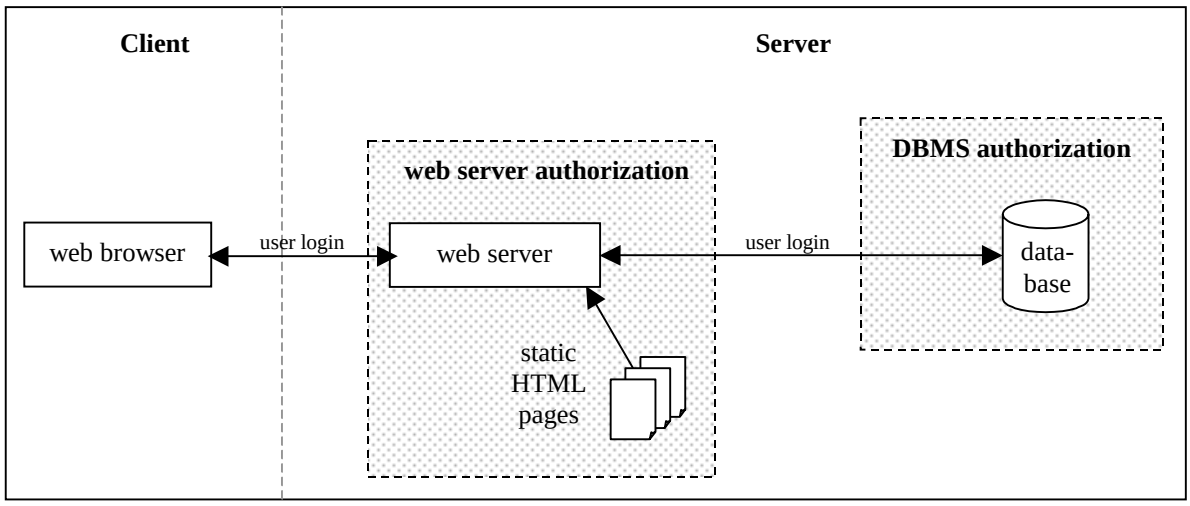

Figure 3-7: application using both web server and DBMS authorization

The advantages of this approach are the easy to use authentication and authorization mechanisms of the web server combined with the higher security level of the more powerful DBMS authorization concept. The major disadvantage lies in the redundant storage and administration of the user accounts for both the web server and the database system, which can lead to inconsistencies and runtime errors.

3.3.4. Middleware

As an alternative to the previous approaches, a middleware can be used to handle authorization issues. This separates authorization aspects both from the user interface (the web server) and the application data (the DBMS) and allows developing an individual authorization strategy tailored to the current application.

In contrast to pure web server authorization, more sophisticated access strategies can be used, but unlike pure DBMS authorization, user maintenance doesn't have to be handled by the DBMS administrator. Compared to hybrid approaches, this solution avoids the redundant storage and maintenance of user accounts. The architecture of an application using middleware for authorization purposes can be seen in Figure 3-8.

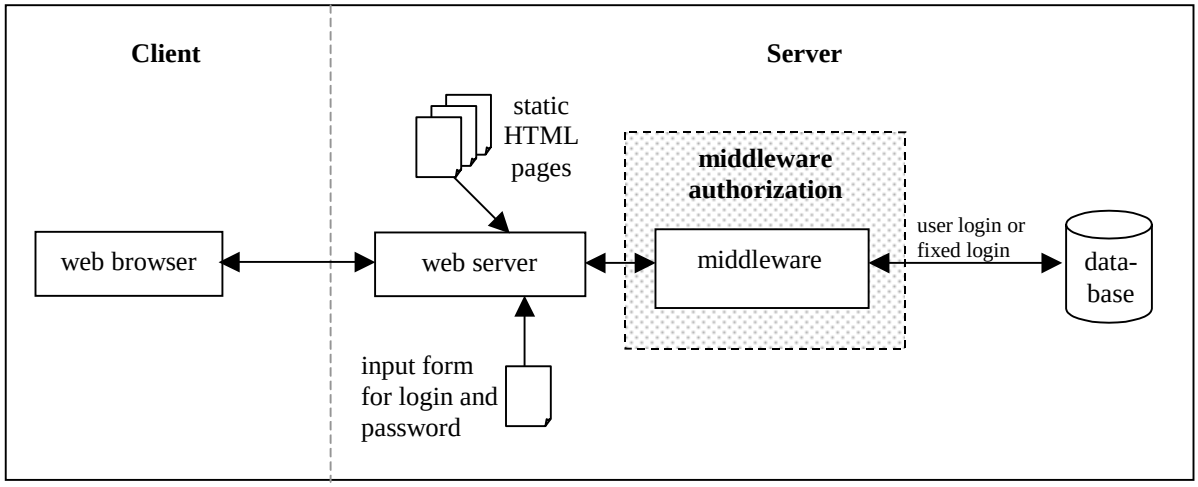

Figure 3-8: architecture of an application using middleware for authorization

As we will see in section 4.4, handling authorization issues by a middleware can be a very advantageous concept especially suited for supporting features like personalization or role based access control, which could not be implemented with reasonable efforts when using one of the other approaches.

The disadvantages of this technique are the higher implementation efforts, because a complete user and session management has to be implemented. But compared to pure DBMS authorization, user and administrator benefit from additional features like role based access control or personalization.

3.4 Commercial Products

A large variety of commercial products and freeware solutions for connecting databases to the web is available today. Besides products especially developed for that purpose, a number of already existing tools and languages has been extended for offering database access from the web, mostly using standardized interfaces like ODBC and JDBC. The majority of these tools cannot be classified according to the characteristics mentioned in this chapter, because they are programming languages that can be used to implement different concepts depending on the current problem and implementation.
We therefore discuss only products and tools explicitly developed to support the connection of databases to the World Wide Web. Two large groups can be distinguished, which are described in the following subsections: tools from database vendors and third party solutions.

3.4.1. Database vendors

In addition to the numerous third party solutions for connecting databases to the web, each database vendor meanwhile offers its own web package. This mostly contains not only the connection component, but also often a proprietary web server and an integrated development environment (IDE). Access to the database system is usually handled by an application server using the vendor specific communication protocol or the java database connectivity (JDBC).
After a short survey on the web components of Sybase, Informix and IBM, we will take a closer look on the web solution of Oracle.

PowerBuilder [Syb00] from Sybase and the Web DataBlade Module [Inf00] from Informix are development platforms that can be used together with arbitrary web servers, because they support the CGI interface. However, server specific connection modules using NSAPI, ISAPI or the apache interface are provided and recommended in order to increase system performance. The Web Sphere [IBM00] development platform of IBM provides a proprietary web server, which is based on the freely available apache web server [Apa00]. This product is used to connect DB2 databases to the web, other web servers are not supported directly. All vendors offer additional application servers for tasks like electronic payment, load balancing or integration of other systems using standard interfaces like CORBA or XML.

The Internet Application Server of Oracle [Ora00a] is an environment for creating multi-tier internet applications using the Oracle8 DBMS. Its architecture is divided into 5 categories as shown in Figure 3-9.

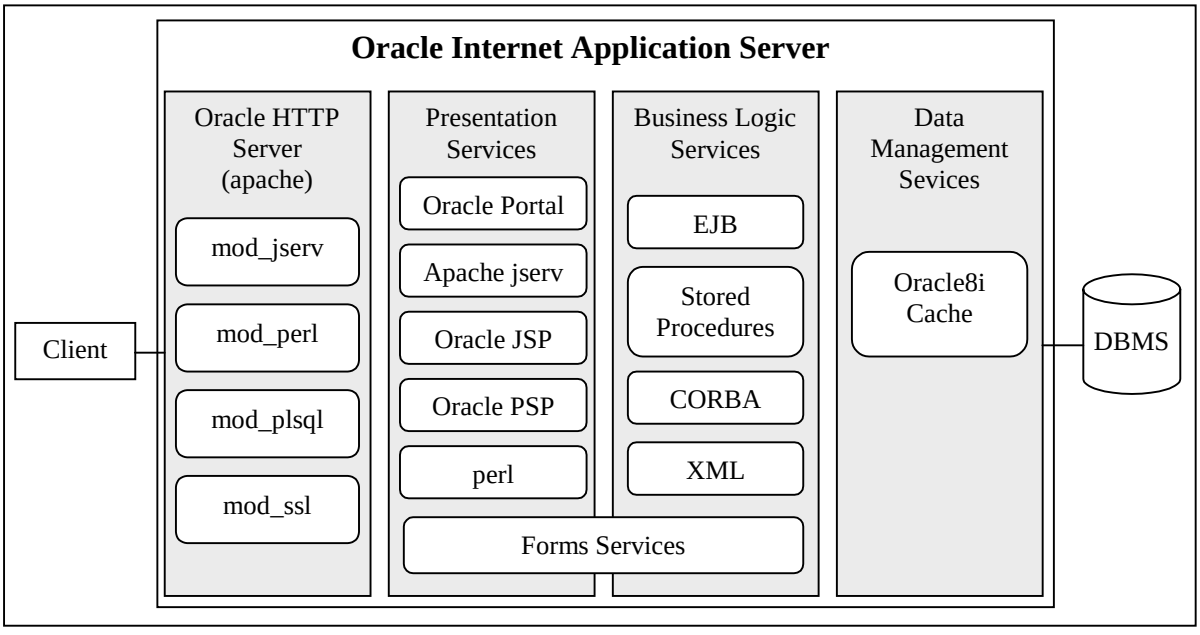

Figure 3-9: architecture of the Oracle Internet Application Server

The communication with the client is handled by the Oracle HTTP server, which is an extended apache web server. Several techniques for creating HTML pages can be used: the jserv module (mod_jserv) allows executing java servlets by using the apache jserv servlet

engine, the perl module (mod_perl) provides support for the perl programming language, and the PL/SQL module (mod_plsql) calls stored procedures on an Oracle8i engine. These can be either PL/SQL stored procedures (PSP) or java stored procedures (JSP). The SSL module (mod_ssl) provides standard HTTPS support enabling secure connections between client and server by using an encryption mechanism over a Secure Sockets Layer (SSL). Besides perl, PSP, JSP and servlets, the Oracle Portal is an additional presentation interface for connecting to the higher level Oracle portal services, which allow integrating other sources and applications. The Business Logic Services provide interfaces for using enterprise java beans (EJB), CORBA services or XML data exchange, while the Data Management Services provide special caching functionality. Additionally, the Oracle Internet Application Server contains a number of development kits (for example the Portal Developer's Kit) not shown in Figure 3-9.

While all the modules and services discussed so far support a server side connection of Oracle8 databases, the Forms Services [Ora00b] allow client side connections using java applets. These java applets connect to the Forms Server consisting of the Forms Listener, Runtime Engine and the optional Forms CGI. The Forms Listener handles the communication with the applet and maintains a pool of Runtime Engines, which connect to the DBMS and execute the client's queries. The Forms CGI allows load balancing by maintaining a "server farm", that means a pool of middle tier machines each running an Oracle Web Server and a Forms Server as shown in Figure 3-10. If a client requests an applet (1), the load balancing server determines the machine with the least load (2) and sends an HTML page back to the client (3) that retrieves the applet from the corresponding server (4,5) (that with the least load). The rest of the communication (6-9) will happen between the client and this machine only.

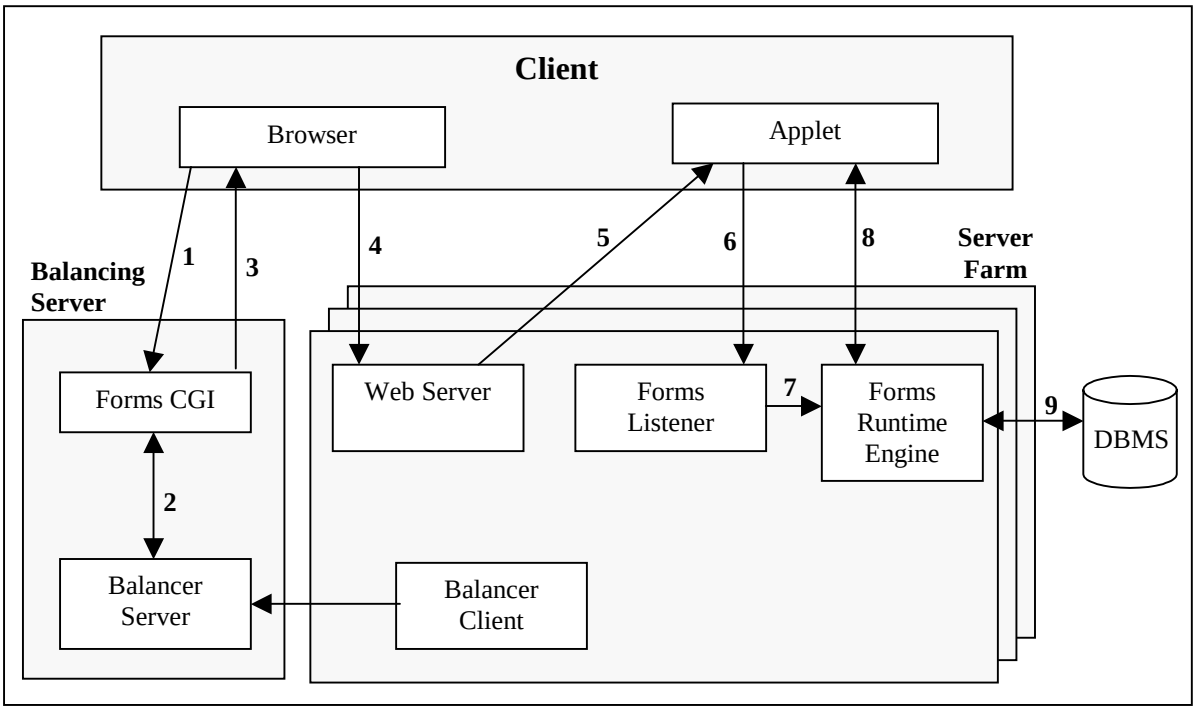

Figure 3-10: load-balancing scenario

3.4.2. Third party tools

During the last years, the family of products for web database integration has grown steadily and reached a size that makes it hard to keep an overview. Especially the availability of standardized database interfaces like ODBC (Open DataBase Connectivity) and JDBC (Java DataBase Connectivity) has promoted the development of numerous tools, languages and development environments.

Traditional programming languages like C++ or perl and new languages like java or php can be used for accessing databases either by using their native or some standardized interface like ODBC and JDBC. These languages help to build individual applications with a variety of different connection architectures like CGI, servlets or proprietary web server API. Although very flexible, we consider these languages to be rather low level programming interfaces than web database integration tools.

The major part of the integration tools available today is the group of HTML extensions, which offer what is often called dynamic HTML (DHTML). These products provide a set of proprietary tags that are used together with standard HTML, the corresponding files are called HTML templates. Instead of requesting these templates directly from the web server, they are processed by a parser that reads the proprietary tags, executes the corresponding commands (for example querying the DBMS) and substitutes the tags with standard HTML. Often integrated development environments of different functionality are part of these commercial products.
Besides a large variety of less known products like WebBase [Exp00], ODBiC [Odb00] or HotSQL [Chi00], the most prominent examples of this technique are Microsoft's active server pages (ASP) [ASP00] with the FrontPage development environment, and Allaire's Cold Fusion platform [All99]. As an example, we will discuss the latter in more detail.

An application built with ColdFusion relies on five components as shown in Figure 3-11: ColdFusion Studio, ColdFusion application pages, ColdFusion server, ColdFusion administrator, and external ODBC data sources.

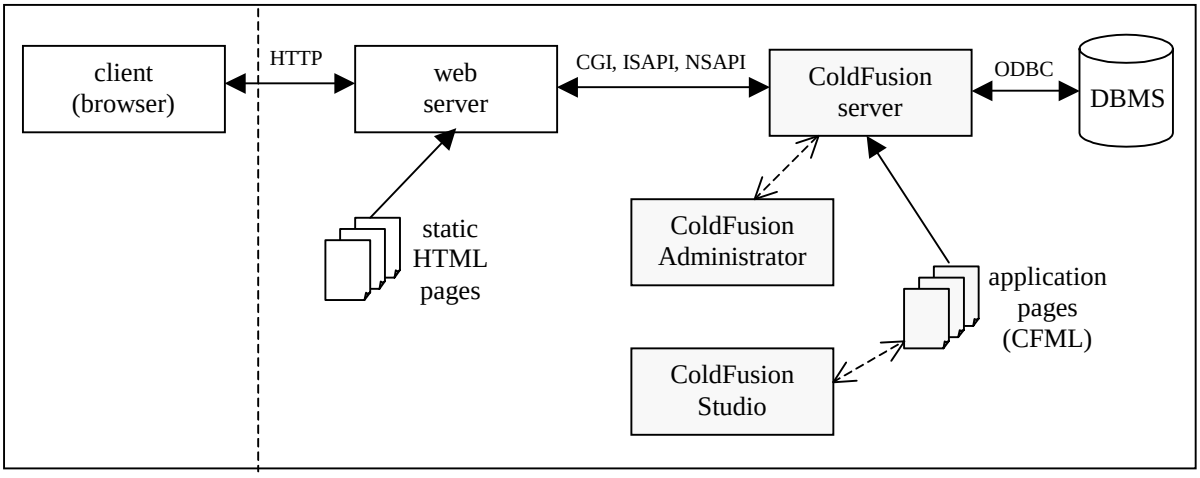

Figure 3-11: architecture of an application built with ColdFusion

The ColdFusion application pages are HTML templates and can be considered as the functional parts of the application. Besides ordinary HTML and other standard client side elements (for example java script), these application pages contain proprietary tags from the ColdFusion Markup Language (CFML). This language provides more than 70 tags and 200 functions for database interaction, variable declaration, conditional expressions and other program structures. The following sample template retrieves the names of all employees from the database table *emp* and displays them within an HTML table:

```
<HTML>
<HEAD>
<TITLE>Employee list</TITLE>
</HEAD>
<BODY>
<H1>List of employees<H1>
<CFQUERY NAME=”EmpList” DATASOURCE=”intranetdb”>
      SELECT FirstName, LastName FROM emp
</CFQUERY>
<TABLE>
<CFOUTPUT QUERY=”EmpList”>
      <TR><TD>#FirstName#</TD><TD>#LastName#</TD></TR>
</CFOUTPUT>
</TABLE>
</BODY>
</HTML>
```

Each tag of the ColdFusion Markup Language begins with CF. The database query is defined between the <CFQUERY> and </CFQUERY> tags, and the results of the query are displayed in the corresponding <CFOUTPUT> … </CFOUTPUT> section. This output section is called in a loop as often as result tuples are available.

The ColdFusion server listens for requests from the web server, reads and executes the corresponding application pages. It can be configured by the ColdFusion Administrator regarding available data sources, debugging, application security, general server settings, server clustering, page execution scheduling or directory mapping.
ColdFusion Studio is the development environment allowing visual development, interactive debugging, and dynamic page previews for application pages. Additionally, it provides support for project management and source control.

Besides this huge group of HTML extensions, a smaller class of commercial products are large scale integration environments like NetDynamics [Sun00] from Sun Microsystems or Total-e-Server (former Saphire/Web) [Blu00] from Bluestone Inc., which offer a set of interfaces and services for integrating applications at enterprise level. The functionality of

these tools can be compared to the complex development environments of database vendors as described in section 3.4.1.

3.5 Summary

This chapter discussed various aspects regarding the web database integration. First, we described the different architectures used for connecting databases to the World Wide Web. The common approach is to create a server side connection using either the standardized Common Gateway Interface (CGI), a proprietary web server API (for example NSAPI), a scripting language extension for the web server, or the java servlet API. The main advantages of a server side connection are the absence of any browser restrictions, lower development efforts and centralized maintenance of the application. Client side connections can be realized by using browser plugins or java applets, which have the advantage of offering a more powerful user interface and the ability of information processing on the client.

Another classifying feature of web database connections is the page generation technique. If online generation (dynamic generation) of HTML pages is used, the documents are created at runtime when a user request is received. This guarantees up to date information and easy maintenance, and allows direct searching within the database and personalization of web pages. Materialization of hypermedia views (offline generation) creates the documents and stores them as HTML files in the web server's file system. This increases performance and security concerning the DBMS, and allows copying the application (for example on cdrom).

The third section of this chapter investigated various authorization concepts for a web database integration. Pure web server authorization is easy to use but provides only limited functionality, whereas pure DBMS authorization suffers from the stateless nature of the HTTP protocol and thus requires additional implementation efforts. Combining both concepts leads to higher maintenance efforts and might cause inconsistencies. Using middleware to handle authorization also requires additional implementation efforts, but enables advanced features like personalization or individual access control concepts.

Finally, we gave a short survey on commercial products for web database integration. One group of products are application development environments offered by the database vendors. As an example, the Oracle8 Internet Application Server has been described. As an alternative, numerous third party tools are available, which are mostly based on HTML language extensions. From this group of products, ColdFusion from Allaire has been presented as an example.

Switching back from the technical aspects of database-driven web applications to the design of general hypermedia applications, the next chapter will introduce the Hypermedia Modeling Technique (HMT).

CHAPTER 4

THE HYPERMEDIA MODELING TECHNIQUE (HMT)

The Hypermedia Modeling Technique (HMT) has been developed in order to provide a solid, intuitive and flexible methodology for database-driven hypermedia design. It is partly based on concepts from the *Relationship Management Methodology* (RMM, see section 2.3.2), but avoids the major drawbacks of RMM and offers several improvements, extensions and completely new modeling concepts.
HMT is the first modeling technique covering information presentation, query interfaces, data manipulation interfaces, authorization design, personalization and temporal design altogether. Additionally, HMT is not bound to a specific hypermedia platform; it can be used to design applications in various hypermedia formats like, for example, HTML [Htm00], XML [Xml00] WML [Ris00] or PDF [AS00].
The following sections describe in detail the different design steps of HMT. In order to provide expressive examples in a familiar environment, we will use a sample www application for illustration purposes.

4.1 Design Process and Overview

The HMT design process consists of a sequence of 6 steps, where each step corresponds to a different abstraction level of design specification. In practice, several iterations of the whole process or of parts of it are necessary in order to create and maintain a hypermedia application. This iterative approach can only succeed if the different design steps form a clearly separated hierarchy where the different layers can be accessed directly and changes to later steps do not influence earlier ones. For example, it must be possible to change the presentation of an element without having to change the structure or definition of the document containing this object. But, of course, changes to one design step are often influencing the following ones.

The HMT design process (see Figure 4-1) is divided into six different steps:

Step 1: Requirements analysis

This is always the first task that has to be accomplished when designing IT-systems. Requirements analysis covers aspects like the definition of the application domain, identification of the intended users, and specification of the system’s functionality and usage.

Although an interesting and important aspect, this topic is not discussed within this work. Numerous publications can give further information on this issue, for example [Wei82, KK92].

Step 2: ER Design

After the application domain has been specified by the requirements analysis, an ER schema has to be built reflecting its objects and relationships, for example *projects*, *employees* and *employee_works_in_project*. Since HMT relies only on the basic ER model as described by Chen [Che76], any extended ER model like, for example, [SSW80, TYF86, Teo94] can also be used for this design step.
If a hypermedia application has to be built upon an already existing database, these first two steps in the HMT design process are omitted.

Step 3: Conceptual Hypermedia Design

The core hypermedia application design starts with the conceptual hypermedia design of the application, based on the ER schema developed in the previous step. Aspects like information clustering within the documents (which attributes in which document) and navigation (which links in which documents) are addressed within this design step.

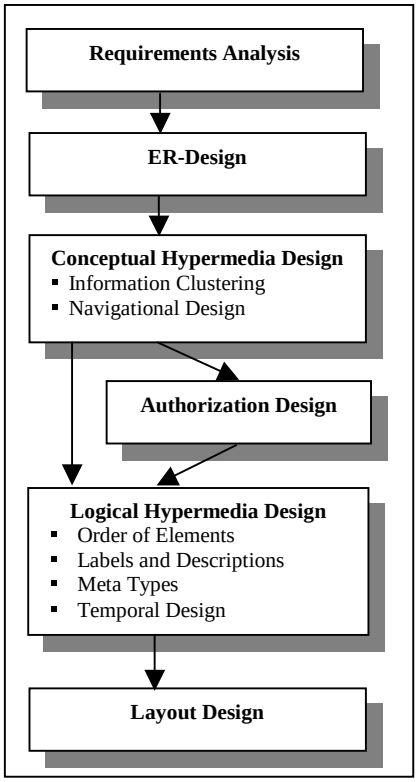

Figure 4-1: the HMT design process

Step 4: Authorization Design

Access restrictions are specified during the authorization design phase using RBAC-techniques (Role Based Access Control, see for example [LS97, SCF+96]). Based upon the conceptual hypermedia schema built during the previous design step, access to certain parts of the application can be limited by specifying roles required for viewing these components. For example, access to private information about projects like internal project reports or implementation details might be restricted to users with the roles *project_partner* or *project_leader*. If applied on components of documents (instead of whole documents), personalized hypermedia applications can be created.

Step 5: Logical Hypermedia Design

During the logical design phase, additional properties concerning the logical representation of a document's content are defined. This mainly refers to the following aspects:

- The sequence of elements within the document
  Each document can be described as a sequence of elements, where each element can be assigned a unique position.

- The temporal order of elements within the document
  Each element of a document can have a temporal specification including the absolute time of visibility (duration), the point of time it starts being visible, or one or more synchronizing elements.
- The labels or descriptions of the elements
  It is often not sufficient to just display the contents of an attribute; some kind of label has to be attached describing the information displayed.
- The meta types of the attributes
  Attribute values or input forms for attributes can be displayed in various ways, for example as plain text, popup menus or button lists. HMT knows various sets of meta types depending on both the database type of the attribute and the kind of document used.

Step 6: Layout Design

Finally, all aspects regarding the layout of the later presentation are covered by the layout design step. This step heavily depends on the hypermedia system used for building a presentation, although certain aspects are common to most systems: The presentation layout can be described by defining a background color or image, font type (plus size and color), standard headers and footers, or default resolution, sound volume, or frame rate.

The following sections will take a closer look at the design steps 2 through 6.

4.2 ER Design

After having finished the requirements analysis (step 1 of the HMT design process), an ER schema of the application domain has to be created. Since HMT relies only on the basic ER model as described by Chen [Che76], any extended ER model like, for example, [SSW80, TYF86, Teo94] can also be used for this design step. The formal specification of the ER model we use is based on [Tha91] and [Vos91].

A basic ER schema contains two types of elements: entities and relationships. **Entity Types** represent sets of real world objects like *employees*, *departments* or *projects*. A single object is called an **entity** and can be regarded as an instance of an entity type. Each entity type provides a set of **attributes**, which are used to store the values of the entities, for example *name* or *title*. A subset of these attributes is used as a unique identifier (called **key**) to distinguish the different entities.

Definition 4.1 (entity type):

> An **entity type** is a pair $e = (ATTR_e, KEY_e)$ with e being the name of the entity type, $ATTR_e$ being the set of **attributes** and $KEY_e \subseteq ATTR_e$ being the **key** of the entity type.

Relationship types are used to model the interdependencies between two or more entity types, for example *employee_works_in_department* or *employee_participates_in_project*.. Instances of a relationship type are referred to as **relationships**. Similar to entity types,

relationship types can have a set of **attributes** carrying the objects information, for example *salary*. In HMT, relationship types are considered to be without direction.

Definition 4.2 (relationship type):

> Let E be a set of entity types, then a **relationship type** is a pair $r = (ENT_r, ATTR_r)$ with r being the name of the relationship type, $ENT_r \subseteq E$ being a multiset of entity types with $| ENT_r | \geq 2$, and $ATTR_r$ being a (possibly empty) set of **attributes**.

Since this basic formal description of an ER schema is sufficient for describing the next steps of the HMT design process, we do not investigate on advanced issues like arity and cardinality of relationships or features of entity keys. Therefore an entity-relationship schema can be defined as follows:

Definition 4.3 (entity relationship schema):

> An **entity-relationship schema** is a pair $s=(E,R)$, where E is a set of entity types and R is a set of relationship types.

Figure 4-2 shows the ER schema of a small sample scenario, which will be used throughout the remainder of this chapter. We use the standard ER notation [Che76] with rectangles for entity types, ellipses for attributes, an rhombs for relationship types. The cardinality of a relationship is specified next to the corresponding entity types.

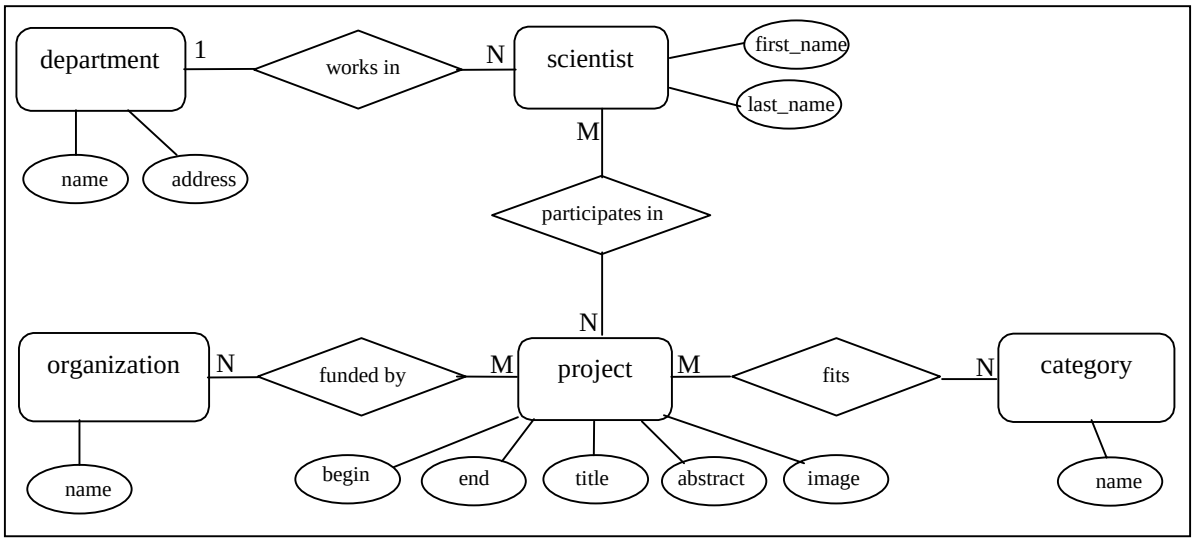

Figure 4-2: ER schema of the sample scenario

The sample scenario contains information about research projects, scientists working in these projects, departments, funding organizations and categories classifying the projects, for example *multimedia*, *electronics* or *agriculture*. Each project can have an arbitrary number (n:m relationships) of funding organizations, participating scientists and relevant categories. A scientist works for only one department, but a department can have an arbitrary number of scientists (n:1 relationship). This scenario is a simplified part of the ER schema of the online information system *abayfor-online* [Ab00], which has been built using a former version of HMT.

Having developed an ER schema of the application domain, the next step in the HMT design process is to create a conceptual hypermedia schema.

4.3 Conceptual Hypermedia Design

The HMT conceptual design of a hypermedia application focuses on two aspects:

Information Clustering is the process of grouping various portions of information into one place. Although the hypermedia application to be developed is based on an ER schema, not all attributes of an entity type necessarily have to be presented together on one page. Some attributes (for example numeric keys or private information) may even have to be hidden from the user. The HMT primitives dealing with information clustering are called **domain primitives**.

Navigational Design determines the access structures which are used to navigate through the application by following the corresponding hyperlinks. Mostly these hyperlinks are based on relationship types from the ER schema. For example, a project document can contain hyperlinks pointing to the participating scientists, which have been calculated using the *participates_in* relationship type. In some cases, however, hyperlinks may also originate from the structure of the hypermedia application, for example if the information about projects is too long to fit in one document and therefore is distributed between several documents which are connected by links. In HMT, so called **access primitives** support the navigational design.

Together, information clustering and navigational design build the HMT conceptual hypermedia design step. The result of this design step is a (conceptual) HMT schema, which consists of a set of documents together with various substructures and a set of access relationships describing the navigational structures between the documents.

Documents are the central element of the HMT domain primitives, which will be described in the following subsection, access structures are discussed in section 4.3.2.

4.3.1. Basic Domain Primitives

For information clustering, HMT offers a set of so called **domain primitives** (see Figure 4-3). The unit of presentation in HMT is a **document**, which can be defined using the HMT **document type**. A document type is identified by a UID (unique identifier, often a meaningful string like “project_overview”) and can be regarded as the frame of a presentation. The graphical representation of a document type is a rectangle with a folded edge, the document type’s UID is located at the bottom of the symbol. Two different main classes of document types can be distinguished:

Document types which are related to a certain entity type or relationship type are called **ER document types**, the corresponding entity type or relationship type is called the document type's **base ER type**. Each ER document type has exactly one base ER type. For example, an ER document type named *project_overview* with base ER type *projects* can be defined. It will be used to generate project documents presenting an overview of a given project containing its title, abstract and the contact person. In other words, ER document types can be considered as templates providing the description for the generation of a set of documents based on the ER type's entity or relationship.

Assigning more than one base ER type to a document is not allowed, because this has no clear semantics: What would it mean if a document would have two base ER types *projects* and *scientists* and set of attributes from both ER types; which scientist entities had to be combined with which project entities? If all scientists of a given project are to be identified, this can be managed using the HMT access struct primitive (see section 4.3.2). Thus each ER document presents exactly one entity or relationship from its base ER type. We deliberately allow to use both entity types and relationship types at this point, because the lack of a precise rule for deciding whether to use a relationship type or an entity type for modeling a given object of the application domain is known to be one of the weak points of the ER model:

> *"The major problem in the entity-relationship approach is that one person's entity is another person's relationship" [Cod90]*.

In addition to these ER documents presenting information contained in the ER schema, there is also a need for documents presenting some general information not coming from the application domain. For example, there could be some kind of options document allowing the user to choose a language and other preferences, or a help document presenting information on how to use the hypermedia system. These kinds of documents can be specified using **general documents types**. Since they do not rely on ER types, each general document type specifies exactly one general document, that means the terms *general document type* and *general document* could be used simultaneously. However, in order to be as precise and consistent as possible, we will continue to use the term *general document type* when talking about the HMT conceptual design primitive, and the term *general document* when talking about the instance of a general document type. When simply talking about *document types*, both ER document types and general document types are meant.

Each document type can contain a set of arbitrary, unspecified contents called **adds**. These adds can be considered as black boxes, their contents are not interpreted by HMT at any time. Adds may be used, for example, to define a standard disclaimer with copyright and webmaster address, which is included as a footer at the end of each document. Adds may also be used to insert additional layout information into the document (for example a piece of HTML code), but doing this is not recommended, because then the document is bound to be used on a specific hypermedia platform and the advantages of a clear distinction between conceptual, logical and layout design are lost. The graphical representation of an add is a rhomb with a unique identifier inside.

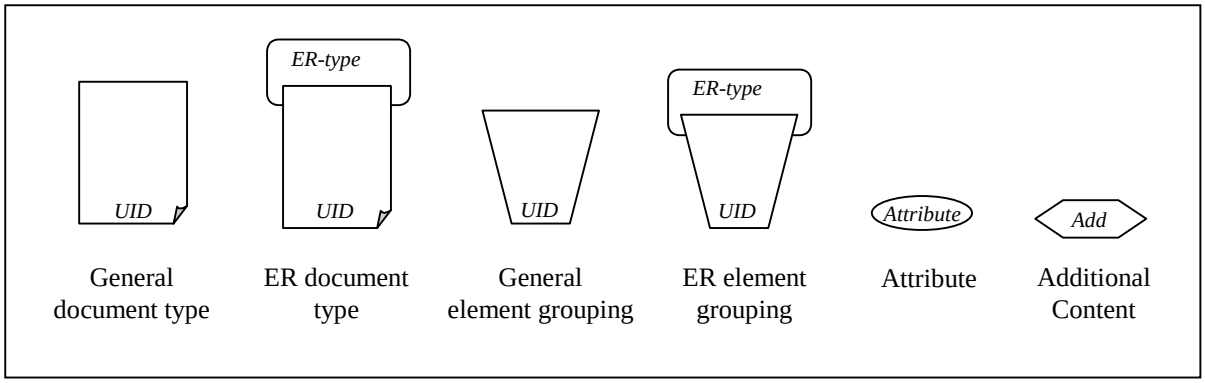

Figure 4-3: basic domain primitives

In contrast to general document types, ER document types may additionally contain a (possibly empty) set of attributes of the corresponding base entity type. The ER documents will then present the values of these attributes. For example, the project_overview ER document type contains the attributes *title*, *abstract*, and *image* as shown in Figure 4-7.

In all document types, the document type's elements (that means adds, attributes or access primitives) can be grouped into so-called **element-groupings**. These groupings are represented by a trapezoid and have a unique identifier (UID). Similar to document types, they can be divided into **ER element-groupings** with and **general element-groupings** without a related entity or relationship type. For example, the address of a scientist may be modeled as an element-grouping with UID *scientist_address* containing the attributes *street, city*, *zip-code*, and *country*. Since element-groupings are treated as individual components, they may be used by different document types or even other element-groupings, if these are of the same kind (i.e. have the same or no base ER type). This allows building nested element-grouping structures, whereas document types cannot be nested. Figure 4-4 shows a sample scientist_info document type which is modeled both with (variant 1) and without (variant 2) the use of an element-grouping.

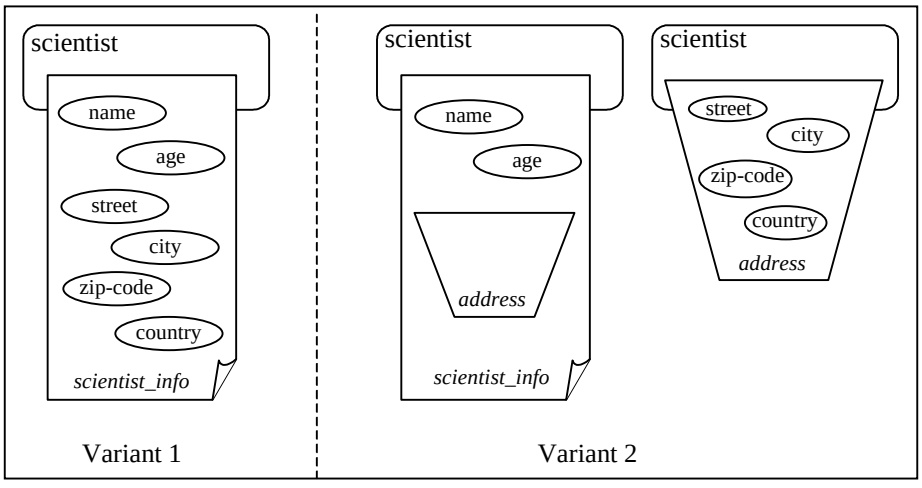

Figure 4-4: HMT document type modeled with and without the use of element-groupings

Although document types and element-groupings embody similar concepts, their distinction is essential for the expressive power and high consistency of HMT.

- First, element-groupings can be used to define information units that can be reused by other document types or element-groupings. This kind of normalization enforces consistency throughout the application by avoiding redundancies, saves time when developing the application and eases maintenance. In our sample scenario, the scientist element-grouping *address* containing the attributes *street*, *city*, *zip-code* and *country* can be subsequently used by different document types like the scientists home document type, the projects document type (for the list of participating scientists) or the departments document type (for the list of employees). A change to this element grouping would automatically affect all document types (or element-groupings) it belongs to.
- Second, another important advantage of building a document type from multiple element-groupings is the ability to create adaptive documents with this technique. As we will see in section 4.4, roles can be assigned to element-groupings, for example the role *manager* or *bookkeeper*. The document type for the entity type *projects* can then be built consisting of several element-groupings with different roles assigned to them, for example a *financial details* element-grouping or an *internals* element-grouping. Depending on the role of the user requesting such a document, different contents will be displayed, although only one document type for projects has to be defined.
- Finally, element-groupings are necessary for several other HMT primitives like links or certain access structures as described in the next subsections. Additionally, there are also some special features like the background color of a document or the number of items displayed when using a search interface (see query and result documents, section 4.3.3), which apply to documents but cannot be assigned to element-groupings.

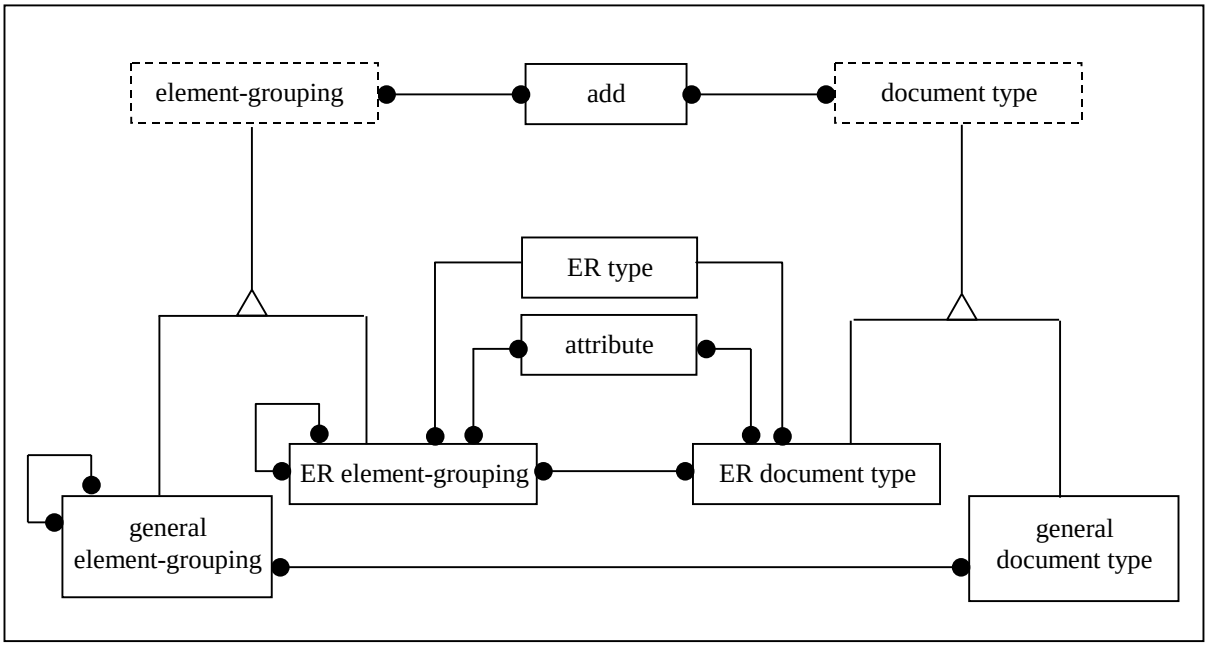

Figure 4-5: type hierarchy of the basic domain primitives

As a short summary, Figure 4-5 shows the OMT [RBP+91] schema of the type hierarchy of the HMT domain primitives and their dependencies, virtual primitives (that means primitives only used for specification purposes) are drawn with a dotted line.

Element-groupings and document types may contain an arbitrary number of adds. In contrast to element-groupings, document types cannot be nested. As a specialization of document types and element-groupings, ER document types and general document types together with ER element-groupings and general element-groupings have been introduced. Each ER document type or element-grouping references exactly one ER type and may contain attributes of this ER type.

Formally, the basic domain primitives of HMT are defined as follows:

General element groupings are defined as recursive structures with a unique identifier and an arbitrary number of adds:

Definition 4.4 (general element-grouping):

Let *ADDS* be the set of HMT adds for a given application and *uid* be a unique identifier, then the **set of general element groupings** $GR^{general}$ is defined as follows:

1) $g=(uid, ADDS_g, \emptyset) \in GR^{general}$ with $ADDS_g \subseteq (ADDS \cup \emptyset)$
2) $g=(uid, ADDS_g, SUB_g) \in GR^{general}$ with $ADDS_g \subseteq (ADDS \cup \emptyset)$ and $SUB_g \subseteq (GR^{general} / g)$

ER element-groupings are similar to general element-groupings, but additionally reference an ER type and a (possibly empty) set of attributes of this ER type:

Definition 4.5 (ER element-grouping):

Let *ADDS* be the set of HMT adds for a given application, *uid* a unique identifier and $e \in E \cup R$ an ER type, then the **set of ER element groupings** GR^{ER} is defined as follows:

1) $g=(uid, ADDS_g, e, ATTR_g, \emptyset) \in GR^{ER}$ with $ADDS_g \subseteq (ADDS \cup \emptyset)$ and $ATTR_g \subseteq (ATTR_e \cup \emptyset)$
2) $g=(uid, ADDS_g, e, ATTR_g, SUB_g) \in GR^{ER}$ with $ADDS_g \subseteq (ADDS \cup \emptyset)$, $ATTR_g \subseteq (ATTR_e \cup \emptyset)$ and $SUB_g \subseteq (GR^{ER} / g)$

e is called the **base ER type** of the element-grouping, and GR_e denotes the set of ER element-groupings with base ER type *e*.

If we do not need to distinguish between ER element-groupings and general element-groupings, we simply speak of element-groupings:

Definition 4.6 (element-grouping):

The **set of element-groupings** is defined as $GR := GR^{general} \cup GR^{ER}$.

As specified before, no grouping can directly be referenced as one of its own subgroupings, but we additionally have to ensure that no cycles appear in a nested grouping structure,

because this would prevent the page generation algorithm (see Figure 5-12) from terminating. We therefore introduce the relation subgrouping:

Definition 4.7 (subgrouping relation):

The relation **subgrouping** is defined as $\{ (s,g) \mid s,g \in GR,\ s \neq g \text{ and } s \in SUB_g \}$ and reads *s is subgrouping of g*. **subgrouping+** denotes the transitive closure of subgrouping,and $SUB^+(g)$ denotes the **set of transitive subgroupings** of *g*:
SUB+(g) := { s ∈ GR | (s, g) ∈ subgrouping+ }

For consistent HMT schemas, the transitive closure subgrouping+ has to be irreflexive, that means $\forall g \in GR: g \notin SUB^+(g)$.

Now document types can be defined. General HMT document types consist of a unique identifier, a set of adds and a set of general element-groupings:

Definition 4.8 (general document type):

Let *ADDS* be the set of HMT adds for a given application and *uid* be a unique identifier, then a **general document type** is a tuple $d = (uid, ADDS_d, GRP_d)$ with $ADDS_g \subseteq (ADDS \cup \varnothing)$ and $GRP_d \subseteq (GR^{general} \cup \varnothing)$ } being a set of general element-groupings.
$DOC^{general}$ denotes the **set of general document types** of a given application.

ER document types additionally include an ER type and a (possibly empty) set of attributes of this ER type:

Definition 4.9 (ER document type):

Let *ADDS* be the set of HMT adds for a given application, *uid* a unique identifier and $e \in E \cup R$ an ER type, then an **ER document type** is a tuple $d=(uid, ADDS_d, e, ATTR_d, GRP_d)$ with $ADDS_g \subseteq (ADDS \cup \varnothing)$, $ATTR_d \subseteq (ATTR_e \cup \varnothing)$ and $GRP_d \subseteq (GR^{ER} \cup \varnothing)$ } being a set of ER element-groupings. DOC^{ER} denotes the **set of ER document types** of a given application, *e* is called the **base ER type** of document *d*. DOC_e denotes the set of ER document types with base ER type *e*.

Speaking of document types, we mean both ER and general document types:

Definition 4.10 (document type) :

The **set of document types** is defined as $DOC := DOC^{general} \cup DOC^{ER}$.

The relation subgrouping can be extended on document types:

Definition 4.11 (extended subgrouping relation):

The relation **subgrouping(p)** is defined as $\{ (s, p) \mid s \in GR,\ p \in GR \cup DOC,\ s \neq p \text{ and } s \in SUB_p \}$ and reads *s is subgrouping of p*. subgrouping+ denotes the transitive closure of subgrouping, and $SUB^+(p)$ denotes the **set of transitive subgroupings** of *p*:
SUB+(p) := { s ∈ GR | (s, p) ∈ subgrouping+ }

The basic domain primitives described in this subsection can be used for grouping adds and attributes into hypermedia documents, but additional concepts are needed for creating navigational structures (hyperlinks) between the different entity types or the world outside. The next section introduces the basic access primitives provided for that purpose.

4.3.2. Basic Access Primitives

In order to reference information from other entity types than the base ER-type and from outside the application domain, a document type or element-grouping can contain access primitives like structural and navigational links or access structures as shown in Figure 4-6.

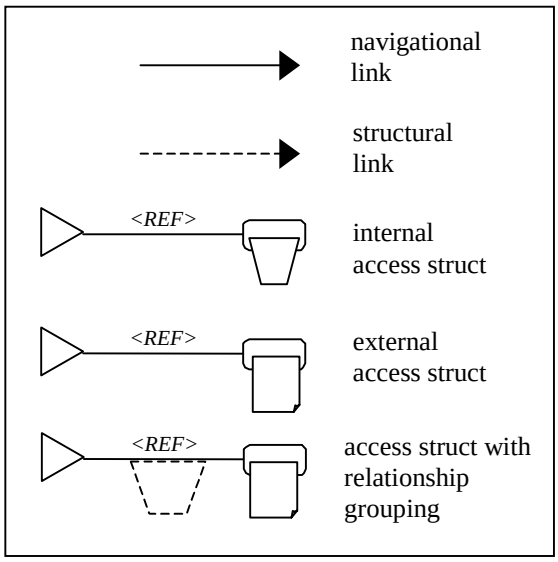

Figure 4-6: basic access primitives

Links in HMT originate from an anchor which can be an add, attribute or element-grouping and point to a target element, which is either a document type or an element-grouping within a document type. This corresponds to span-to-node and span-to-span links as defined in the Dexter Hypertext Reference Model [HS94].
For example, in a project overview document type, the attribute *title* can be defined as the anchor of a hyperlink. The target of this hyperlink could be either another document type containing, for example, an abstract and other information about the current project, or a certain element within this document type.

To give a more precise definition, HMT distinguishes two types of links: **navigational links** and **structural links**. The difference lies in the set of anchors and targets allowed for each kind of link:

- Navigational links may originate from any kind of element-grouping and are pointing to general document types, general element-groupings or external sources. For example, any document type may contain a navigational link pointing back to some kind of starting document (in most cases modeled as a general document). Navigational links are represented by a simple arrow.
- Structural links can only originate from ER element-groupings and lead again to ER document types or elements having the same base ER type. In other words, structural links are pointing to a different presentation of the same structure, that means they remain within a given structure (which is either an entity or a relationship). For example, a project overview document type may contain a structural link pointing to another project document type where more information on the current project is given. The design primitive for structural links is an arrow with a dotted line.

For the remainder of this work, we simply speak of HMT links when referring both to structural and navigational links.

While HMT links are hypermedia constructs that do not depend on the application data, **access structs** are used for showing the interdependencies between the entities and relationships of the application domain. Access structs have a source document type or element-grouping and reference a (possibly empty) set of entities of another ER type. For example, a project overview document type may contain an access struct referencing the scientists working in that project (relationship *participates_in*). There are three ways of specifying the subset of the set of all possible targets (labeled "<REF>" in Figure 4-6):

- **Relationship reference**
  By using the name of a relationship type, the access structure references all entities of the target entity type, which are related to the current entity by this relationship type. The project-employee example mentioned before is such a relationship reference. Both the source and the target document type of a relationship reference must be ER document types, and their base ER types have to correspond to the entity types the relationship references. If the relationship itself has attributes, an additional relationship element-grouping (drawn with a dotted line in Figure 4-6) can be attached to the access structure. The contents of this additional element-grouping will be displayed together with the target element-grouping or document type.
- **Conditional reference**
  Similar to relationship references, a conditional reference specifies a subset of the set of all entities of the target entity type. Instead of using a relationship type, a condition can be used for specifying the desired entities. Interpreting entity types and relationship types as relations and attributes as columns of the relational model described by Codd [Cod90], this condition may be specified using the extended theta select operator on the target ER type or the extended (theta) join operator on both ER types involved (see Definition 4.14).
  A conditional reference can be used together with a relational reference: For example, an access struct from departments to scientists can be defined using the relationship reference *works_in* and a condition *age>40*, specifying all employees of a certain department which are older than 40 years. This type of reference is called a **conditional relationship reference**.
- **Total reference**
  A total reference simply addresses all entities of the corresponding target entity type and is labeled with *. In our sample scenario, a possible access struct with a total reference could be a list of departments reachable from the starting document. While the target for such a total reference has to be an ER document type, the source can be any kind of document type.

Each access struct has a document type or element-grouping attached to the target ER type, specifying how (that means using which attributes, links or access structs) the referenced entities or relationships have to be presented. If a document type is provided, the referenced set of target entities or relationships will be presented within a new document and only a link

pointing to this document is inserted into the current document. This is called an **external access struct**. If an element-grouping is attached to the target ER type, then the referenced entities or relationships will be included right inside the current document, which is called an **internal access struct**.

Although the distinction between internal and external access structs might also be considered to belong to the logical design step, we see it as a matter of conceptual hypermedia design, because a decision for either of the two possibilities requires to select the right domain primitive (document type or element-grouping) as the target of the access struct. On the logical level, the conceptual HMT model should not be changed, so the decision has to be made during the conceptual design step.

A sample HMT diagram for the projects overview document together with its corresponding HTML representation can be seen in Figure 4-7. We'd like to stress that the HTML pages shown in the screenshots are the result of a complete HMT modeling process including logical and layout design, so not all information used to create these HTML pages can be found in the conceptual schemas presented. For example, the order and the labels of the attributes will be specified during the logical design step and thus cannot be found in the corresponding conceptual schemas. Nevertheless, we find it necessary to give some hints on how the resulting documents of the sample conceptual HMT schemas may look like.

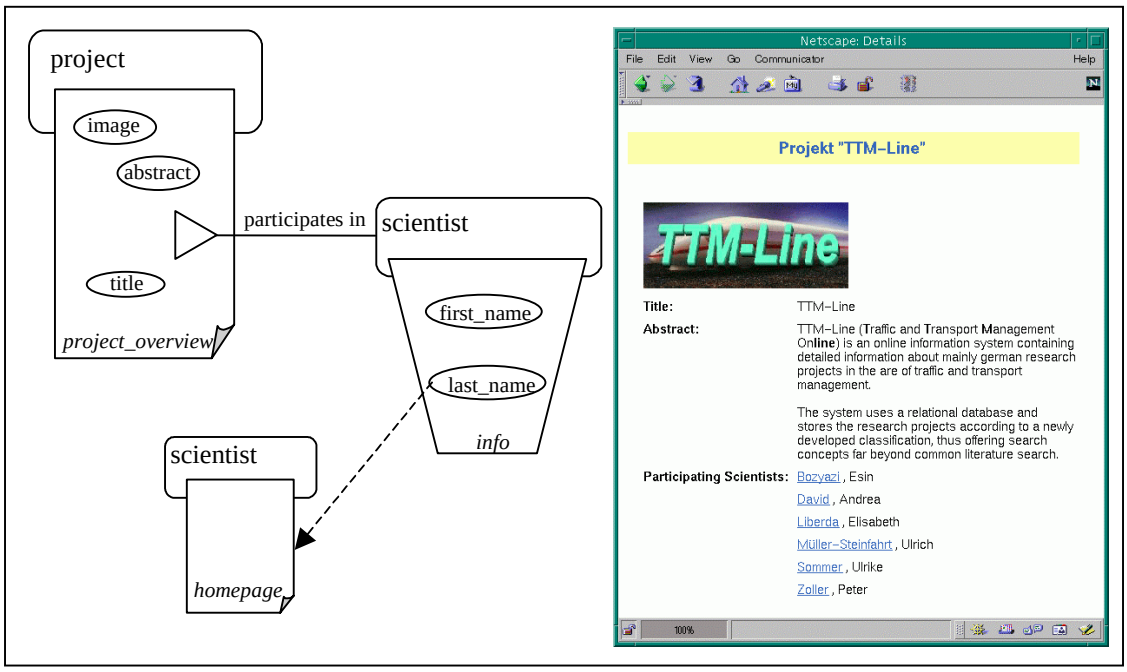

Figure 4-7: HMT schema of the project overview document type

The *project overview* document type is an ER document type with base ER type *project*, consisting of three attributes (*image*, *title*, *abstract*) and an access struct on the entity type *scientist*. This access struct uses a relationship reference (the *particicpates_in* relationship) and the *info* element-grouping as the target of the access struct, which causes the information about participating scientists to be displayed right inside the project overview document

(internal access struct). The *scientists_info* element-grouping contains two attributes, one of which (*last_name*) is used as the anchor of a structural link pointing to the *homepage* document of the corresponding scientist.

Formally, the basic access primitives of HMT are defined as follows:

A navigational link originates from an anchor (which can be an element-grouping, add or attribute) inside a document type or element-grouping, and points to either an external source or a general document type within the application:

Definition 4.12 (navigational link):

Let EXT be a set of external hypermedia presentations and ε be the empty element-grouping, then a **navigational link** is a tuple $l_{\text{nav}} = (s, g_a, d_t, g_t)$, with $s \in DOC \cup GR$ being the **source of the link**, $d_t \in DOC^{general} \cup EXT$ being the **target**, and $g_t \in SUB^+(d_t) \cup \varepsilon$ being a subgrouping of d_t called **target grouping**.
If s is an ER document type or ER element-grouping with base ER type e, then $g_a \in SUB^+(s) \cup ATTR_s \cup ADDS$, otherwise $g_a \in SUB^+(s) \cup ADDS$. G_a is called the **anchor** of the link.
$LINK^{nav}$ denotes the set of navigational links.

If the target grouping g_t is not the empty element-grouping, then l is called a span-to-span link, otherwise span-to-node link [HS94]. Of course there can only be a target grouping if the target is a general document type with at least one element-grouping, so links to external sources are always span-to-node links.

Structural links are similar to navigational links, but can only be applied to ER documents and ER element groupings:

Definition 4.13 (structural link):

Let ε be the empty element-grouping, then a **structural link** is a tuple $l_{\text{struct}} = (s, g_a, d_t, g_t)$, with $s \in DOC^{ER} \cup GR^{\text{ER}}$ being the **source**, $g_a \in SUB^+(s) \cup ADDS \cup ATTR_s$ being the **anchor**, $d_t \in DOC^{ER}$ being the **target**, and $g_t \in SUB^+(d_t) \cup \varepsilon$ being a subgrouping of d_t called **target grouping**. $LINK^{struct}$ denotes the set of structural links.

Structural links are also called span-to-span or span-to-node links depending on the target grouping used. The set of HMT links is defined as the union of $LINK^{struct}$ and $LINK^{nav}$.

While structural links always lead to different presentations of the same entity or relationship and navigational links only point to general documents, access structs allow to reference entities or relationships of other ER types than the base type of the current document type or element-grouping. Three types of references are possible: relationship reference, total reference and conditional reference. Before defining an access struct, first a valid condition has to be specified:

Definition 4.14 (valid condition):

Let $u, v \in E$ be entity types, then a valid **condition** $\theta(u,v)$ is any condition expressed in an extended theta-select operator or an extended theta-join operator of the relational model ([Cod90]), where relations are to be substituted by u and v and columns are to be substituted by $ATTR_u \cup ATTR_v$. Theta stands for any of the following comparators:

1) equality
2) inequality
3) less than
4) less than or equal to
5) greater than
6) greater than or equal to
7) greatest less than
8) greatest less than or equal to
9) least greater than
10) least greater than or equal to

Now that we have specified valid conditions for one or two entity types, an access struct can be defined. Access structs originate from a source (a document type or element-grouping) and reference a subset of the set of entities or relationships of another ER type (called target). The actual subset of the set of target entities or relationships is specified by the reference of the access struct, which can be a relationship reference, a conditional reference, a conditional relationship reference or a total reference.

Definition 4.15 (access struct):

Let $u, v \in E$ be entity types and $\Theta(u,v)$ the set of valid conditions for u and v, then an **access struct** is a tuple $s = (d_1, d_2, r)$, with $d_1 \in DOC_u \cup GR_u$, $d_2 \in DOC_v \cup GR_v$, and $r \in (\Theta(u,v) \times R) \cup \Theta(u,v) \cup R \cup \{*\}$.

D_1 and d_2 are called **source** and **target** of access struct s, r is called the **reference** of the access struct. We speak of a **relationship reference** if $r \in R$, a **conditional reference** if $r \in \Theta(u,v)$, a **conditional relationship reference** if $r \in (\Theta(u,v) \times R)$, and a **total reference** if $r = \{*\}$. *ACS* denotes the **set of access structs**.

Of course, relationship references can only be used if the source of the access struct is an ER document type or an ER element-grouping.

Depending on whether a document type or an element-grouping is used for the target of an access struct, we distinguish two types of access structs:

Definition 4.16 (internal and external access struct):

Let $s = (d_1, d_2, r)$ be an access struct and v be the base ER type of d_2, then s is called an **internal access struct** if the target $d_2 \in GR_v$ is an element-grouping. If $d_2 \in DOC_v$ is a document type, then s is called an **external access struct**.

In addition to the basic domain and access primitives described so far, the following subsection discusses extended HMT primitives for modeling applications with advanced functionality.

4.3.3. Specialized ER Documents

Experiences with several database-driven online information systems have shown that the pure presentation of data as specified in section 4.3.1 is only one aspect of such systems. There are often additional requirements like, for example, querying or manipulating the underlying data source, which is not covered by the basic domain primitives described so far.
A solution to this problem is the introduction of so called **specialized ER documents**. These specialized ER documents improve the expressive power of HMT and allow to model not only passive presentation documents, but also query forms for searching the data source, documents for presenting the query results, and web interfaces for manipulating the underlying data source. Together with these specialized documents, the basic access structs are extended and specified more precisely. The concept of distinguishing between documents and element-groupings as described in the previous sections is retained without exception.

When analyzing the typical process of querying an online information system, three steps can be identified:

- First, the query is specified by filling in some forms being part of what we call a **query document**.
- Second, the query is evaluated and the results are displayed as a list of items, called the **result document**.
- The list of items on the result document usually contains a structural link for each result item pointing to a more detailed description, which is called the **detail document**.

The document type primitive described in section 4.3.1 corresponds to the third step of the query process. To distinguish this kind of document type from others, we will refer to it as a **detail document type** from now on (and the element-grouping will be called details element-grouping).
In addition to the three document types described before, often also means for manipulating and populating the underlying data source are required. For this purpose, we introduce the concept of **input documents**.

Before going into detail, we extend the basic specification of document types and element-groupings given in the previous section in order to allow specialized versions: A specialized ER element-grouping is an ER element-grouping with a class c ∈ {query, result, detail, input).

Definition 4.17 (specialized ER element grouping):

> Let $C = \{query, result, detail, input\}$ be the **set of ER document classes**,then a **specialized ER element grouping** is a tuple $(g, class_g)$ with $g \in GR^{ER}$ and $class_g \in C$.

Similarly, a specialized ER document type is an ER document type with a class c ∈ {query, result, detail, input):

Definition 4.18 (specialized ER document type):

Let $C = \{query, result, detail, input\}$ be the **set of ER document classes**, then a **specialized ER document type** is a tuple $(d, class_d)$ with $d \in DOC^{ER}$ and $class_g \in C$.

The following sections will present in detail the different classes of document types together with their specific access primitives.

4.3.3.1 Query Document Types

Query document types represent documents containing input forms for querying the underlying database. Similar to detail document types, we distinguish between document types and element-groupings. Both may consist of attributes, adds, navigational links, internal access structs, or query element-groupings. Query document types are distinguished from detail document types by a question mark in the upper left corner of the document symbol. (see Figure 4-8). Query element-groupings can also be marked that way, but since query document types are only allowed to reference query element-groupings, the question mark in query element-groupings may be omitted if the context is clear.

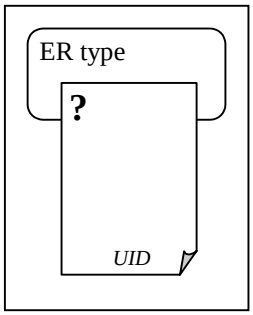

Figure 4-8 : query document type

Similar to general document types, a query document type has only one instance (the type itself can be regarded as an instance). For example, a query document type for the projects entity type of our sample scenario does not depend on the project entities (like the project overview document type does), but only on the ER type itself. Thus it is legal to use the expressions *query document* and *query document type* synonymously.

An attribute attached to a query document type or element-grouping means that some kind of input form will be provided where the user can restrict this attribute for searching. For example, a query document type named *project_search* could contain the attributes *title* and *abstract*, which would result in a query document containing one input form for the title and one for the abstract. Any entry made by the user will be matched against the corresponding values of all entities inside the database.
The kind of input form used (this could be, for example, a popup menu or a simple input line) is not specified at this point, but will be determined during the logical design step of HMT (see section 4.5.2). If several attributes are used in a query document type, it is left to the application to decide whether restrictions on different attributes are combined using the logical *and* or the logical *or* operator. It is suggested that the application offers a possibility for the user to take this decision himself, for example by providing a small popup menu after each input field allowing to choose between *and* and *or*.

By using access structs, input forms from related entity types can be included that allow defining complex search interfaces. Since there has to be a semantic relationship between two entity types if they are to be used for defining an ingenious access struct, only relationship references are allowed to be used for that purpose. For example, a document type for querying research projects may include an access struct on scientists using the *participates_in*

relationship. By this way, an input form for the scientist's name can be provided allowing to search for projects a certain scientist is working in. The scientist element-grouping used could in turn include another access struct to a third entity type (for example *departments*) so that there would also be a possibility to search for projects depending on the department the participating scientists belong to. This allows using transitive relationships as well.
Regarding internal and external access structs, it makes no sense to offer a query document with a hyperlink pointing to another query document, if the restrictions on the second document should be combined with those of the first input form. A better solution is to include this second input form right into the first query document, because this is much easier for the user to handle. For this reason, only internal access structs are allowed for query document types.

Adds and navigational links have the same functionality as described for the basic domain and access primitives. Using structural links in query documents is not possible, because unlike detail documents, query documents do not represent single entities.

Submitting the query form can be interpreted as navigation from the query document to a new document containing the query results. Each query document contains such a virtual link (called *form submitter*) pointing to a result document type (see next subsection) containing the query results. To distinguish from ordinary links, these form submitters are represented by an arrow with two lines as shown in Figure 4-9.

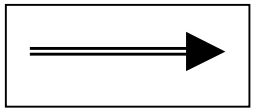

Figure 4-9: form submitter primitive

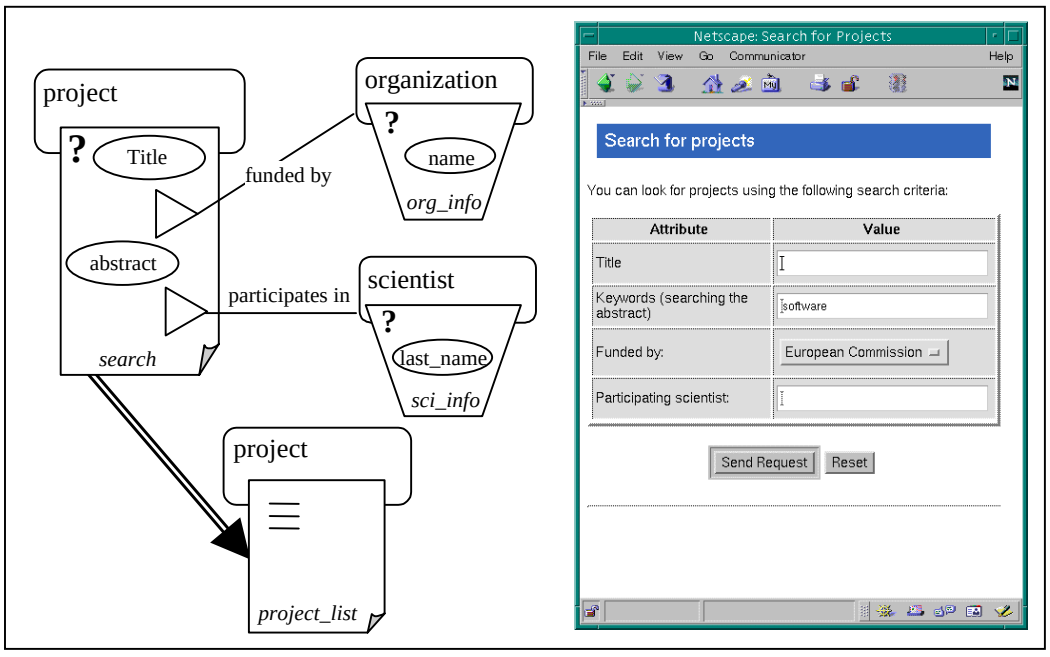

Figure 4-10: HMT schema of a project search document

Figure 4-10 shows a HMT schema for a simple project search document type and the corresponding HTML document. The base ER type of the search document type is the *project*

entity type, and it contains the attributes *title* and *abstract*. In addition to this, two access structs are defined: using the *funded_by* relationship and the *org_info* query element-grouping as its target, the name of a funding organization can be used for restricting the set of projects. The *participates_in* relationship with its target element-grouping *sci_info* allows searching for projects a certain scientist is working for. The target of the form submitter is the result document type *project_list*, which will be described in the next subsection.

The following formal specification is based upon the basic specification of sections 4.3.1, 4.3.2, and 4.3.3, and addresses only those issues that have been modified or introduced during this subsection.

Query document types and query element-groupings are defined as follows:

Definition 4.19 (query document type):

Let $q = (d, class_{\mathrm{d}})$ be a specialized ER document and $e \in E \cup R$ an ER type, then $DOC^{query} = \{ q \mid class_{\mathrm{d}} = \text{query} \}$ denotes the **set of query document types**, and DOC_e^{query} denotes the set of query document types with base ER type *e*.

Definition 4.20 (query element-grouping):

Let $q = (g, class_{\mathrm{g}})$ be a specialized ER element-grouping and $e \in E \cup R$ an ER type, then $GR^{query} = \{ q \mid class_{\mathrm{q}} = \text{query} \}$ denotes the **set of query element-groupings** and GR_e^{query} denotes the set of query element-groupings with base ER type *e*.

If access structs are used in query document types, they have to be internal, because the semantics of submitting a form related to the current query document type by an external access struct is not clear and would confuse the users. The reference type of a query access struct must also not be a total reference, because then there would be no relationship between the target and the current ER type at all, making the inclusion of a query form from the target ER type completely useless. Conditional references may be used if dependencies or correlations between two entity types have not been modeled as a relationship type.

Definition 4.21 (query access struct):

Let *ACS* be the set of access structs, then the **set of query access structs** is defined as $ACS^{query} = \{s = (d_1, d_2, r) \in S \mid r \neq \{*\}, d_1 \in DOC^{query} \cup GR^{query} \text{ and } d_2 \in GR^{query}\}$.

A query form submitter is used to specify a result document displaying the query results. It consists of a query document type or query element grouping (the source) and a result document (the target):

Definition 4.22 (query form submitter):

Let DOC^{result} be the set of result document types, then a **query form submitter** is a tuple $u = (d_1, d_2)$, with $d_1 \in GR^{query} \cup DOC^{query}$ being the **source** and $d_2 \in DOC^{result}$ being the **target**. SBM^{query} denotes the set of query form submitters.

The following subsection discusses the result document type used for displaying the results of query documents.

4.3.3.2 Result Document Types

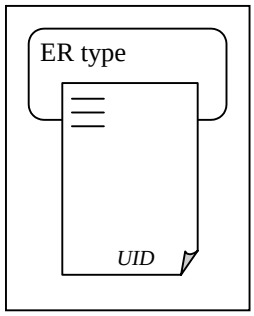

Figure 4-11: result document primitive

Result document types are used to display a set of entities of a given ER type. The result document type primitive is represented by a document symbol with three lines in the upper left corner (see Figure 4-11) and inherits all elements of the basic document type described in section 4.3.1, which are groupings, adds, attributes, links and access structs. But while the detail document type uses its specification to display one entity of an ER type, a result document type applies its specification on each entity of the set of entities to be displayed. For example, a result document type for the project entity type could contain the attributes *title* and *start*, which would lead to a document containing a list of projects where each item would be displayed with its title and date of initialization. The information about which entities have to be displayed is specified by the access struct or the query document that is referencing the result document.

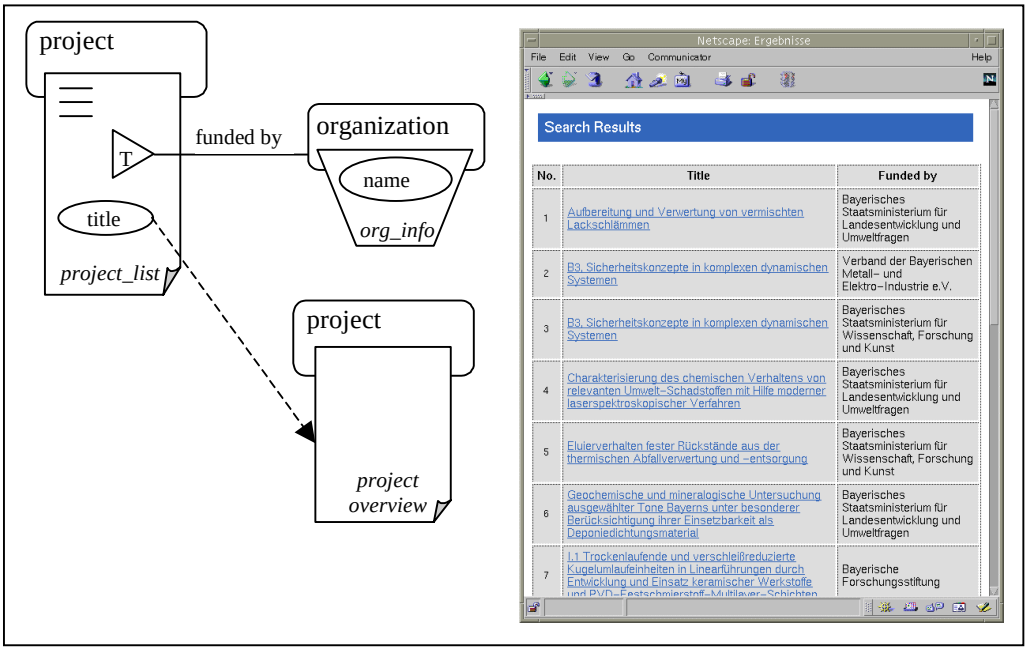

Figure 4-12: HMT schema of a project result document

Figure 4-12 shows a sample project result document type and the corresponding HMT schema. The result document type is named *project_list* and contains one attribute (*title*) and an access struct on the entity type *organization*. The target of the access struct is the element-grouping *org_info* (indicating that this is an internal access struct), which includes only the name of the funding organization. The *title* attribute of the project is used as the source of a structural link pointing to the *project_overview* detail document type showing further information on the corresponding item. The screenshot on the right side shows the list of projects, where the title is shown in the second column and the name of the funding organization is listed in the third column. The first column of the table containing a number for each item is created automatically and is not specified during conceptual hypermedia design, but is one of the features the HMT logical design step deals with.

For a result document type, the number of items to be displayed cannot be estimated when designing the application; it depends, for example, on the search phrases entered by the user. In some cases, there may be result sets containing hundreds or thousands of items to be displayed, which leads to extremely large documents. One solution for this problem is to divide the list of entities among several linked documents. In HMT, the maximum number of elements for a single result document can be specified during the logical hypermedia design phase. This improves readability and avoids large documents with long download times, but still overloads the user with perhaps hundreds of results. A more sophisticated search interface enables the user to interactively refine his query if the result set is too large.

In HMT, this can be achieved by allowing a result document type to contain one query element-grouping. This query element-grouping can be used to refine the current query until the result set has reached a reasonable size. In order to display the results of the refined query, a result document has to be specified by using the submitter primitive. Of course the target of the submitter may also be the same document type.
Unlike the result element-groupings of a result document, this query element-grouping is only displayed once, not for each element of the result set.

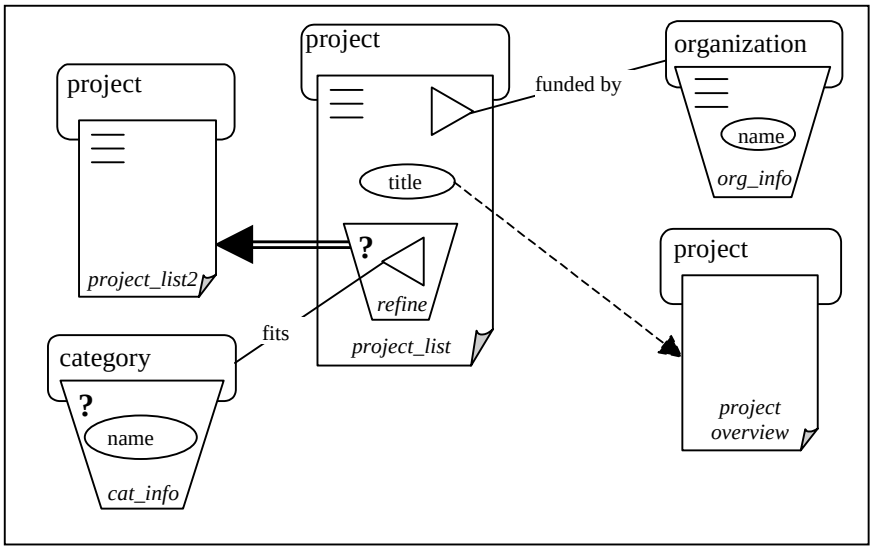

Figure 4-13: result document with query refinement capabilities

Figure 4-13 shows the extended *project_list* result document type, now containing a query element-grouping named *refine* for refining the query. This query element-grouping contains an internal access struct on the *category* entity type using the *fits* relationship. This allows selecting only those projects fitting a certain category. If the query element-grouping is used, the refined list of projects is displayed by the use of the result document type named *project_list2*.

It is not required that the query element-grouping in the result document is identical to the one used in the initial query document. You could, for example, start searching for projects using

a simple keyword search, and afterwards refining your query with a more advanced search covering scientists, departments or categories.

Formally, result element-groupings and result document types are defined as follows:

Definition 4.23 (result document type):

Let $q = (d, class_d)$ be a specialized ER document and $e \in E \cup R$ an ER type, then $DOC^{result} = \{ q \mid class_d = \text{result} \}$ denotes the **set of result document types**, and DOC_e^{result} denotes the set of result document types with base ER type *e*.

Definition 4.24 (result element-grouping):

Let $q = (g, class_g)$ be a specialized ER element-grouping and $e \in E \cup R$ an ER type, then $GR^{result} = \{ q \mid class_q = \text{result} \}$ denotes the **set of result element-groupings** and GR_e^{result} denotes the set of result element-groupings with base ER type *e*.

In addition to result element-groupings, also query element-groupings may be included by result documents in order to provide query refinement capabilities. Of course an included query element-grouping must have the same base ER type as the including result document or element-grouping.

With the introduction of specialized document types, the definition of structural links has to be changed. Each ER document type presenting one or more entities might be the source of a structural link (i.e. result, detail and input document types), and each document type presenting exactly one entity may be the target of a structural link (i.e. detail and input document types):

Definition 4.25 (exact definition of a structural link):

Let ε be the empty element-grouping and DOC^{input} be the set of input document types, then a **structural link** is a tuple $l_{struct} = (s, g_a, d_t, g_t)$, with
$s \in (DOC^{ER} / DOC^{query}) \cup (GR^{ER} / GR^{query})$ being the **source**,
$g_a \in SUB^+(s) \cup ADDS \cup ATTR_s$ being the **anchor**, $d_t \in (DOC^{detail} \cup DOC^{input})$ being the **target**, and $g_t \in SUB^+(d_t) \cup \varepsilon$ being a subgrouping of d_t called **target grouping**.

4.3.3.3 Detail Document Types

Detail document types and detail element-groupings are almost identical to the basic domain and access primitives as described in section 4.3.1. A detail document represents one entity of its base ER type using attributes, adds, groupings, links and access structs. In contrast to the other kinds of document types, detail document types can use four different kinds of access structs: **Table of Contents** (TOC) **Guided Tour**, **Guided Table of Contents** and **Slide Show**. The corresponding HMT design primitives are labeled with a *T* (for TOCs), *G* (for Guided Tours), *GT* (for Guided TOCs) and *S* (for Slide Shows) in the anchor triangle of the access struct symbol (see Figure 4-14).

These access structs differ in the way they offer access to the information referenced:

- A TOC provides a list consisting of the entities referenced. The contents of this description are defined by the result element-grouping (in case of internal TOCs) or the result document type (in case of external TOCs) that is attached to the access structure. Usually, a description for a TOC will contain only few attributes and one structural link pointing to a detailed presentation of the entity. In our sample scenario, the presentation of a TOC on research projects could be defined as an element-grouping containing the title and a hyperlink pointing to the project's overview document.
- A Guided Tour does not display a list of all relevant entities, but shows one single entity of the set together with links pointing to the next or maybe the previous entity. The presentation of each entity referenced by the Guided Tour is defined by the document attached to it. Additional information like the order of the tour elements or the buttons used for selecting the next or previous tour entity will be specified during the logical and layout design step of HMT.
- Combining these two concepts leads to a Guided Table of Contents (GT) with both direct access to all items referenced and a Guided Tour.
- Slide Shows are similar to Guided Tours, except for the fact that they automatically display the referenced entities one after the other instead of waiting for the user to select the next or previous item. The information about the overall length of a Slide Show or the time used to display each entity referenced are specified during the temporal design step (see section 4.5.4).

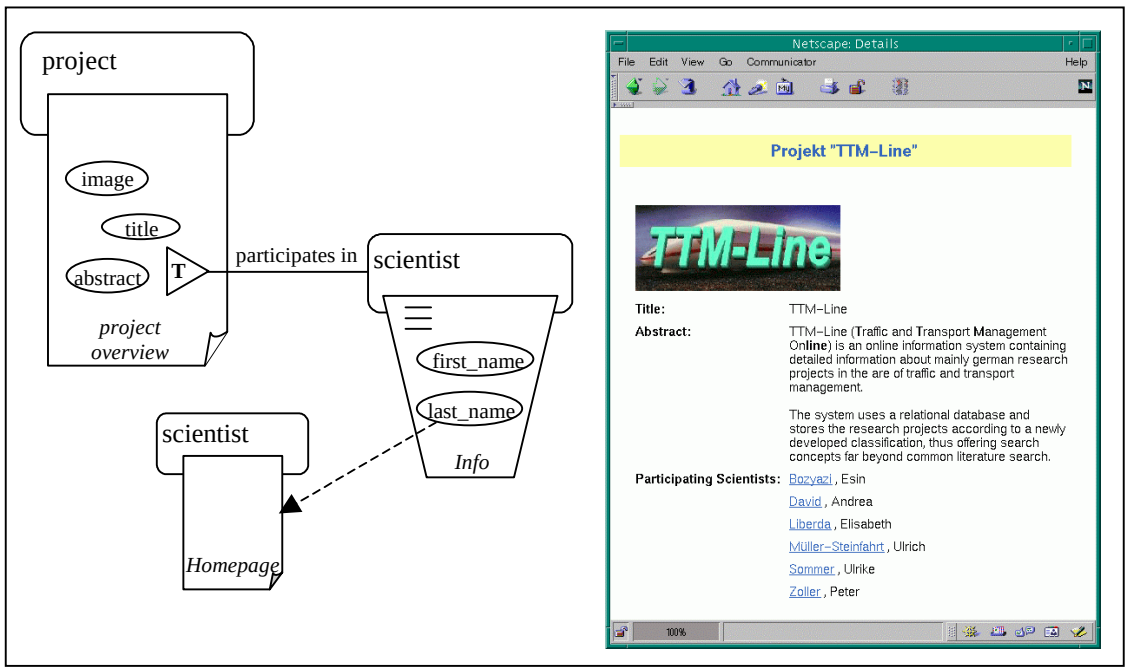

Figure 4-14: HMT schema of the project overview document

Similar to the distinction between internal and external access structs, the specification of the type of the access struct (TOC, Guided Tour, Guided TOC or Slide Show) could be regarded

as a matter of logical rather than conceptual hypermedia design. But again, the need to specify either an element-grouping or a document type as the target of the access struct depending on its type requires this decision to be made during conceptual hypermedia design.

Figure 4-14 shows the modified version of the HMT schema for the *project_overview* document type. Besides the attributes *image*, *title* and *abstract*, this document type contains an internal TOC on the scientists entity type using the *participates_in* relationship in order to specify the set of employees working for that project. The target of the TOC is the *info* element-grouping, which consists of the attributes *first_name* and *last_name*, the latter of which is used as the source of a structural link pointing to the scientist's homepage.

The basic formal specifications of sections 4.3.1 and 4.3.2 can be extended as follows:

Definition 4.26 (detail document type):

Let $q = (d, class_d)$ be a specialized ER document and $e \in E \cup R$ an ER type, then $DOC^{detail} = \{ q \mid class_d = \text{detail} \}$ denotes the **set of detail document types**, and DOC_e^{detail} denotes the set of detail document types with base ER type *e*.

Definition 4.27 (detail element-grouping):

Let $q = (g, class_g)$ be a specialized ER element-grouping and $e \in E \cup R$ an ER type, then $GR^{detail} = \{ q \mid class_q = \text{detail} \}$ denotes the **set of detail element-groupings** and GR_e^{detail} denotes the set of detail element-groupings with base ER type *e*.

Access structs for detail document types or detail element-groupings carry additional information about their type of access struct: Guided Tour (G), TOC (T), Guided TOC (GT) and Slide Show (S):

Definition 4.28 (detail access struct):

Let *ACS* be the set of access structs, then the **set of detail access structs** is defined as $ACS^{detail} = \{ (s, type_s) \mid s = (d_1, d_2, r) \in ACS, d_1, d_2 \in (DOC^{detail} \cup GR^{detail}), type(s) \in \{T, G, GT, S\} \}$. If $type_s$ is a TOC or a Guided TOC, then the target of the access struct has to be an element-grouping, otherwise it has to be a document type.

Query, result and detail document types are used for modeling advanced information retrieval interfaces of hypermedia applications. In the following section, we introduce the input document type, which can be used for manipulating the application's data source.

4.3.3.4 Input document types

Input document types are used to provide an interface for manipulating the underlying data source by either updating existing entries or creating new ones. Their corresponding primitive is a document type with an arrow pointing into a barrel in the upper left corner (see Figure 4-15). Similar to query document types, input document types may contain adds, attributes, links, internal access structs, element-groupings and form submitters.

For each attribute contained in an input document, an input form of a certain type will be provided where the user can enter a value or change an already existing one. A detailed

description of how the type of the input form is specified and what types are available is provided in section 4.5. Attributes not contained in the input document will be assigned a default value, which can be specified during the logical design phase. If no default value is specified there, the attribute will be assigned a NULL value.

Internal access structs in input documents allow the user to relate entities of other entity types to the entity that is currently created or updated. In database terms, foreign keys are defined by using these internal access structs. For example, an input document type for the *project* entity type could contain an access struct on *categories* that allows specifying the categories relevant for the current project entity. In this example, a relationship reference has been used for the access struct. In order to also be able to specify dependencies between entities where no relationship type has been defined in the ER schema, conditional references may also be used for input access structs, whereas total references do not provide useful semantics in that case.

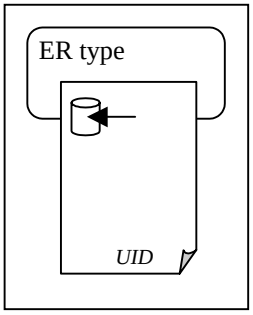

Figure 4-15: input document primitive

Input documents can be referenced by both navigational and structural links. In the first case, the input document is used for creating a new entry for the given base ER type, because no information about a certain entity is available. For example, a general document type named *administration* acts as the central document for administration purposes and includes navigational links on input documents for projects, scientists and departments, which allows inserting new entries into the database.

In contrast to this, a structural link leading to an input document causes the values of the current entity to be displayed for manipulation. For example, the result document type for projects shown in Figure 4-12 could contain a structural link pointing to a project input document type instead of the project overview document type. If one entity on the project result document is selected by the user, the corresponding input document is presented.

Finally, a form submitter is used to initiate the storage process. It points to a detail document type providing some kind of feedback to the user. For example, some characteristic attributes of the saved entity can be presented again together with an acknowledgement that this entry has been saved successfully. Form submitters can be assigned to both input documents and input element-groupings, but a specific input document must only contain one submitter. If more submitters are specified, only the top-most is considered as valid, the others are ignored.

Figure 4-16 shows a sample HMT schema of a project input document type named *project_input*. It contains the attributes *title*, *abstract*, *begin*, *end* and *image*, plus internal access structs on *scientists*, *categories* and *funding_organizations*, each using a relationship reference. The submitter target is the *input_conf* detail document type, which presents a confirmation to the user containing the title of the project saved.

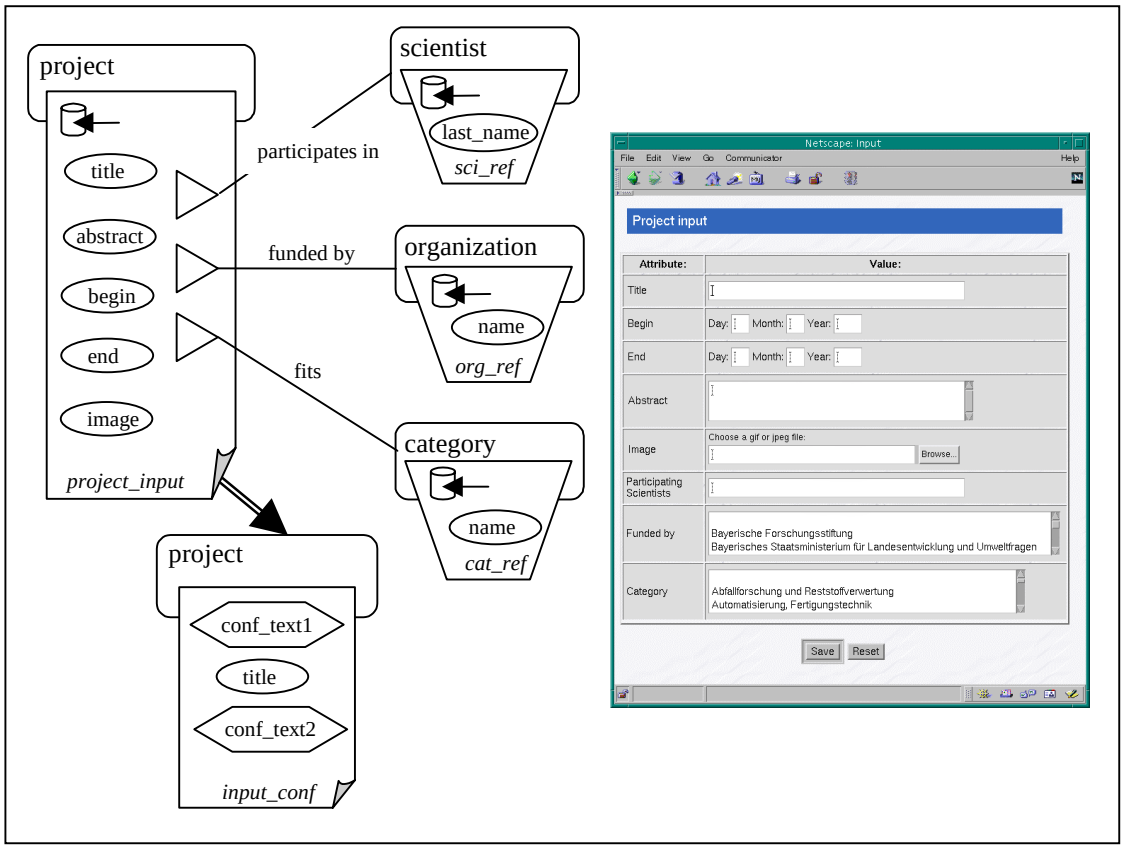

Figure 4-16: HMT schema for a project input document

Formally, input document types are specified as follows:

Definition 4.29 (input document type):

Let $q = (d, class_d)$ be a specialized ER document and $e \in E \cup R$ an ER type, then $DOC^{input} = \{ q \mid class_d = \text{input} \}$ denotes the **set of input document types**, and DOC_e^{input} denotes the set of input document types with base ER type e.

Definition 4.30 (input element-grouping):

Let $q = (g, class_g)$ be a specialized ER element-grouping and $e \in E \cup R$ an ER type, then $GR^{input} = \{ q \mid class_q = \text{input} \}$ denotes the **set of input element-groupings** and GR_e^{input} denotes the set of input element-groupings with base ER type e.

Each input document has a form submitter pointing to a detail document type:

Definition 4.31 (input form submitter):

An **input form submitter** is a tuple $u = (d_1, d_2)$, with $d_1 \in GR^{input} \cup DOC^{input}$ being the **source** and $d_2 \in DOC^{detail}$ being the **target**. SBM^{input} denotes the set of input form submitters.

Input access structs are used to relate entities of different ER types to the current entity. Only relationship references or conditional references are allowed, and the access struct must be internal:

Definition 4.32 (input access struct):

Let *ACS* be the set of access structs, then the **set of input access structs** is defined as $ACS^{input} = \{s = (d_1, d_2, r) \in S \mid r \neq \{*\}, d_1 \in DOC^{input} \cup GR^{input}$ and $d_2 \in GR^{input}\}$.

4.3.4. Summary

In this section, the conceptual hypermedia model of HMT has been described and formally specified. The HMT conceptual design step is based on the ER schema of the application domain developed during the second HMT design step (ER design). Figure 4-17 shows a summary of the HMT conceptual design primitives.

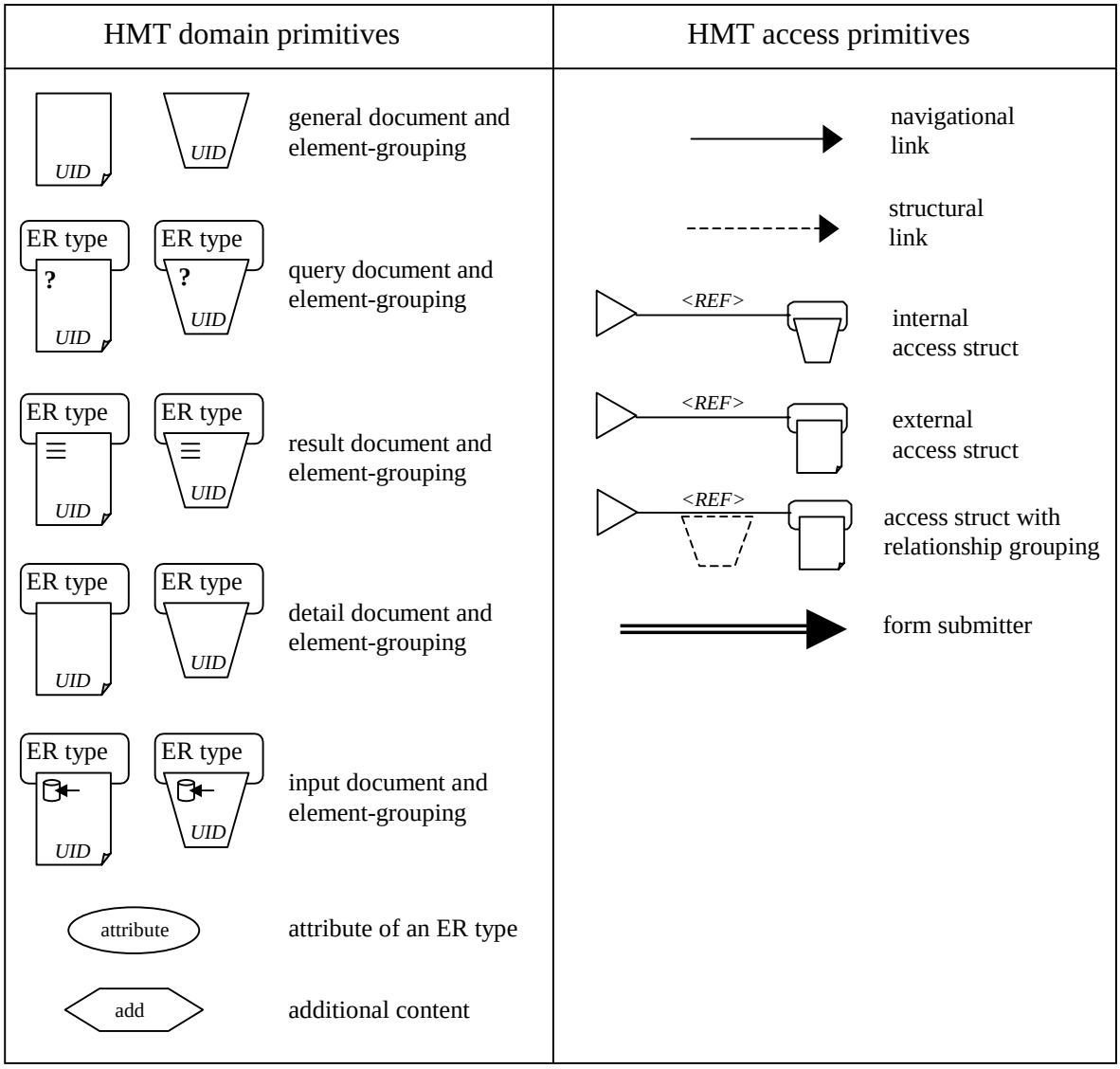

Figure 4-17: HMT conceptual design primitives

Document types, element-groupings, adds and attributes have been introduced as the basic domain primitives. ER document types are assigned a base ER type from the ER schema of

the application domain allowing them to contain attributes from this ER type. General documents lack this ER type and therefore can only contain adds or element-groupings.
In order to include information from related entity types and allow navigation within the application, a set of basic access primitives like structural links, navigational links and access structs with relationship references, conditional references or total references have been defined. Extending the basic ER document type, the introduction of query documents, result documents, detail documents and input documents allows to model not only passive documents for information presentation, but also interactive documents for querying or updating the underlying data source. For each of these specialized documents, the relevant sets of domain and access primitives have been described in detail.

As a result of this design step, a conceptual HMT schema can be defined as follows:

Definition 4.33 (conceptual HMT schema):

> A **conceptual HMT schema** is a tuple $h = (s, DOC, GR, ADDS, ACS, LINK, SBM)$ with s being an ER schema, *DOC* being a set of document types, and *GR* being a set of element-groupings, *ADDS* being a set of adds, *ACS* being a set of access structs, *LINK* being a set of links, and *SBM* being a set of form submitters.

4.4 Authorization Design

Considering today's most popular hypermedia system world wide web, applications to a great extent do not only contain public information, but also data intended to be visible only for certain users or groups. This could be, for example, internal information about a certain project, which should only be accessible for project partners, or some special services available only for customers or registered users. To this day, access control in WWW applications is usually handled on the physical layer, namely by the web server, which implies several disadvantages:
Access restrictions have to be specified separate from the hypermedia design process, and are influenced by the architecture of the CASE-tool used: If the pages are materialized in the file system, then their location has to be chosen according to the desired access restrictions (which are defined at directory level by the web server). If the pages are to be generated dynamically, only the entire application can be controlled by restricting the gateway, or the application has to implement its own access control mechanism.
Standard web server authentication and authorization only apply to files within the document directory tree of the web server, thus only whole documents (except for inline images) can be protected, a finer granularity cannot be specified. Additionally, changes to restriction policies require reorganizations in the file system. On the other hand, moving files can influence access restrictions. These considerations lead to the conclusion that access control should be a matter of hypermedia application design. In HMT, the authorization design step deals with all aspects of access restriction.

Besides traditional access control techniques like discretionary access control (DAC) or mandatory access control (MAC)**,** role based access control (RBAC) [SCF+96] has been

discussed as a promising alternative in recent years. Moreover, it has been shown that RBAC can be used to simulate both DAC and MAC [NO96,San96,SM98]. The following sections describe the basic principles of Role Based Access Control and how HMT uses this technique.

4.4.1. Role Based Access Control

Originally, the concept of role has been examined in sociology, where it is defined as the set of duties and rights associated with a position, which is assigned to a person occupying this position [TB79]. This concept has been adapted to the area of computer science with the consistent notion of role as

> *"... a job or function within the organization that describes the authority and responsibility conferred on a user assigned to the role." [SCF+96]*.

This very intuitive approach has been motivated by the fact, that in most organizations and institutions access restrictions are based upon the role of a user and not upon the user himself. Roles are created for the different job functions in an organization and the users are assigned one or more roles depending on their responsibilities. This simplifies the task of administering access control policies for several reasons:

- For a given position, only one rule needs to be specified independent of how many users occupy this position.
- If users change positions, access restrictions do not have to be redefined.
- RBAC is policy neutral: additional security principles like separation of duties or role cardinality can easily be specified by declaring conflicting roles or limitations of the number of users assigned to a given role.

Roles can be grouped to form hierarchies, where superior roles (senior roles) inherit all the access rights of their inferiors. There is no general rule how these hierarchies are to be built, but [Mof98] identifies three approaches: An *isa role hierarchy* based on generalization, an *activity role hierarchy* based on aggregation, and a *supervision role hierarchy* based on the organizational hierarchy.
Each role is assigned a set of permissions and a set of users. Although a user can have more than one role, it may be necessary to restrict the number of roles at a given point of time. For example, the personalization of a document might depend on the user's role, and therefore only one role must be assigned to a user for the personalization algorithm to work properly. This can be achieved by introducing the concept of *user sessions* and assigning the roles currently activated to the user's session.
Constraints can be defined for permission assignment, role-to-session assignment or role hierarchies. For example, there might be a mutual exclusion for certain pairs of roles activated at the same time like *manager* and *administrator*. Figure 4-18 shows the basic RBAC model as defined in [SCF+96].

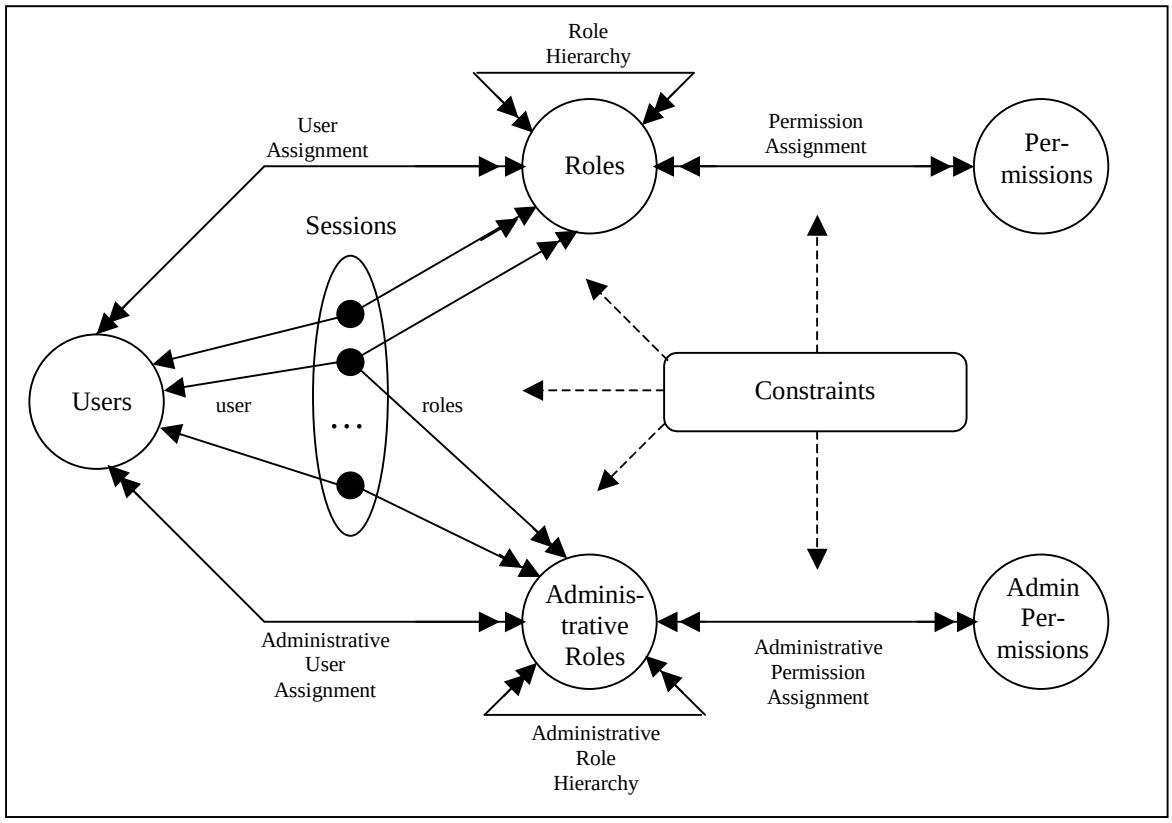

Figure 4-18: RBAC96 Model

Other advanced concepts like the scope of roles, separation of duties or activation and deactivation of roles have been subject to research [NO93,LS97,GB98,Mof98], but do not or only marginally influence the authorization design step of HMT. The reason is that this design step only deals with the assignment of roles to HMT documents or element-groupings, it is independent of role administration issues. Therefore we do not discuss such issues further.

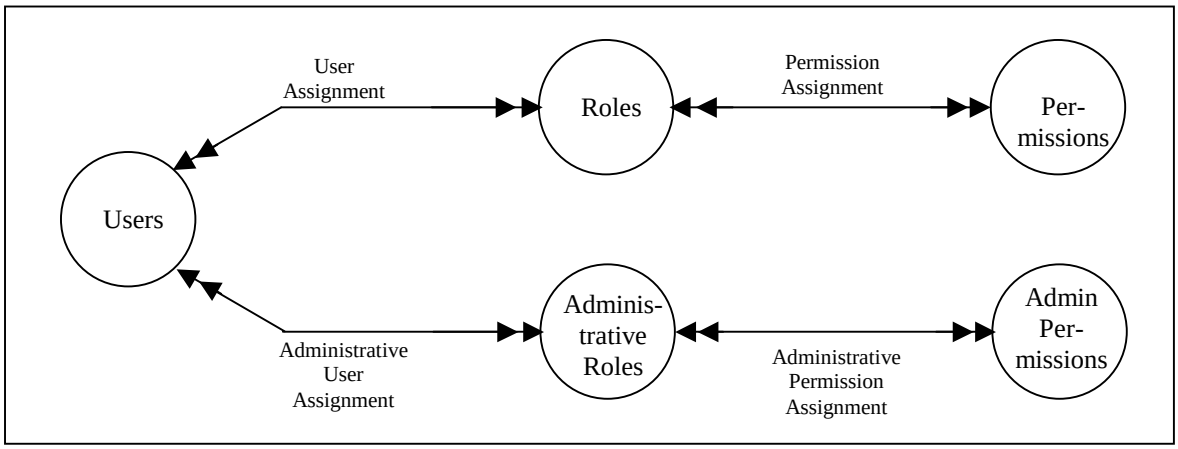

Figure 4-19: The HMT RBAC model

Since all roles of a user are assumed to be activated automatically during every session, the HMT authorization design step does not need the session concept shown in Figure 4-18. For

our purpose, a simplified, basic RBAC concept with flat and situation-insensitive roles is absolutely sufficient. Figure 4-19 shows the basic RBAC model as used by HMT. The introduction of advanced concepts like separation of duties, role cardinality, or role hierarchies is possible, but not required for full functionality. It is left to the administration tool to support these features or not, the authorization design step itself is not affected by such considerations.

4.4.2. RBAC in HMT

RBAC in HMT can be applied on two different levels: document level and element-grouping level. The differences between the two levels of application are discussed in the following subsections. There is, however, no difference between restricting a general document type/element-grouping or an ER document type/element-grouping.

Consequently, HMT authorization diagrams contain all kinds of documents and element-groupings, but ignore other conceptual design primitives like links or attributes. This improves readability and maintainability of HMT authorization diagrams. If single elements like adds or links are to be restricted, they have to be placed inside an element-grouping first. The close relation of HMT authorization diagrams to HMT conceptual schemas is the main reason for placing the authorization design step before logical design and layout design, although it could also have been specified as the last HMT design step. Organizing the HMT design process this way, each design step only relies on the previous one and extends the corresponding diagrams, there is no step switching back to a more simple view of the application.

Access restrictions are represented by a rectangle containing a key symbol and the name of the role that is allowed to access the document type or element-grouping (see Figure 4-20). The access restriction primitive is placed inside the document or element-grouping primitive. If no restriction is specified, the document type or element-grouping is accessible by every user.

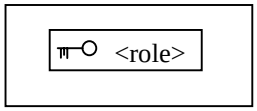

Figure 4-20: access restriction primitive

This strategy is different from many other systems like file systems or database management systems, where only the owner and users mentioned explicitly are allowed to access the information. Typical hypermedia systems are designed to be open to the public (like the World Wide Web) or are accessible only in places where the possible users are already authorized (like fairs or intranets). This means that access restrictions in hypermedia applications usually concern only a smaller part of the whole application, and therefore the default behavior of HMT documents and element-groupings is to be open to the public unless restricted explicitly.

Formally, a HMT authorization schema can be defined as follows:

Definition 4.34 (HMT authorization schema):

Let L be a set of roles, then a **HMT authorization schema** is a set $A \subseteq L \times (DOC \cup GR)$, with DOC being the set of document types and GR being the set of element-groupings.

For the remainder of this section, we use a modified version of our sample scenario. The project entity now contains a number of new attributes and relationships like *overall funding*, estimated *future expenses*, *annotations* of the management, or *internal reports* as shown in Figure 4-21.

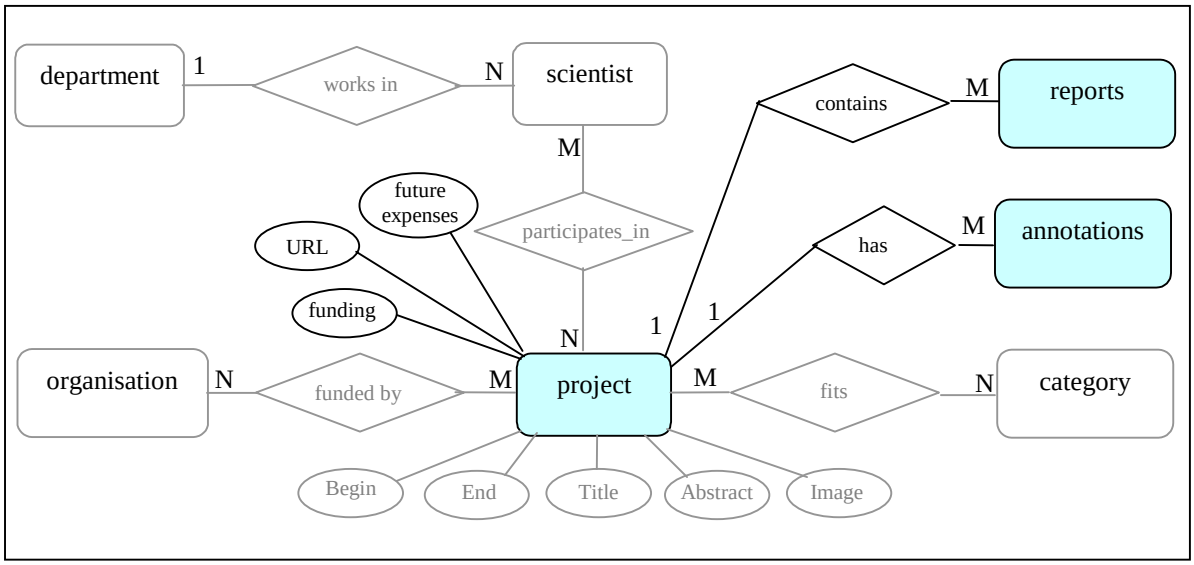

Figure 4-21: modified sample scenario

4.4.2.1 RBAC for documents: traditional access control

Traditional access control as, for example, in the World Wide Web is managed by assigning roles to document types in HMT. If a user requests such a restricted document, he must have the appropriate role in order to receive the document, otherwise some kind of access denied message will be given.

Figure 4-22 shows a modified version of the sample HMT conceptual schema for the project overview document from Figure 4-14 and the corresponding authorization diagram. The project overview document now contains an additional structural link leading to the project input document from Figure 4-16, which allows changing the current project's properties.

Of course only special users, for example the administrator, should be allowed to change the project's properties. Therefore access to the input document is restricted as shown in the authorization diagram on the right side of Figure 4-22.

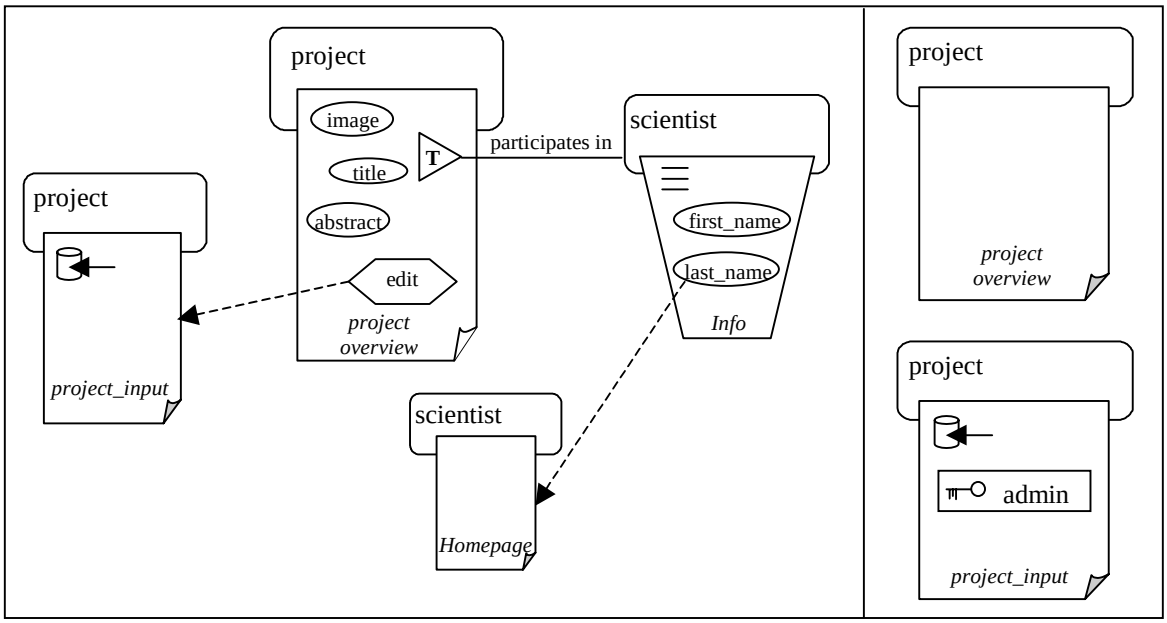

Figure 4-22: Access restriction on document level

Although an access restriction on a document restricts the entire document together with all included element-groupings, this restriction is not explicitly inherited to the related element-groupings. On the one hand, this is not necessary because a denial of access to a document automatically prevents all related element-groupings from being displayed in that context (*implicit inheritance*). On the other hand, explicit inheritance of access restrictions to included element-groupings could cause side effects, because an element-grouping may be used by multiple documents, some of which may have no or different restrictions.

4.4.2.2 RBAC for element-groupings: adaptive documents

Assigning roles to documents can be used for traditional access control with document granularity, but restricting access to single element-groupings has a completely different effect. If an element-grouping is assigned a role, only the users with the appropriate permission will see this element-grouping, the other users will be presented only the remaining unrestricted parts of the document (and those restricted parts the user has permission for).

This technique of role based access control on element-grouping level can be used to build personalized documents depending on the users role(s). Thus, often only one document has to be designed containing all possible pieces of information, where just different roles have to be assigned to the corresponding element-groupings.

Figure 4-23 shows the adaptive version of the projects overview document, where the information is divided into three element-groupings according to its level of security. The *basic_info* element-grouping contains the information that is available for the public. This refers to the project *title*, *begin* and *end*, the *abstract* and the project logo (*image*). The *internal* element-grouping contains information about the *funding* and *future expenses*, which

should only be accessible for project partners. The *annotations* element-grouping contains private remarks of the project management.

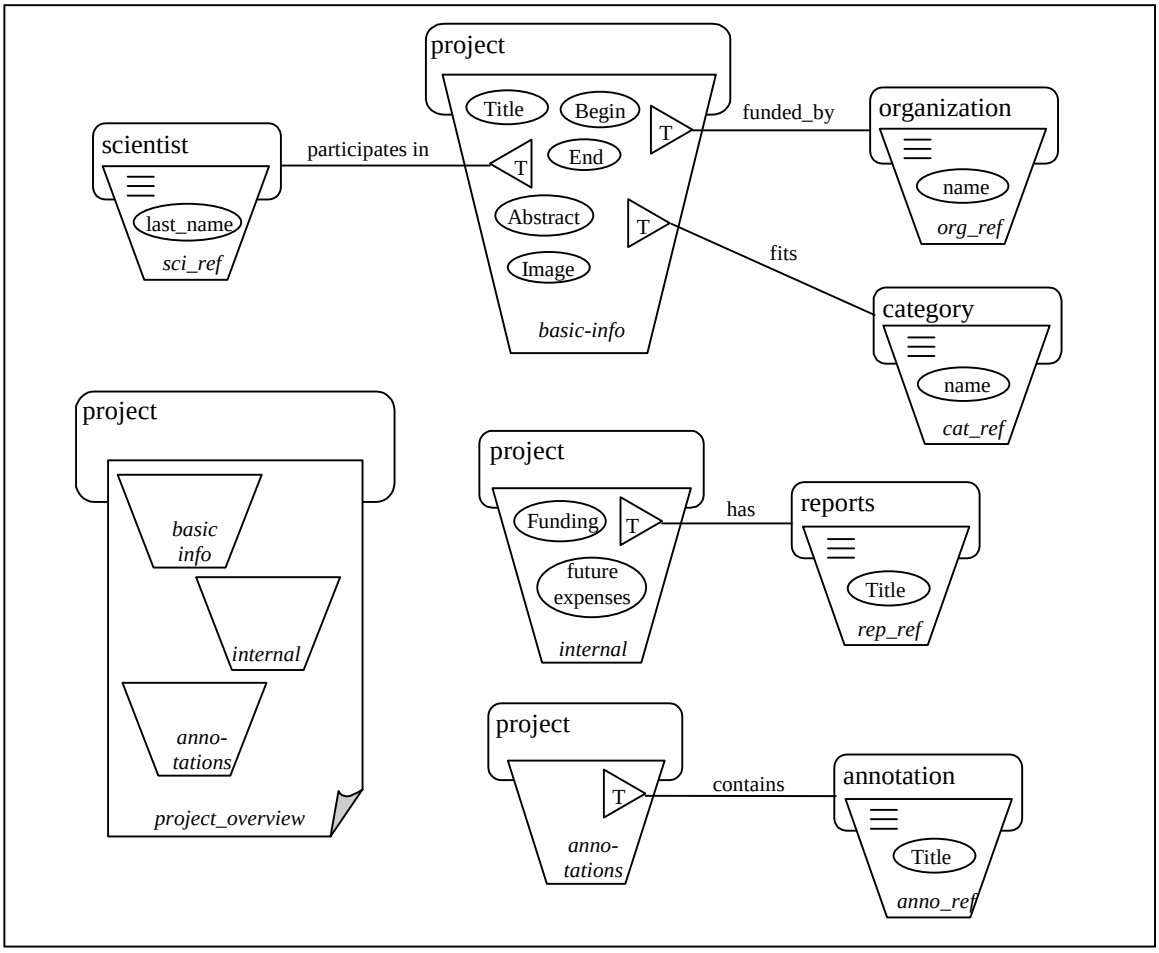

Figure 4-23: modified project overview document type

The corresponding HMT authorization diagram plus screenshots of the adaptive document from the manager's and an ordinary user's point of view can be found in Figure 4-24.

Using access restrictions on the element-grouping level for securing critical information is not the only application for personalized hypermedia presentations. Considering systems for educational purposes or distance learning, another possible area of application is the creation of documents for users with different levels of skills or knowledge. Roles like *beginner*, *advanced* or *expert* could be provided which may be chosen freely by the user when entering the hypermedia system. Also intranet information systems might benefit from this technique by allowing to create adaptive documents containing information tailored to the employee's tasks and interests.

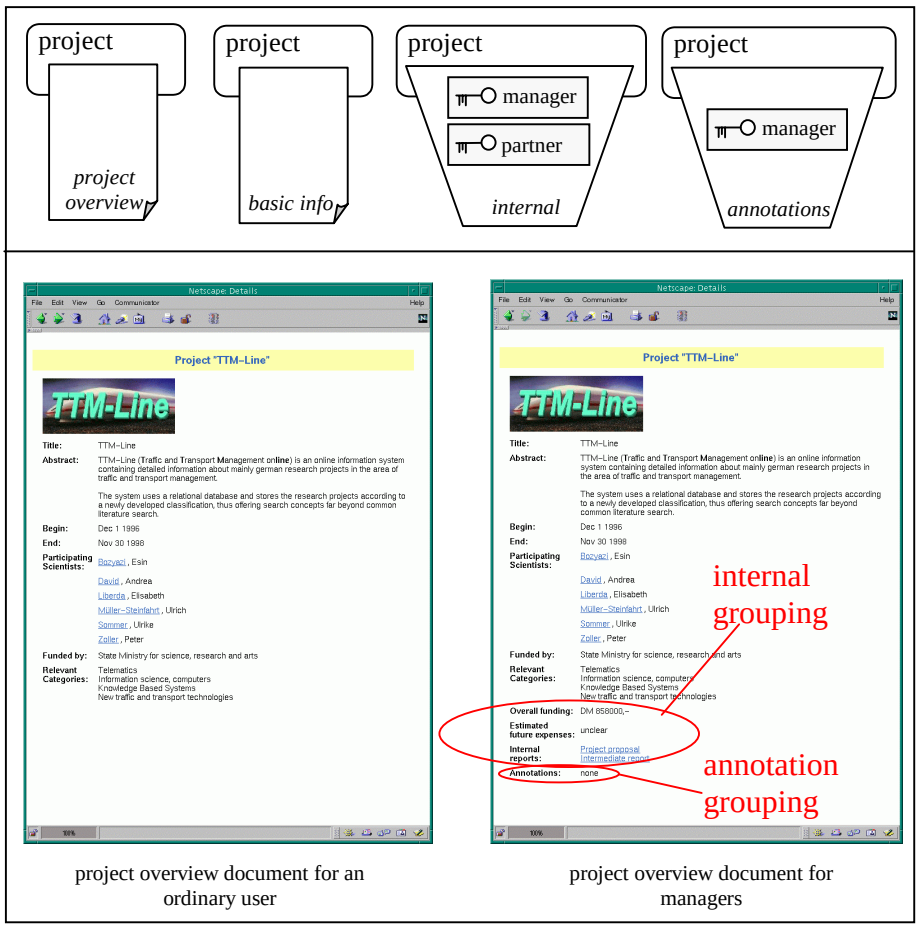

Figure 4-24: adaptive version of the project overview document

4.4.3. Summary

In this section, we have described the concept of role based access control (RBAC) and some advanced features like role hierarchies or separation of duties. We have defined the concepts required for the authorization design step of HMT and have presented the basic RBAC model used.

RBAC can be applied on two levels in HMT: First, restricting access to whole document types equals the traditional access control mechanism in www applications. Only users with the appropriate permissions can access restricted documents. Second, access restrictions on element-grouping level allow to model adaptive documents, where unrestricted elements are visible to all users, but restricted ones are only presented to the users with the appropriate rights. The usage of adaptive document types eases the generation and administration of hypermedia applications, because only one document type has to be defined in order to serve as a template for a variety of presentations for different types of users.

4.5 Logical Hypermedia Design

Until now, we have designed information clusters, navigational structures and access restrictions with the Hypermedia Modeling Technique. But we still need further information before the application can be built: The elements of every document type and element-grouping have to be ordered for presentation, and most of them should be assigned some kind of label describing their contents. These contents can be displayed in many different ways depending on the type of the document, for example as a popup menu in query documents or as an email link in detail documents. Additionally, true hypermedia applications consider temporal aspects and might want to order a document's elements in a timely manner. And last but not least, some kinds of document types need some specific information like, for example, the number of entries displayed on a single result document. All this information is specified during what we call the Logical Hypermedia Design step in HMT.

A logical HMT diagram consists of a document type or element-grouping containing a list of all its elements using the following format:

<position><label><element><additional information>

The different fields of this specification will be described in the following subsections. Special features like the number of entries on a result document are specified after the element list. For the temporal design, a separate box drawn with a dotted line is attached to the document type or element-grouping containing the temporal specification as described in section 4.5.4.

4.5.1. Order and labels of elements

Independent of how the elements of a document type or element-grouping are presented, they can be assigned some kind of order in which they are to appear. This order determines an element's position compared to the other elements, but does not specify a precise location within the application window, because this would require a decision for a certain hypermedia system in order to use the appropriate syntax and techniques. By deferring this decision to the layout design step, the logical hypermedia design of HMT is still completely independent of the hypermedia system used. The scope of an element's position is always limited to the containing document or element-grouping that means it's a relative position. This allows reusing element-groupings instead of defining identical ones with different absolute element positions.

Considering attributes like *phone_number* or relationship references like *participates_in_project*, it is obvious that often the pure contents of elements are not self-describing. This means that labels have to be attached to the elements describing the semantics of the values displayed. For example, on the projects overview document of our sample scenario, the list of participating scientists is labeled "*participating scientists*", because otherwise the semantics of the names listed there would not be clear to the user.

Of course each attribute has a name usually describing its content, but very often abbreviations are used or the name is not expressive enough for the users. In addition to this, not only attributes or access structs, but also entire element-groupings might require some hints regarding their semantics. For example, a personal homepage may contain an element-grouping containing the business address (*street*, *number*, *city*, *zip-code*, *phone* etc.) and an element-grouping containing the private address (again *street*, *number*, *city*, *zip-code*, *phone* etc). Assigning a unique label to each attribute ("*street (private)*", "*street (office)*" and so on) is not satisfactory, assigning a label to each element-grouping ("*private address*", "*business address*") is the better solution.

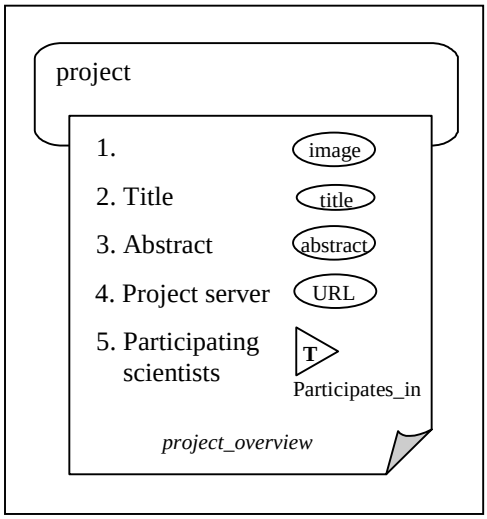

Figure 4-25: Logical HMT diagram of the project overview document type

Figure 4-25 shows the logical HMT diagram of the (non-adaptive) project overview document of our sample scenario (without the meta types, see next section for information on that issue). The first attribute image has no label, because the project's logo does not have to be described.

But labels are not just used for describing the contents of attributes or internal access structs. If external access structures have been defined during the conceptual hypermedia design, the labels will be used as the link text for the hyperlinks pointing to the corresponding documents. In any case, a label's scope is limited to the containing document type or element-grouping, because the corresponding element may be used by different documents or element-groupings requiring different labels.

4.5.2. Meta types

Until now, we have defined which attributes are to be displayed by a document or element-grouping and have specified their labels and position, but we have not described **how** the contents of these attributes are to be presented. In a project overview document, for example, the content of the *URL* attribute could be displayed as plain text or as a hyperlink pointing to the location specified by its content. In a project query document, the attribute *title* could get a simple input line or a popup menu containing the titles of all projects for selection.

In HMT, **meta types** are used to specify the presentation of attributes. Similar to the specification of an element's position, meta types are independent of the hypermedia system used, because they do not define real code for the presentation. This again is deferred to the layout design step.

For each attribute, a set of possible meta types is available depending on both logical representation of the attribute inside the database (the database type) and the type of document this attribute belongs to. As we have seen in the example described above, different meta types are necessary for different kinds of document types, for example a meta type *URL* for detail documents and a meta type *popup* for query documents. Concerning the attributes' database type, we distinguish five types:

- **DB_CHAR** for any kind of character-based types like Char, Char(*), String and so on
- **DB_NUMERIC** for any kind of numeric types like Integer, Short, Byte, Float and so on
- **DB_DATE** for any kind of date or time types
- **DB_BOOL** for boolean types
- **DB_BLOB** for BLOBs

The following tables describe the standard HMT meta types depending on the document type and the attribute's database type (also referred to as db type).

Meta types for query document types:

Meta type	Representation	Available for DB types
Char	Textual input line	DB_CHAR
Numeric	Numeric input line with three buttons for selecting the comparison operator (´=´,´<=´,´>=´).	DB_NUMERIC
Popup	Popup menu consisting of all values of the attribute in the underlying data source	any type
Multi	Popup menu allowing multiple selections and two buttons for choosing their logical connection are displayed	any type
Buttons	A list of radio buttons consisting of all values of the attribute in the underlying data source	any type
Bool	Three radio buttons labeled *yes*, *no* and *don´t care* .	DB_BOOL
Datetime	Numeric date input fields (year, month, day, hour ...)	DB_DATE

Meta types for result and detail document types:

Meta type	Representation	Available for DB types
Plain	Plain text (not formatted).	Any type
Email	Email link with attribute's content used for anchor and target.	DB_CHAR
Url	Hyperlink with the attribute's content as anchor and target.	DB_CHAR
Bool	*Yes* or *no* depending on the attribute's value	DB_BOOL
Date	String representing the date/time value	DB_DATE
Image	The content of the corresponding attribute is interpreted as an image and presented within the current document.	DB_BLOB
Download	A link is provided the selection of which transfers the attribute's content to the user as raw data.	DB_BLOB

Meta types for input document types:

Meta type	Representation	Available for DB types
Numeric	Numeric input line.	DB_NUMERIC
Auto_num	Nothing. Used for automatic sequence generation.	DB_NUMERIC
Text	Textual input line.	DB_CHAR
Textarea	Textarea (input field with several lines)	DB_CHAR
Datetime	Numeric date input input fields	DB_DATE
Current_date	The current date	DB_DATE
Popup	Popup menu containing all current values of this attribute in the underlying data source.	Any type
Bool	Three radio buttons (*TRUE, FALSE* and DON'T CARE).	DB_BOOL
File	File upload form.	DB_BLOB

Figure 4-26 shows the completed logical specification of the project overview document type now containing the meta type for each attribute.

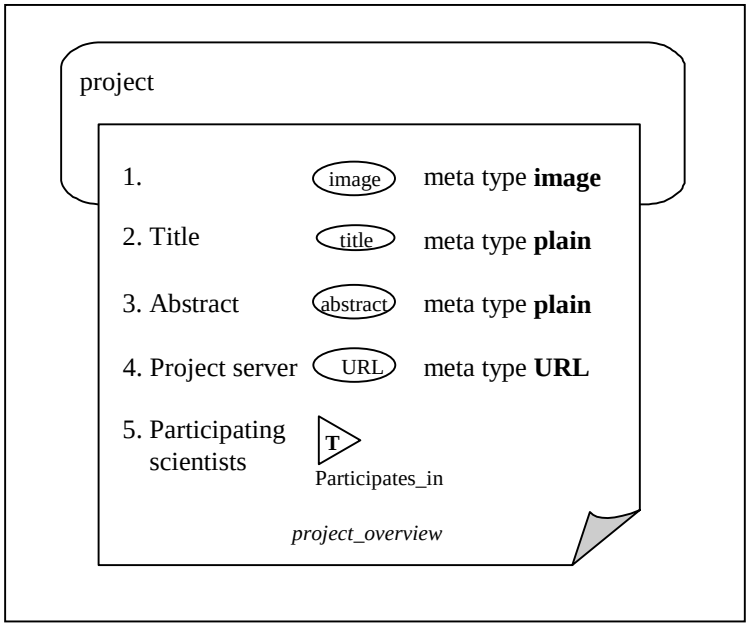

Figure 4-26: logical HMT diagramm of the project overview document type

4.5.3. Special features

In addition to the aspects already discussed, there are some other issues belonging to the logical design step which only apply to specific kinds of document types or to specific elements like submitters or links. The following subsections will present these special features sorted according to the kind of document type.

4.5.3.1 Query document types

Usually, labels of attributes in documents are only used to explain the semantics of the information displayed like *project title* or *participating scientists*.
In query document types, labels can have an additional meaning: if two or more different attributes have identical labels, then only one common input field will be displayed. If the user specifies a restriction on that field, this restriction will be matched with all attributes having this identical label using the *or* operator for combination.
For example, the projects query document type named *search* from Figure 4-10 contains the attributes *title* and *abstract*. If both have the identical label *keywords*, then the user will see only one input field labeled *keywords*. If he specifies a search string in this input field, all entries will be selected where either the title or the abstract of the project (or both) contain this search string.
This kind of unstructured search (on various attributes) is typical for a lot of hypermedia information systems especially in the World Wide Web, and most users are familiar with it because of experiences with common search engines [Alt, Lyc00, Yah00]. Using identical labels to provide an interface for this kind of unstructured search on structured information is a useful feature of HMT query document types.

4.5.3.2 Result document types

In result documents, there should always be a limit for the number of items displayed on a single page, because the user's queries are not predictable and the corresponding result sets can grow very large. To avoid frustrating users with long download times and information overload, splitting the result set into several documents connected with hyperlinks is a common technique, which can easily be combined with query refinement strategies. Figure 4-27 shows the logical specification of the project result document type of Figure 4-13.

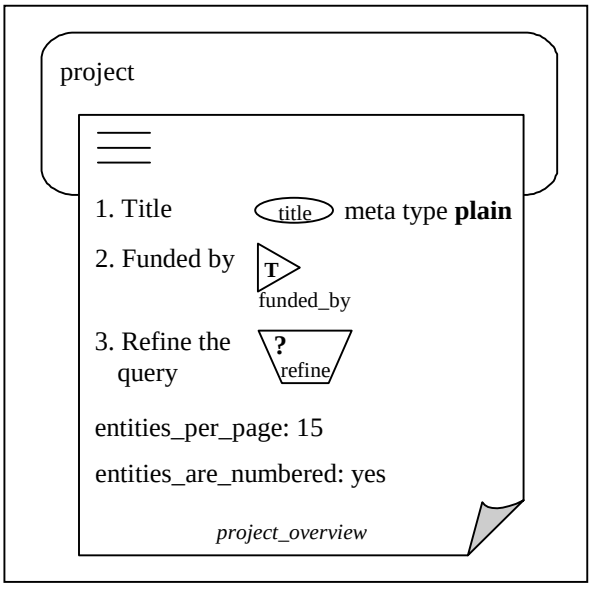

Figure 4-27: logical HMT diagram of a project result document type

The variable *entities_per_page* is defined after the logical specification of the result document type's elements and is set to 15 in our example. If more than *entities_per_page* entries are found, links on the result document pointing to the next or previous page are generated automatically. The specification of these links (they could be, for example, textual hyperlinks or images) is done during the layout design step of HMT.
Numbering the items in a result document often helps the user to keep an overview when result sets grow very large. The boolean variable *entities_are_numbered* indicates whether results are numbered or not.

4.5.3.3 Input document types

For input documents, it is necessary to have some additional logical features in order to be able to specify powerful, flexible and user-friendly data manipulation interfaces. Often it is required that a certain attribute has to be assigned a value if the user did not specify one himself. In HMT, this can be done by defining a default value for a given attribute. This default value is specified after the attribute's meta type. In a similar fashion, it is also possible to define a default value for access structs.

In some cases, however, it may be necessary to specify a default value for an attribute or access struct without allowing the user to change or even see this selection. This can be achieved by not specifying a label for this attribute. Each attribute without label is not displayed, but will be assigned the default value specified or NULL if no default value is defined. A typical example for this is the numeric primary key of an entity type, which shall be created automatically for new entries by assigning it the *auto_num* meta type. This key does not need to be visible to the user, but has to be calculated for every new entry.

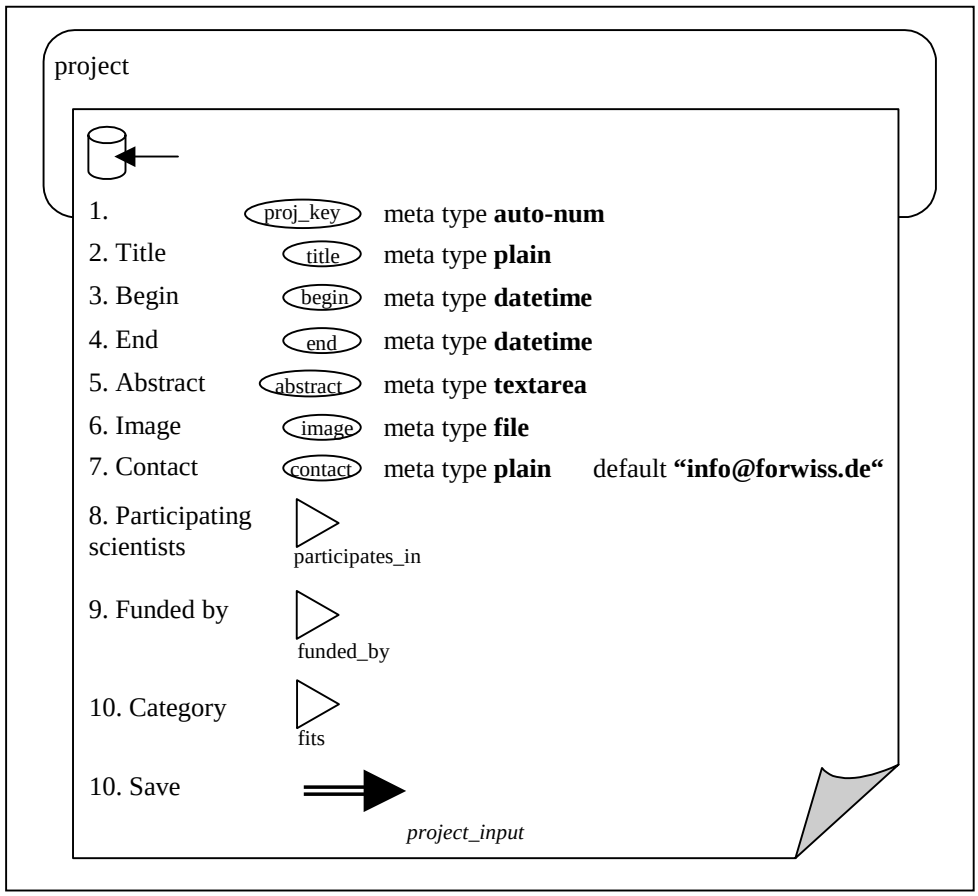

Figure 4-28: Logical HMT schema of a project input document

Figure 4-28 shows the logical specification of a modified version of the projects input document from Figure 4-16. In contrast to the original example, we now assume that the project entity has a numeric key (the attribute *proj_key*) and an additional *contact* attribute.

4.5.3.4 Links and form submitters

The link primitive as described until now allows navigating to a different HMT document or some kind of external hypermedia presentation. If selected by the user, the target of this link replaces the source document. In a similar way, the usage of a form submitter causes the original document to disappear and the target document is displayed instead.
In some cases, however, it might be desirable to invoke an additional presentation unit for the target of a link or form submitter and to keep the original document visible. A typical example for such a behavior is the world wide web, where often the selection of a hyperlink causes a new browser window to appear, leaving the original document still available for the user.

HMT allows specifying the target of a link or form submitter by providing a **context** string in the logical HMT schema as shown in Figure 4-29. The default value for the context is *main*, which references the initial presentation unit. Each new context string specifies exactly one presentation unit, which can be referenced by any other link or form submitter.

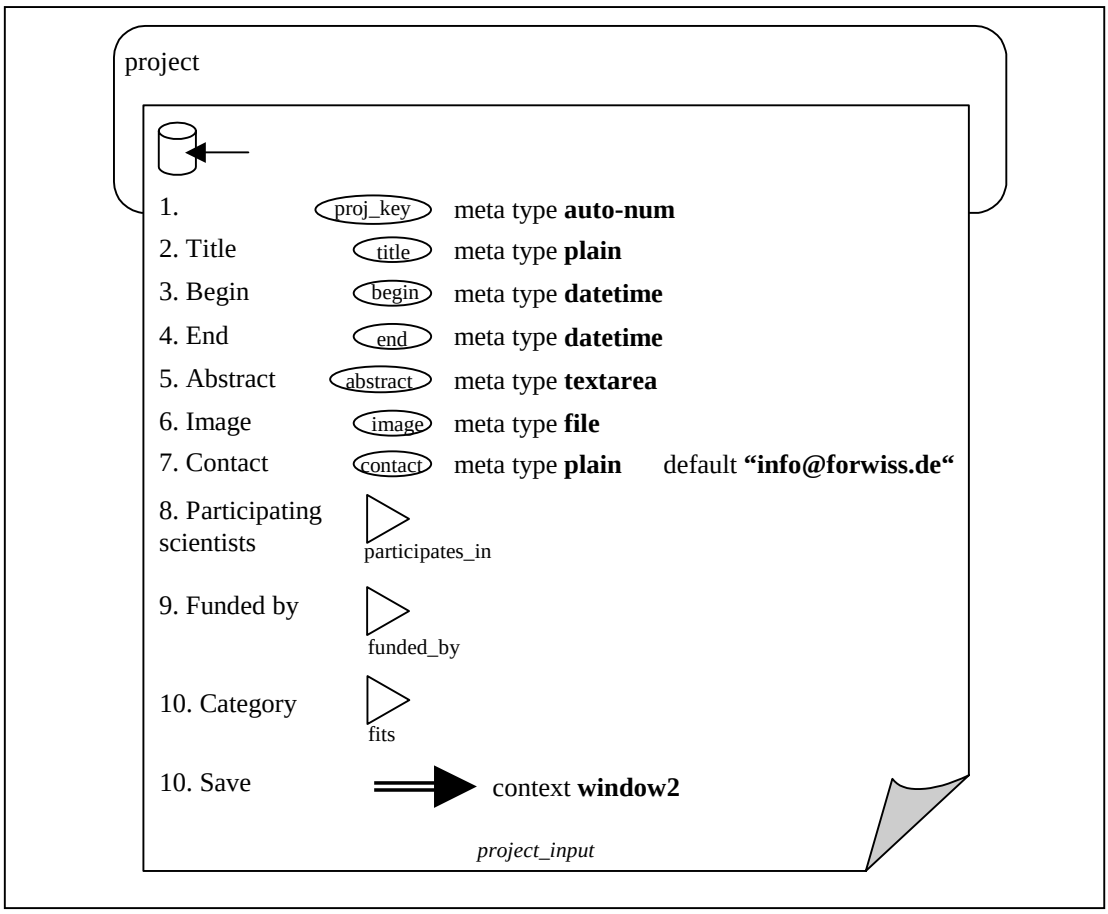

Figure 4-29: logical HMT schema with submitter context

4.5.4. Temporal design

The temporal specification of a hypermedia presentation is an important aspect that should be supported by hypermedia modeling methodologies. Not only the duration or sequential execution of the presentation's elements, but also various kinds of interdependencies between the different elements have to be specified
Former approaches were often based on scripting languages, for example Lingo [MMD]. Although very powerful, these approaches are rather inflexible concerning changes within the presentation, and interdependencies between different elements are often not obvious from the coded specification. Besides, programming skills are required which restricts the number of users for such a methodology.
Another classical approach is to use a timeline and to specify absolute positions for each element. This very intuitive technique is rather inflexible regarding changes (all dates "behind" the changed one have to be recalculated), and elements with infinite duration cannot be modeled with this approach, because all dates are absolute points of time.

Newer concepts therefore do not rely on exact points of time, but allow specifying temporal relationships between the different elements of hypermedia applications. Based on the work of Allen [All83], several approaches for modeling temporal relations have been proposed [Jou97, Hos97, Yu97, RJM+93, SDK96].
The temporal design concept of HMT is based on these works and has been tailored to the special features of its conceptual model. HMT allows specifying the time each design primitive of a document or grouping is active (that means visible). The scope of a temporal specification is always the surrounding HMT element-grouping or document, which eases the task of synchronizing complex parts of a presentation.

Section 4.5.4.1 introduces the temporal design primitives, section 4.5.4.2 describes the specification of temporal relations between the elements, and section 4.5.4.3 shows the temporal specification of a modified version of our sample scenario. A summary is given in section 4.5.4.4.

4.5.4.1 Temporal Design Primitives

In order to specify aspects of time, the conceptual HMT design primitives are extended by some temporal design primitives. Each document or element-grouping defines its own local presentation, which is specified within a dashed box attached to the document or element-grouping.
Within this dashed box, the start of a presentation is marked with a special symbol, which is connected to the first element of the presentation.
Each element is represented by a rectangle containing the element's name. The length of the rectangle is primarily determined by the length of its name and is no indicator for the element's absolute duration. However, relations between elements are to some extent symbolized by the relative position of the elements against each other (for example "element a includes element b", see next section).

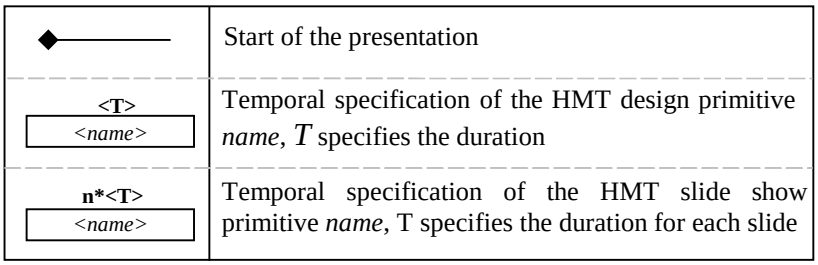

Figure 4-30: temporal design primitives

If the absolute duration of an element has to be specified, the corresponding period of time *T* is written above the rectangle. Two cases have to be differentiated:

- If timeless elements like text or images are concerned, the effect is trivial. These elements are simply displayed for the specified period of time and then vanish.
- Time-based components like audio or video elements already have a predefined duration T_P. If they are assigned a certain time of duration T_S, (specified duration) these elements are either cut off if $T_P > T_S$ or restarted if $T_P < T_S$.

Special kinds of element are Slide Shows, because for them a temporal specification is mandatory. Three different approaches are possible:

- If an overall duration *T* is specified, the period of time *t* used to present a single element of the Slide Show is calculated as t = T/n, where n is the number of elements of the Slide Show. This ensures that a given presentation doesn't have to be changed if the number of elements of the Slide Show changes. For example, a company's homepage may consist of a welcome audio track and a Slide Show where the main products are shown, and the Slide Show has to finish together with the audio track. If new products are added to the Slide Show, the synchronization of the two elements doesn't have to be changed.
- Alternatively, it is also possible to specify the period of time *T* used for a single element of the Slide Show by using the expression n**T*. In this case, changes to the number of elements of a Slide Show influence the overall duration and thus have effects on the synchronization with other elements.
- If all elements of the Slide Show are time-based (like video or audio), then no explicit specification of *T* is necessary, because each element has its own duration.

Since each element of a document or grouping has its own temporal representation, also links and access structures can be temporarily restricted and synchronized with other elements, for example video clips or audio tracks.

4.5.4.2 Temporal Relations

For interval-based approaches, Allen [All83] has identified 13 characteristic relations, which can be reduced to 7 if the inverse relations are derived by swapping the corresponding elements. HMT adopts and extends these relations:

First, a new relation *synchronizes* is introduced which is especially suited for time-based elements like video or Slide Shows. Second, a simple arrow denotes a delayed sequential execution of two or more elements, whereas an arrow originating from a bullet is used for specifying parallel execution. The element where the arrows originate from are called *synchronizing* elements, the arrows' targets are the *synchronized* elements. Third, the *equals* relation is left out, because it can be replaced by a combination of the *starts* and the *finishes* relation.

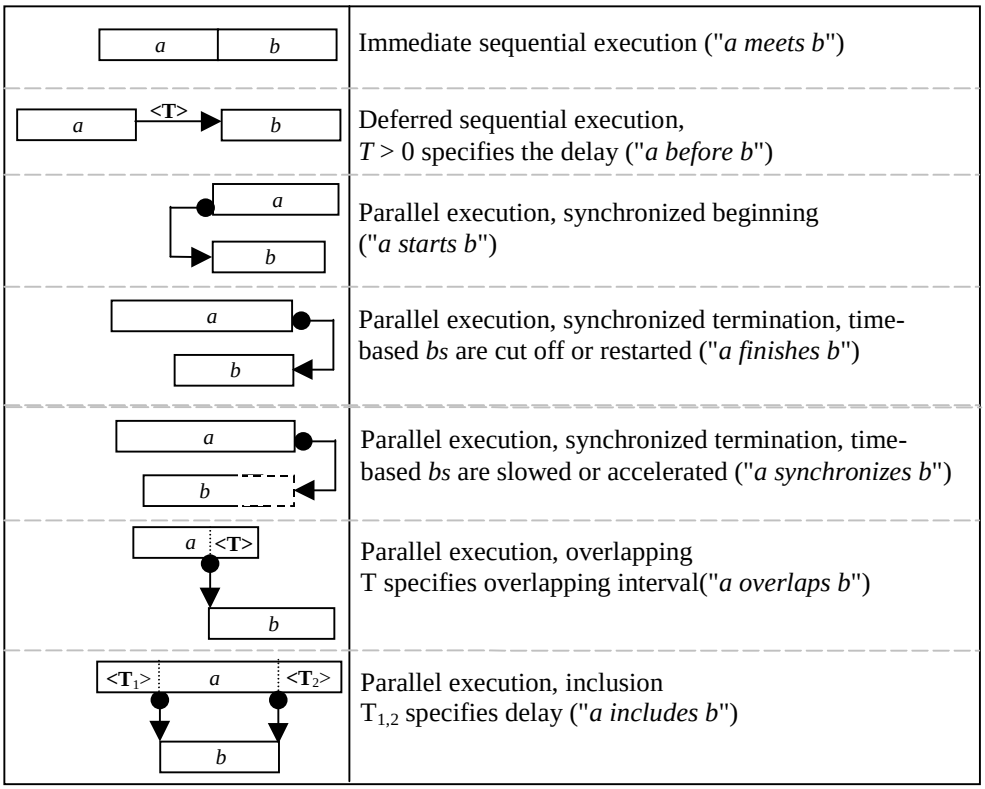

Figure 4-31: temporal relations in HMT

Figure 4-31shows an overview of the temporal relations used in HMT.

For a detailed description of the temporal relations in HMT, we use the following definition:

Definition 4.35 (duration of an element):

Let $H = GR \cup ADDS \cup ATTR \cup ACC \cup LINKS$ be the set of HMT elements, and $x \in H$, then $T(x)$ denotes the **overall duration of x** (either explicitly specified or implicitly defined in case of time-based elements), *start(x)* and *end(x)* define the **starting point** and the **end point** of *x*.

We like to stress that *start(x)* and *end(x)* are abstract measures and are only used to define the semantics of the temporal relations, they are not part of the temporal model of HMT.

The relation **a meets b** defines an immediate sequential execution, where *b* is started after *a* has finished. Prerequisite for this relation is that *a* is either a time based element (thus having a predefined duration), or explicitly restricted by specifying *T(a)*.

Definition 4.36 (temporal relation *meets*):

Let $H = GR \cup ADDS \cup ATTR \cup ACC \cup LINKS$ be the set of HMT elements, then the temporal relation **a meets b** is defined as $\{ (a,b) \mid a, b \in H$ and $start(b) := end(a) \}$.

Similar to the previous case, in relation **a before b** the element *b* is started after *a* has finished. Instead of an immediate execution, the start of *b* is delayed by *T*. Element *a* must have a finite duration *T(a)*.

Definition 4.37 (temporal relation *before*):

Let $H = GR \cup ADDS \cup ATTR \cup ACC \cup LINKS$ be the set of HMT elements and $T > 0$ be a period of time, then the temporal relation **a before b** is defined as $\{ (a,b) \mid a, b \in H$ and $start(b) := end(a) + T \}$.

In the following relation, the elements *a* and *b* are started simultaneously and are executed in parallel. The starting point is determined by element *a*.

Definition 4.38 (temporal relation *starts*):

Let $H = GR \cup ADDS \cup ATTR \cup ACC \cup LINKS$ be the set of HMT elements, then the temporal relation **a starts b** is defined as $\{ (a,b) \mid a, b \in H$ and $start(b) := start(a) \}$.

The relation **a finishes b** synchronizes two elements *a* and *b* in the way that the end of *a* causes *b* to terminate. Prerequisite for this case is that *a* has a finite duration (either specified or predefined by time-based elements).

Definition 4.39 (temporal relation *finishes*):

Let $H = GR \cup ADDS \cup ATTR \cup ACC \cup LINKS$ be the set of HMT elements, then the temporal relation **a finishes b** is defined as $\{ (a,b) \mid a, b \in H$ and $end(b) := end(a) \}$.

While the synchronization of timeless elements *b* is trivial, time-based elements have to be treated differently: If *b* has been started by some other element and would normally end before *a* has finished, then *b* has to be restarted in order to bridge the gap until *a* terminates.

Constraint 4.1:

Let $a, b \in H$ be HMT elements and *a* finishes *b*, then the presentation of *b* has to be restarted if $T(b) < T(a) - (start(b) - start(a))$.

On the other hand, if *b* would normally end after *a* has finished, *b* has to be stopped:

Constraint 4.2:

Let $a, b \in H$ be HMT elements and *a* finishes *b*, then the presentation of *b* has to be stopped if $T(b) > T(a) - (start(b) - start(a))$.

For timeless elements, the relation **a synchronizes b** has the same effect as *a finishes b*. But if *b* is a time-based element and has been initiated by some other element *x*, then *b* is slowed or accelerated to match the period of time needed for synchronization with *a*:

Definition 4.40 (temporal relation *synchronizes*):

Let $H = GR \cup ADDS \cup ATTR \cup ACC \cup LINKS$ be the set of HMT elements, then the temporal relation **a synchronizes b** is defined as
$\{ (a,b) \mid a, b \in H$ and $T(b) := T(a) - (start(b) - start(a)) \}$.

For example, as Slide Show *b* has been initiated by a third element *x* and has to be synchronized with an audio track *a* in the way that the Slide Show is only run once and finishes together with *a*. The speed of the Slide Show *b* is automatically adapted to match the necessary period of time.
Of course, the same effect could be reached by defining the times T_i for all relevant elements and then calculating *T(b)* manually, but with that approach every change to a previous element would require a recalculation of *T(b)* and possibly following elements.

The relation **a overlaps b** describes a partially parallel execution of *a* and *b*, where *T* specifies the overlapping interval. Element *a* must have a finite duration *T(a)*.

Definition 4.41 (temporal relation *overlaps*):

Let $H = GR \cup ADDS \cup ATTR \cup ACC \cup LINKS$ be the set of HMT elements and $T > 0$ be a period of time, then the temporal relation **a overlaps b** is defined as
$\{ (a,b) \mid a, b \in H$ and $start(b) := end(a) - T \}$.

In the following relation **a includes b**, the elements *a* and *b* are executed in parallel, and *a* starts before and ends after *b*. Element *b* must have a finite duration, and element *a* may be infinite unless T_2 is specified.

Definition 4.42 (temporal relation *includes*):

Let $H = GR \cup ADDS \cup ATTR \cup ACC \cup LINKS$ be the set of HMT elements and $T_1, T_2 > 0$ be periods of time, then the temporal relation **a includes b** is defined as
$\{ (a,b) \mid a, b \in H$ and $start(b) := start(a) + T_1$, $end(b) := end(a) - T_2 \}$.

4.5.4.3 Sample Application

We use a modified version of the project overview document as an example for the HMT temporal design step. We want to design a project overview document where first a short audio track is played, for example naming the project title with some background music. While the audio track is being played, a Slide Show displaying the names and portraits of all participating scientists should be shown. Two seconds after this short “intro”, the ordinary project overview document with all its attributes and access structs shall be presented.

Figure 4-32 shows the conceptual schema of this extended project overview document together with its temporal specification.

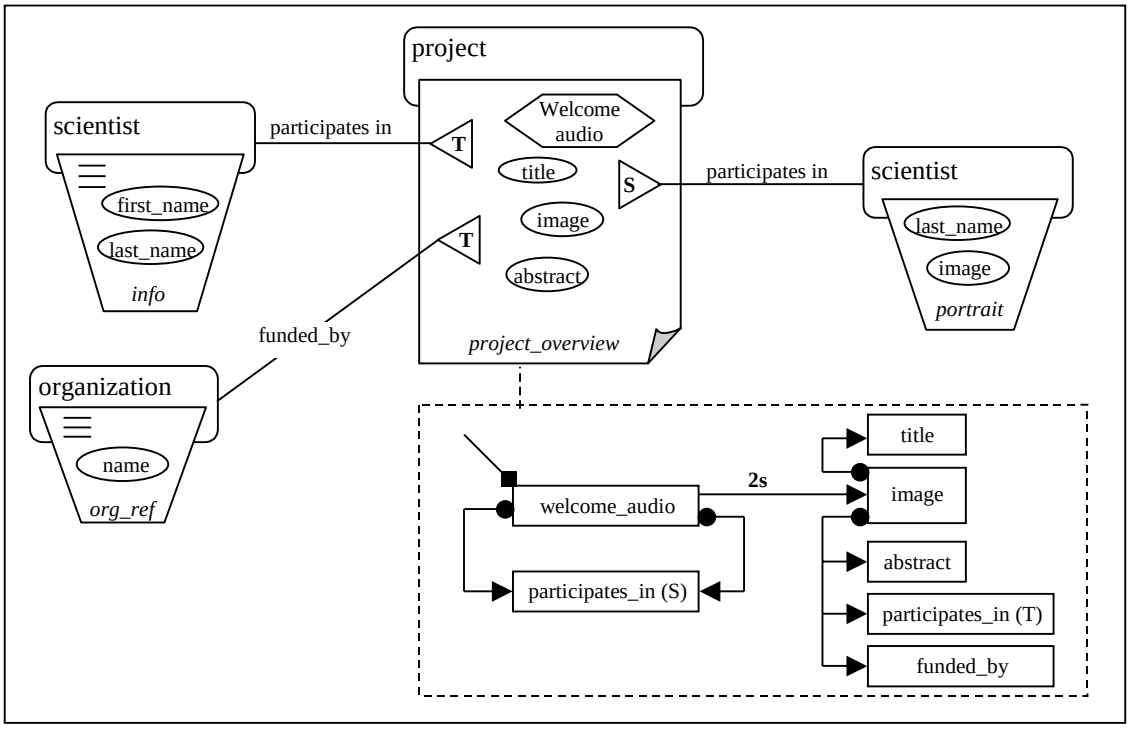

Figure 4-32: temporal specification of the project overview document

4.5.4.4 Summary

In this subsection we have presented the HMT temporal design step. Consistent with the other HMT design steps, a graphical interface is provided for the specification of time related issues.

Each element of a document type can be assigned its own temporal specification, which is relative to the surrounding element-grouping or document type. This counts for attributes and adds as well as for links or access structs. Each element can be assigned a duration defining how long the element will be displayed. In order to synchronize the various elements of a document type, HMT offers seven temporal relations covering both sequential and parallel presentation. The temporal specification of a document or element-grouping is attached to its conceptual HMT schema.

4.6 Layout Design

This final step in the process of defining a hypermedia application is the one supported least by most methodologies, because layout design is hard to formalize and cannot be captured in its entirety. It depends to a great extent on the actual hypermedia platform and its tools and programming languages.

Because of the lack of a general formal specification for layout design, HMT does not provide explicit support for this aspect, but leaves this task to the designer of a HMT CASE-tool. A prototype of such a CASE-tool is described in Chapter 6. There are, however, some issues regarding general or technical aspects of layout design that can be mentioned.

- Each document may be assigned a title, a background image or a background color, basic font properties and colors for text and links.
- Global headers and footers containing general formatting instructions can be useful.
- The layout of the presentation of an attribute's content is defined by specifying the mapping of its meta type to the corresponding hypermedia system (for example WWW/HTML).

If more advanced layout specifications are needed, arbitrary layout instructions can be inserted into a document by using adds as described in the conceptual hypermedia design step. However, by choosing this option, the code of these adds will be platform dependent and therefore has to be changed if the underlying hypermedia system changes.

4.7 Summary

The Hypermedia Modeling Technique (HMT) has been developed for designing interactive and adaptive, database-driven hypermedia applications. It consists of six design steps: requirements analysis, ER design, conceptual hypermedia design, authorization design, logical design and layout design.

Requirements analysis covers aspects like the definition of the application domain, identification of the intended users, and specification of the system's functionality and usage. Numerous publications can give further information on this issue, for example [Wei82, KK92].

After the application domain has been specified, an ER schema has to be built reflecting objects and relationships from the real world application domain. HMT relies on the basic ER model as described by Chen [Che76]. If a hypermedia application has to be built upon an already existing database, these first two steps in the HMT design process are omitted.

The core hypermedia application design starts with the conceptual hypermedia design of the application, based on the ER schema developed in the previous step. Aspects like information clustering and navigation are addressed within this design step.

Access restrictions are specified during the authorization design phase using role based access control Based upon the conceptual hypermedia schema, access to certain parts of the application can be limited by specifying roles required for viewing these components. This

can be done on document level or on element-grouping level: The first equals the traditional access control mechanism in the world wide web, the second leads to adaptive documents.

During the logical design phase, additional properties concerning the logical representation of a document's content are defined. This covers the order and labels of a document's elements, the meta types of the attributes, some document specific features like the number of items on result documents, and the temporal design of a document or element-grouping.

Finally, all aspects regarding the layout of the later presentation are covered by the layout design step. Since this step heavily depends on the hypermedia system used, HMT does not specify an interface for this task, but leaves this to the HMT CASE-tool used. A prototype of such a tool will be described in Chapter 6.

CHAPTER 5

THE HMT META SCHEMA

One motivation for the creation of modeling methodologies like HMT is to provide an intuitive and consistent graphical notation for the specification of applications. This allows to present concepts and ideas at a very early stage of the design process and in a way understandable also for people with only little information technology experience. At the same time, it is a precise specification of the application, which can be used by the expert implementing the system.

A second important motivation for the development of modeling methodologies is the ability to provide a set of development tools, which allow building and maintaining these graphical application schemas in electronic form instead of drawing them manually on a sheet of paper. One prominent example for such a tool is Rational Rose [Ro00] from Rational Software Corporation, which is used for building and maintaining UML diagrams.
On demand, these tools can automatically produce portions of code needed for the implementation of the application. In order to accomplish this task, the information contained within the graphical schemas has to be converted into a format suitable for further processing by the development tools. This is usually either some kind of (database) repository or a specification language, both of which have advantages and disadvantages:

- Specification languages provide the schema information in a human readable way and can easily be modified with any text editor, but require their own compiler and lack a standardized interface. Queries on the schema (for example “List all documents being targets of navigational links”) are not possible unless an algorithm for extracting this information from the specification files is implemented.
- Database repositories store the information in a set of meta tables, which cannot easily be maintained without the corresponding administration tool, but offer a standardized and well known interface (e.g. SQL) together with all the benefits of a DBMS (e.g. the ACID principles or logical data independence). The application schema can directly be queried using SQL, which even allows executing complex operations on the meta schema without using the administration tool. Moreover, many of the integrity constraints needed in order to guarantee schema consistency could automatically be assured by the DBMS if the meta schema is specified accordingly.

Since HMT has been developed for designing database-driven hypermedia systems, a DBMS is already provided by the application. Storing both application data and hypermedia presentation within a DBMS enhances consistency and integrity of the whole application. Together with the advantage of having a standardized interface for storing, maintaining and

retrieving repository data, these considerations lead to the decision to map HMT schemas to a set of meta tables.
A detailed specification of the HMT meta model and the mapping of HMT design primitives is presented in section 5.1. Section 5.2 describes the algorithms needed for generating applications based on HMT meta schemas, and section 5.3 discusses the query generation process for HMT input and query documents. The chapter ends with a short summary in section 5.4.

5.1 Specification of the HMT meta model

During the HMT design process, it is useful to clearly separate the different design steps from each other in order to foster a structured, well defined and powerful hypermedia design. The HMT meta schema that stores information from all steps of the HMT design process has to accomplish a different task: The creation of a complete hypermedia document should consume as little time as possible, that means requests to the repository should be minimized.

Therefore, the HMT meta schema does not separate information according to design steps, but tries to group all information relevant for the creation of a document into as few entity types as possible. Since the size of the HMT meta schema is independent of the size of the application database and can usually be neglected, we decided to not normalize the meta schema, because this would reduce readability and require more repository requests in order to create an application.
For example, there exists only one entity type for all kinds of document types (general, query, result, detail and input) in our HMT meta schema. This means that some special attributes of the documents meta table (for example *itemsPerDocument*) are not used for all but one document type (in this case, the result document type). Although normalization could avoid these effects, maintenance and use of such a normalized HMT meta schema would be much more complicated and reduce the performance of the application.

For the remainder of this chapter, we will use the term *meta table* when talking about an entity type of the HMT meta schema. This helps to distinguish clearly between entity types of the application domain (for example *projects*) and entity types of the meta schema (for example *documents*).

5.1.1. Overview

The core HMT meta schema consists of 14 meta tables and about 40 relationship types storing all information of a HMT schema from the conceptual hypermedia design down to the layout design. Each conceptual design primitive is assigned its own meta table, and additional meta tables are provided for users, roles, temporal specifications and layout design. The meta tables provided for layout design purposes address only the basic requirements and are not bound to a specific hypermedia platform. For example, the layout of an attribute’s content is stored

within the *elementFormats* meta table, and it is up to the administrator to decide whether its content should be coded, for example, in HTML or XML format.

The meta schema described in this chapter can be considered as the minimal meta schema required, actual HMT implementations might use additional attributes or meta tables if necessary. As we will see in Chapter 6, especially additional meta tables addressing aspects of layout design or administration may be introduced by an implementation.

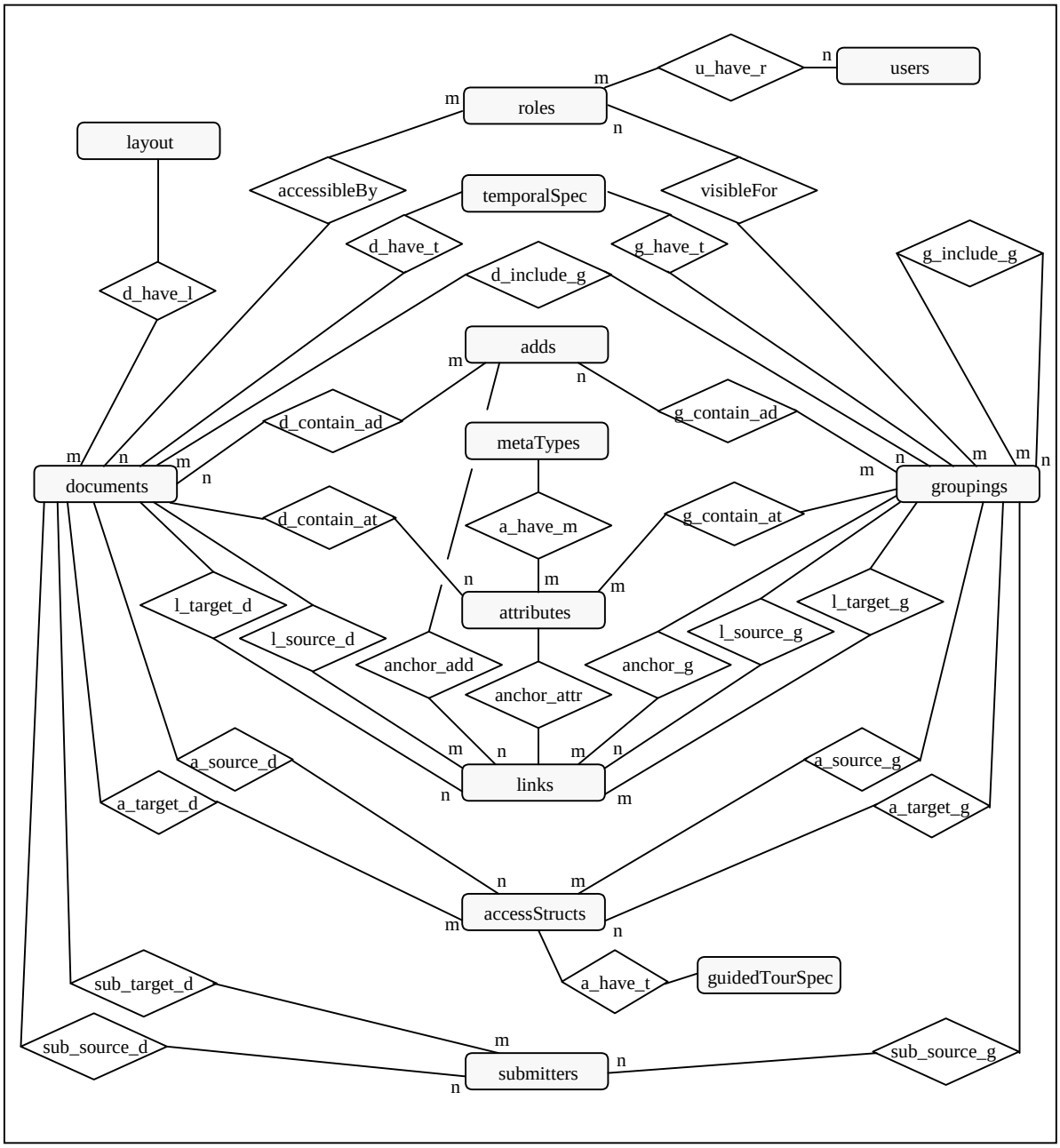

Figure 5-1: overview of the core HMT meta schema

Figure 5-1 shows the ER diagram of the core HMT meta schema. In order to improve readability of this and all following ER diagrams, we only specified relationship cardinalities explicitly if they are different from 1.

Documents and *groupings* are the central meta tables, accompanied by *attributes*, *adds*, *access structs*, *links* or *submitters*. The meta tables *users* and *roles* and the corresponding relationship types carry the information that has been generated during the authorization design step, *temporalSpec* contains the temporal specification of the document type or element-grouping. Not shown in this diagram is the *elementFormats* meta table, which contains information from the layout design step. Since this meta table is referenced by most other meta tables of the HMT meta schema, the diagram would be too complex and hard to read if it had been included in Figure 5-1. For the same reason, the *target* relationships for links are only connected to the documents and groupings meta tables, although any other element can also be the target of a link.

Each meta table has a numeric primary key and an identifier string (ID-string) to be used by an administration tool. The identifier string has deliberately not been chosen as the primary key, because this allows changing an entity's ID-string without generating a new entity and having to update all references accordingly. The following subsections describe the most important HMT meta tables in detail, separated into tables for HMT domain primitives (section 5.1.2), access primitives (section 5.1.3), layout (section 5.1.4), users/roles (section 5.1.5) and temporal specifications (section 5.1.6).

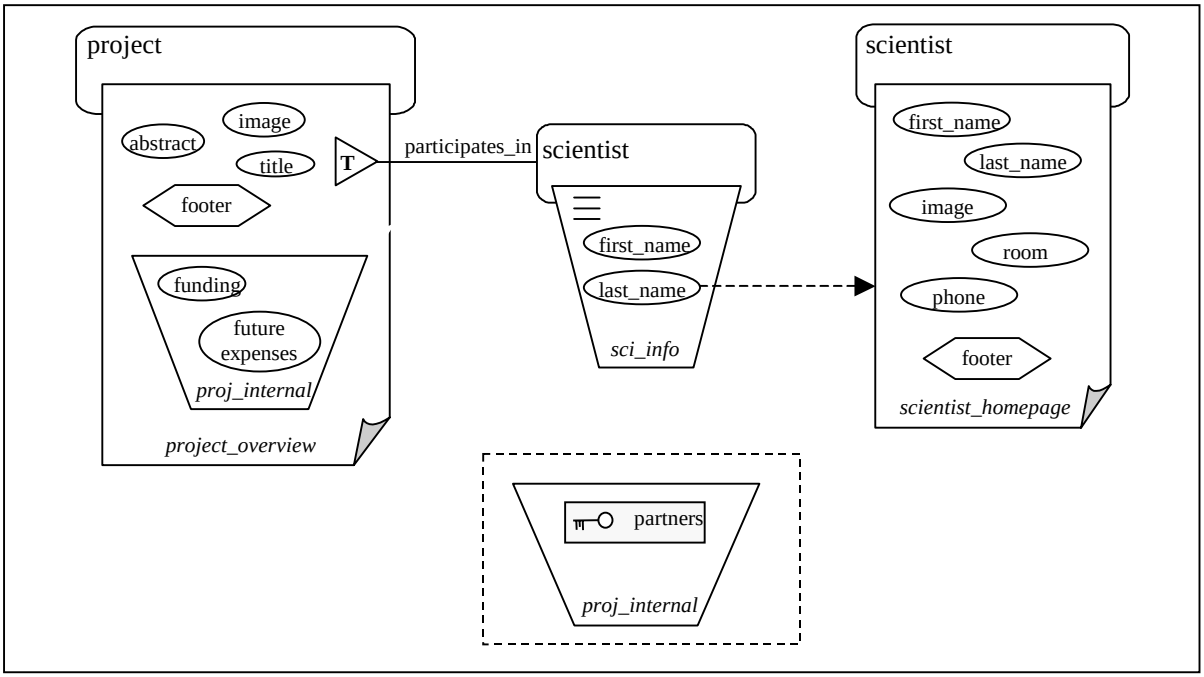

Figure 5-2: sample scenario for the mapping of HMT schemas to HMT meta tables

In order to be consistent throughout this subsection, we will use a graphical ER schema for each meta table presented, even if sometimes this ER schema is rather simple and might also

be replaced by a simple table. The mapping of HMT schemas to HMT meta tables will additionally be explained using the sample scenario specified in Figure 5-2. According to the standard approach for transforming ER schemas into relational schemas, m:n relationships will be mapped to separate tables.
The sample scenario contains a modified version of the *project_overview* document type described in Figure 4-14. Besides the freely accessible *title*, *image* and *abstract* attributes and the *footer* add, this document type also contains the *proj_internal* element-grouping, which is only visible for users with role *partner*. The internal access struct on scientists creates a list of all project members, where the *last_name* of each scientist acts as the anchor of a structural link pointing to the corresponding *scientist_homepage*.

5.1.2. Domain primitives

As specified in section 4.3.1, HMT offers four basic domain primitives: document types, element-groupings, attributes and adds. Each of them gets its own meta table in the HMT repository, which will be described in the following subsections.

5.1.2.1 Document types

The two central meta tables of the HMT meta schema are *documents* and *groupings*, which store information about all kinds of HMT document types and element-groupings.

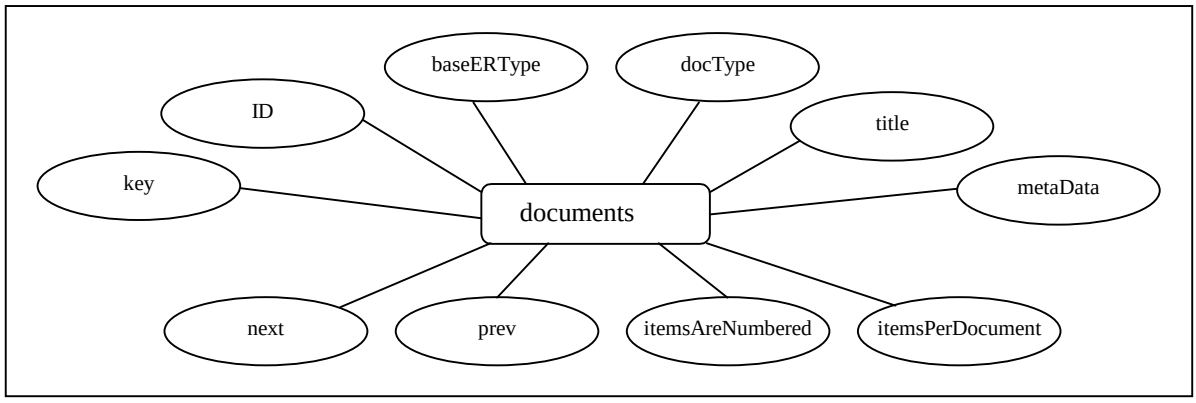

Figure 5-3: ER schema of the documents meta table

Each HMT document type is represented by one entity of the *documents* meta table. This table contains the name of the document type's base ER type (*baseERType*) or NULL if it is a general document. It also includes the type of the document (*docType*), which can be *general*, *query*, *result*, *detail* and *input*. Additional attributes are the official *title* (for example *project overview*, can be used to label the presentation window) and some meta information about the document (*metaData*) in an arbitrary format.
The attributes *itemsPerDocument*, *itemsAreNumbered*, *prev* and *next* are only valid for result document types. The first two specify the distribution of large result sets over several documents, *prev* and *next* provide means for navigating between the result documents, for example arrow buttons or some text for the automatic generation of the corresponding hyperlinks.

There are several relationship types between the *documents* meta table and other meta tables containing further information as described in Figure 5-4: Each document type has a certain *layout*, which includes information about the background color or background image of the document and general font specifications. A document type also references an entity of the *temporalSpec* meta table containing the temporal specification of a document type. This meta table will be is described in detail in section 5.1.6.

The main contents of a document are defined by the relationships with the *adds*, *attributes* and *groupings* meta tables. Each referenced entity of these meta tables has a *position*, *label* and a certain *elementFormat* within the referencing document type. The position in this context is not an exact specification of coordinates, but some kind of sequence number. When creating a hypermedia application from HMT schemas, the elements of a document type (for example attributes, links or element-groupings) will be sorted according to their position.
The *elementFormat* meta table contains layout information that is specified for the actual hypermedia system used, it will be described later in section 5.1.4. In order to be able to reuse adds and element-groupings, the information about *position*, *label* and *elementFormat* is stored within the corresponding relationship. Access restrictions on document types are specified by assigning an arbitrary number of roles, which are exclusively allowed to view this document. If no roles are assigned, the document will be freely available.

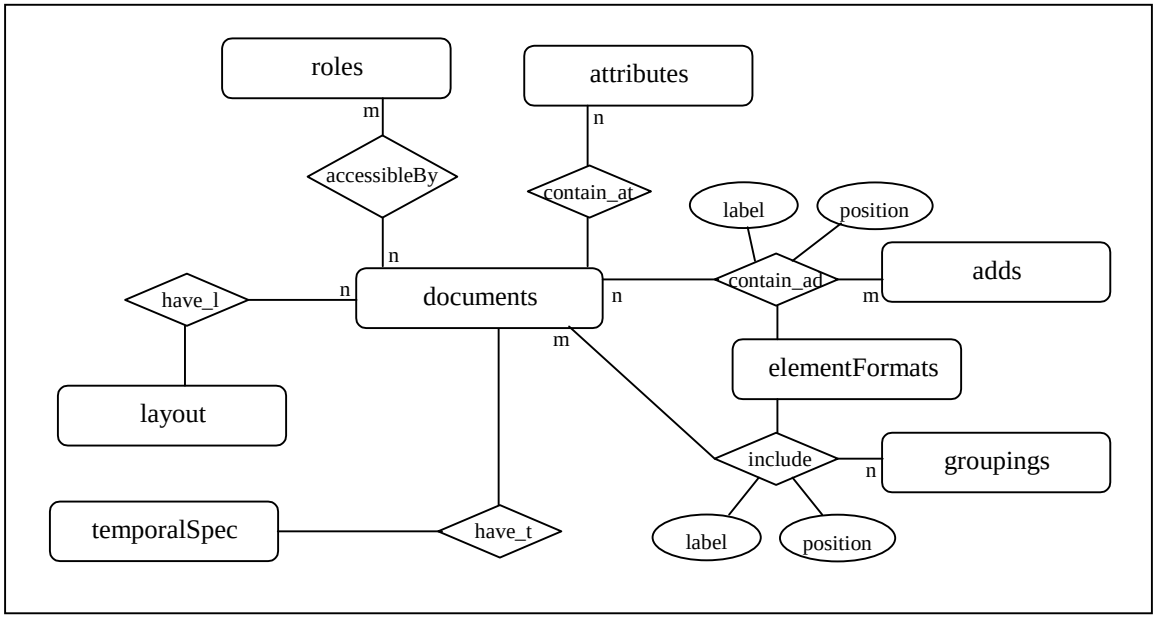

Figure 5-4: basic relationship types for the documents meta table

The mapping of the two detail document types *project_overview* and *scientist_home* to the *documents* meta table is shown in Table 5-1. Due to space limitations for the page format, we skipped the presentation of the attributes *itemsPerDocument*, *itemsAreNumbered*, *prev* and *next*. The mapping of the relationships with *attributes*, *adds*, *element-groupings* and *roles* will be specified in the corresponding subsections describing these meta tables.

documents meta table						
ID	key	docType	baseERType	Title	metaData	layout
project_overview	d17	detail	Project	Project information	author=zoller institution=FORWISS	3
scientist_homepage	d21	detail	Scientist	Homepage		3

Table 5-1: mapping the sample scenario to the documents meta table

5.1.2.2 Element-groupings

A HMT element-grouping is represented by an entity of the *groupings* meta table. Besides the key and the ID, this meta table contains information about the base ER type and the type of the element-grouping (i.e. *general*, *query*, *result*, *detail* and *input*).
In contrast to HMT document types, element-groupings are only parts of hypermedia documents and therefore do not need document-specific information like the document title or describing meta data.

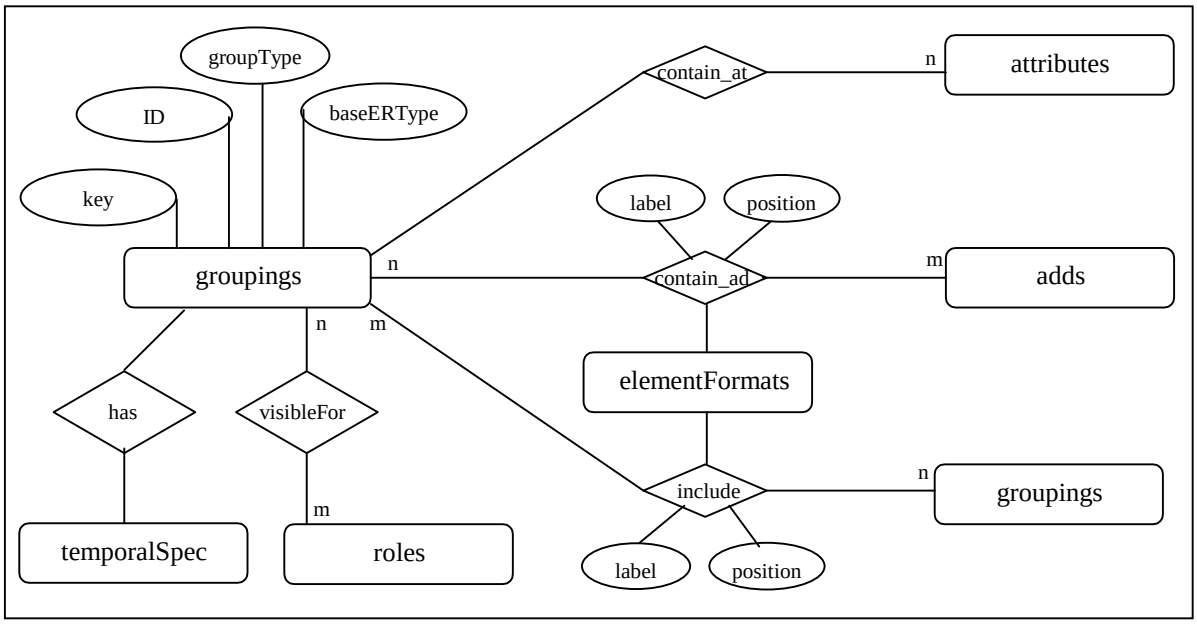

Figure 5-5: specification of the groupings meta table

The *groupings* meta table is involved in several relationships, which are similar to those of the *documents* meta table: an element-grouping may contain attributes of the corresponding base ER type, adds and other element-groupings, and may also be assigned an arbitrary number of roles and one temporal specification.

Table 5-2 shows the mapping of the *sci_info* and *proj_internal* element-groupings from the sample scenario and the connection of the latter to the containing document type using the *document_has_grouping* table.

groupings meta table			
ID	key	GroupType	baseERType
proj_internal	g3	Detail	Project
sci_info	g9	Result	Scientist

document_has_grouping meta table	
document	grouping
d17	g3

Table 5-2: mapping the sample scenario to the groupings meta table

5.1.2.3 Attributes and adds

ER document types and element-groupings may contain some or all attributes of their corresponding base ER type. Each attribute selected to be displayed in a document or element-grouping is represented by an entity of the *attributes* meta table. It contains information about the name of the selected attribute (*attr*), the *label* and *position* within the containing document type or element-grouping, the *elementFormat* (the layout specification) and the meta type of the attribute (relationship with *metaTypes*). Meta types indicate how the content of a certain attribute has to be presented, for example as a popup menu, button list or simply plain text (see section 4.5.2).

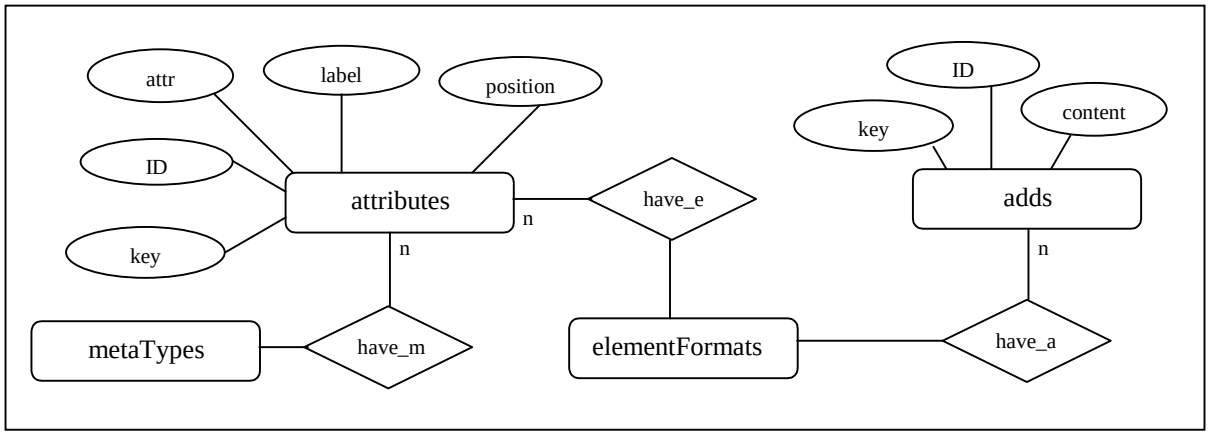

Figure 5-6: specification of the attributes and adds meta tables

Adds can be used by all document types or element-groupings and have only one *content* attribute (besides the standard *ID* and *key* attributes) for storing any kind of information. The content of such an add is not processed by the system, it can be regarded as some kind of black box. In the sample scenario, an add is used for storing a global footer that will be displayed on each document type.

The *elementFormats* entity referenced by an *attributes* entity specifies presentation issues for that attribute as described in section 4.6. The specification of the adds and attributes meta tables together with their basic relationships is shown in Figure 5-6, the mapping of the scientists attributes the sample scenario can be seen in Table 5-3.

attributes meta table								
ID	key	attr	posi-tion	label	metaType	element Format	grouping	document
first_name	a13	first_name	1		plainText	3	g9	
last_name	a14	last_name	2		plainText	3	g9	
image	a15	image	1		image	1		d21
first_name	a16	first_name	2	First Name:	plainText	5		d21
last_name	a17	last_name	3	Last Name:	plainText	5		d21
phone	a18	phone	4	Telephone:	plainText	5		d21
room	a19	room	5	Room Number:	plainText	5		d21

Table 5-3: mapping the sample scenario to the attributes meta table

5.1.3. Access primitives

Similar to the domain primitives, the HMT access primitives are mapped to a set of meta tables, one for structural and navigational links, one for all kinds of access structs, and one for query and input submitters. The following subsections describe these meta tables and the mapping of the sample scenario.

5.1.3.1 Links

In the HMT meta schema, there is no distinction between structural and navigational links, because HMT documents are not separated according to their type (e.g. *general* or *query*), which would be prerequisite for distinguishing the two types of links.

A HMT link always originates from a document type (*sourceDoc*) or an element-grouping (*sourceGroup*). Within this document type or element-grouping, a certain element (for example an attribute or another element-grouping) is specified as the *anchor_element* of the link. It is necessary to specify both the anchor and the containing source document type or element-grouping, because the element which is used as the anchor could be reused in other document types or element-groupings without being the anchor of a link there.
The target of a link can be an external presentation (attribute *destURL*) or a HMT document type (relationship type *destDoc*), which may be extended by the specification of a certain element-grouping within this target document type (relationship type *destGroup*). The first corresponds to span-to-node links, the latter to span-to-span links as described in the Dexter Hypertext Reference Model [HS94].

The *context* of the link allows specifying the presentation unit (the window), which will present the target of the link. If nothing is specified, the target of the link will replace the source document the link originates from. The attribute *elementFormat* references the layout used to display the link.

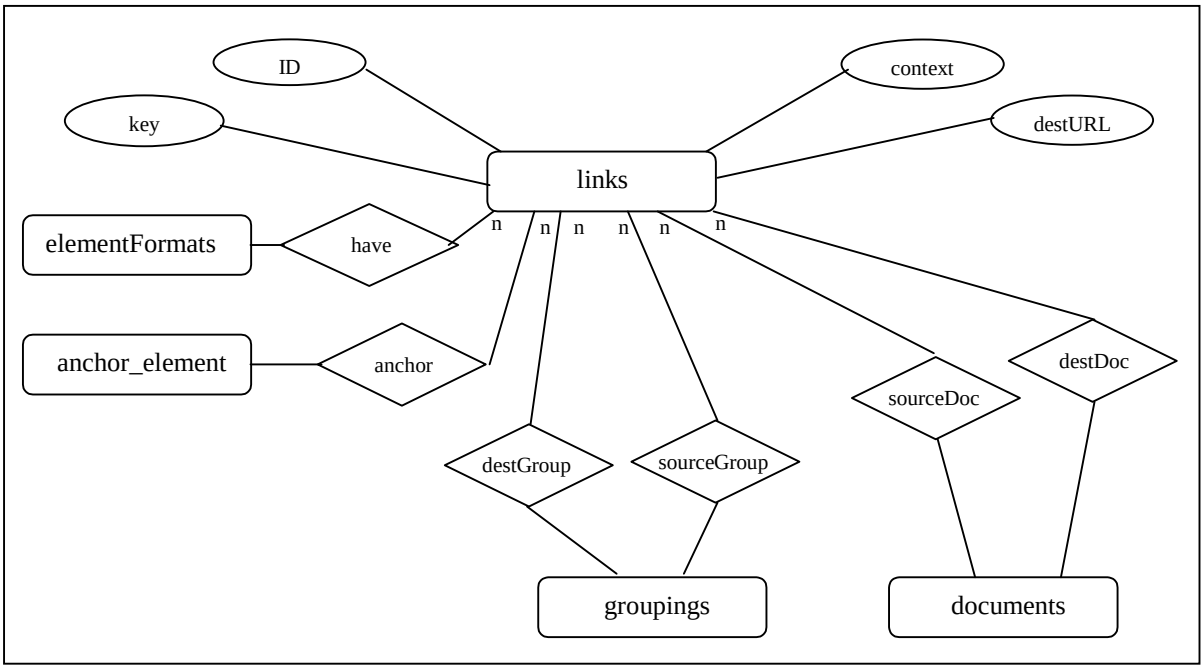

Figure 5-7: specification of the links meta table

The structural link from the sample scenario is mapped to the *links* meta table as follows:

links meta table									
ID	key	context	destURL	element Format	anchor	source Doc	source Group	dest Doc	dest Group
info-home-link	l3	window2		7	a14		g9	d21	

Table 5-4: mapping the sample scenario to the links meta table

5.1.3.2 Access structs

Access structs are used to reference a set of entities of a certain entity type. This can either be done by specifying a relationship reference, a conditional reference (see section 4.3.2) or a total reference. While relationship references obviously can only be used if both the source (relationship *sourceDoc* or *sourceGroup*) and the target (relationship *destDoc* or *destGroup*) are ER document types or ER element-groupings, conditional or total references may originate also from general document types or general element-groupings.

The attribute *relationshipRef* contains information about the relationship reference (for example, the name of the relationship type or the SQL query for retrieving the referenced entities); the attribute *conditionalRef* is used for conditional and total references. For conditional references, this attribute contains the condition as a valid SQL expression (see section 4.3.2); total references are represented by a special token (for example “*”). The

attributes *label*, *position* and *elementFormat* are already known from the other meta tables. The order of presentation of the referenced entities is specified by *orderBy*.

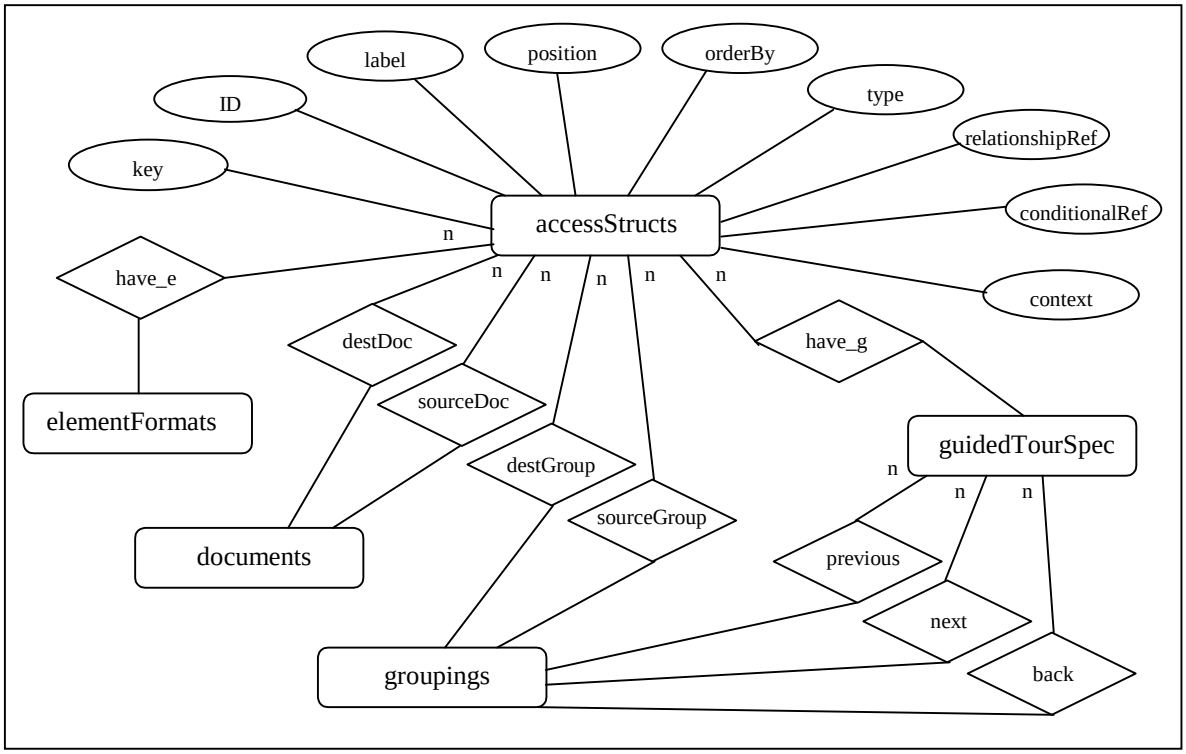

Figure 5-8: specification of the accessStructs meta table

The *type* attribute defines the type of the access struct, which can be TOC, Guided Tour or Guided TOC. If a Guided TOC or a Guided Tour is used, the meta table *guidedTourSpec* contains some further information.

We chose a separate meta table in order to enable the administrator to define the details of such a tour only once and reference it whenever needed. For example, if a standard Guided Tour on employees is defined, it can be used for access structs coming from projects, departments or any other entity type related to employees.
The *guidedTourSpec* meta table references three element-groupings *previous*, *next* and *back* which are used for navigational purposes. The *previous* and *next* groupings are used as hyperlinks pointing to the previous and next element of the Guided Tour, the *back* grouping allows returning to the document that offered the Guided Tour. By using element-groupings for the links instead of plain text, arbitrary navigational aids can be designed for Guided Tours and Guided TOCs.

Similar to the HMT link primitive, the *context* attribute allows specifying the presentation unit (the window), which is used to display the contents of external access structs. An example for the mapping of an access struct to the corresponding meta table can be seen in Table 5-5. Due to space limitations, we do not show the *elementFormats*, *conditionalRef*, and *context* attributes.

accessStructs meta table										
ID	key	label	posi-tion	type	orderBy	Relation-shipRef	source Doc	source Group	dest Doc	dest Group
sci-list	x4	Participating Scientists:	6	TOC	last_name	participates-in	d17			g9

Table 5-5: mapping the sample scenario to the accessStructs meta table

5.1.3.3 Form submitters

Form submitters are used in query and input document types in order to send the user's data to the server. A submitter always consists of a source document type (relationship *sourceDoc*) or source element-grouping (relationship *sourceGroup*) and a destination document type (*destDoc*).

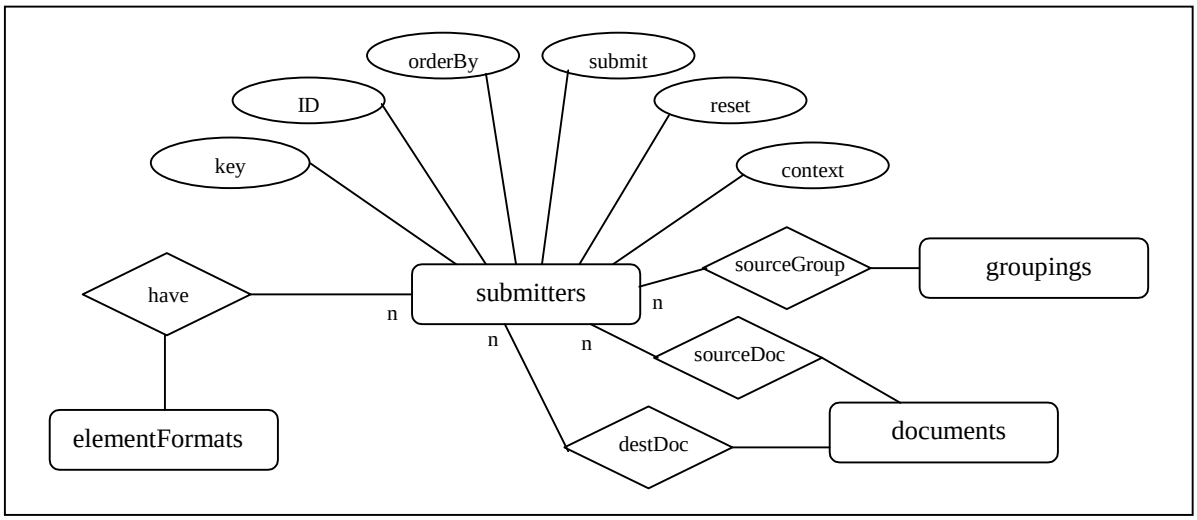

Figure 5-9: specification of the submitters meta table

Since the HMT meta schema does store the different types of documents (query, result, detail, input and general) in its meta tables, it is left to the implementation to ensure that the destination of a form submitter for query documents is always a result document type and the destination for submitters for input documents is always a detail document type.
The attribute *orderBy* is used to specify the order in which the elements are presented within a result document, *submit* and *reset* are used to label the submit and reset buttons of the query or input document. The attribute *context* and the relationship type *elementFormat* have the same meaning as described in the previous sections. Since the sample scenario does not contain submitters, the mapping to the submitters meta table is omitted here.

5.1.4. Layout specification

As described in the previous sections, the meta tables representing the HMT design primitives all reference an entity of the *elementFormats* meta table. This entity (together with the implementation of the meta types for the HMT attributes) contains all information necessary to present any HMT element on a specific hypermedia platform. By changing the code of these elementFormats, an existing HMT schema can easily be ported to a different hypermedia platform without having to change the structure on the conceptual or logical level.

Besides the primary key, this *elementFormat* meta table consists of only one attribute containing the format specification. Within this format specification, some special tokens can be used:

[**CONTENT**]
[**LABEL**]
[**RESULT_NUMBER**]
[**TOTAL**]

The first of these tokens represents the content of the element this format has been assigned to. For example, an elementFormat *table_element* could be defined for www applications as

<TR><TD> [**CONTENT**] </TD></TR>

which presents the element's content within the cell of an HTML table. If an attribute is assigned this format, the string [**CONTENT**] will be replaced by the attribute's value. If the element is an element-grouping, the whole content of the element-grouping will be put inside this HTML table cell. In a similar way, the expression [**LABEL**] can be used to place the label of the element, for example

<TR><TD> [**LABEL**] </TD><TD> [**CONTENT**] </TD></TR>

defines a table row with the first cell containing the label and the second one presenting the contents of the element. Only available for result document types are the two expressions [**RESULT_NUMBER**] and [**TOTAL**], the first of which represents the sequence number of a given result entity and the second of which represents the overall number of entities found. For example, a typical format for presentation of a query result is:

<TR><TD> [**RESULT_NUMBER**] </TD><TD> [**CONTENT**] </TD></TR>

Usually, the labels of attributes in result documents are undefined or ignored, because the meaning of the attribute's contents is mostly displayed as the heading of the corresponding table column of the document. The total number of entities for a result document is often displayed at the bottom of the result document in order to give the user a hint of how many items are still available.

5.1.5. Users and Roles

As described in section 4.4.2, HMT uses a simplified RBAC model without role hierarchies. The *roles* meta table stores the *rolename* and *description* of a role, and allows assigning roles to users (relationship type *hasRole*). The relationship types *accessibleBy* and *visibleFor* are used to assign roles to document types and element-groupings.

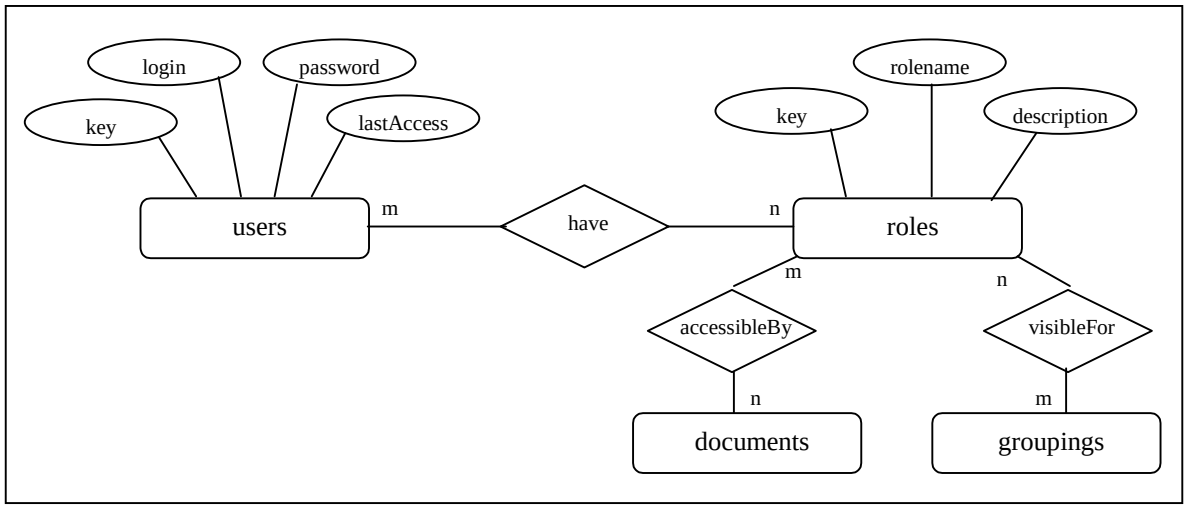

Figure 5-10: specification of the roles and users meta tables

Besides the primary key, users have a *login*, a (possibly empty) *password*, and information about their last access to the system (attribute *lastAccess*), which can be used for statistical purposes or security precautions (e.g. ask the user to login again if a certain period of time has passed since his last access).

Table 5-6 shows the mapping of the sample scenario to the *role* and *visibleFor* meta tables.

roles meta table		
rolename	key	description
partners	r1	all project partners

visibleFor meta table	
role	grouping
r1	g3

Table 5-6: mapping the sample scenario to the role and visible_for meta tables

5.1.6. Temporal specifications

The temporal specification of a document type or element-grouping is stored within the *temporalSpec* meta table and seven relationship types representing the seven temporal relations. The *temporalSpec* meta table contains an entry for each element of a document type or element-grouping, which is identified by the key of the corresponding document type (*docKey*) or element-grouping (*groupKey*) and its *position* within. Each element may have a finite *duration*, and one element per document type or element-grouping is specified to be the

start of the presentation. The seven temporal relations of HMT (see section 4.5.4.2) are directly mapped to the relationship types *meets*, *before*, *starts*, *finishes*, *synchronizes*, *overlaps* and *includes*. The relationship type *before* has an attribute storing the *delay* between the first and the second element, the relationship type *overlaps* stores additional information about the overlapping interval (*overlapTime*). The *includes* relationship type has two attributes *T1* and *T2* for the specification of the time intervals before and after the parallel execution of the two elements involved.

An alternative approach for storing the temporal specification of a document type or element-grouping would be to introduce temporal relationship types directly connecting the various meta tables for the HMT elements, for example *attributes* and *links*. Although this approach could avoid the redundant storage of the element's position within the containing document type or element-grouping, it has a major disadvantage:
A large number of relationship types would be necessary to completely support the temporal capabilities of HMT. Each kind of element (attributes, adds, links, access structs, submitters and element-groupings) needs seven relationship types itself plus seven relationship types with each of the other elements, which leads to 252 relationship types. This increases the complexity of the corresponding CASE-tools significantly and reduces the performance of an implementation following this approach.
For this reason, we have decided to store the temporal specification of document types or element-groupings in a separate meta table as shown in Figure 5-11.

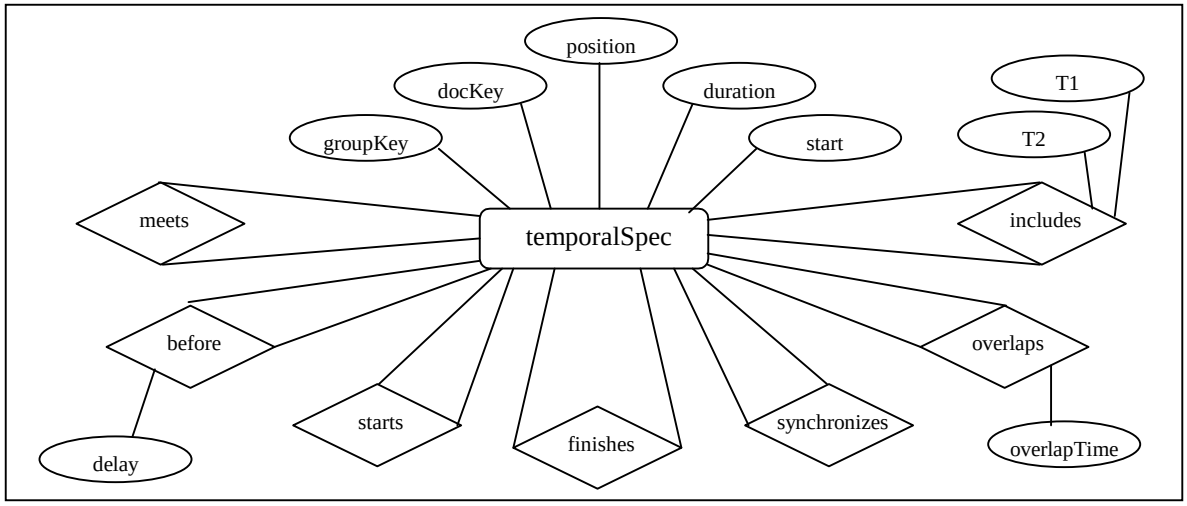

Figure 5-11: specification of the temporalSpec meta table

5.2 Document generation

The information stored within the HMT repository can be used by a CASE-tool in order to create the corresponding hypermedia application. This can happen both at runtime (online generation) and at a specific point of time (offline generation). In both cases, the algorithm for creating the hypermedia documents is identical except for the URL generation procedure, which has to create dynamic URLs in the first case and static URLs in the second case.

In general, the generation algorithm can be divided into four parts:

- checking the user's authorization
- querying the HMT repository
- creating the appropriate SQL queries
- accessing the application database and generating the hypermedia document

First, the HMT repository has to be queried in order to get information about access restrictions on the document or element-grouping requested by the user. If the user is granted access, the second step of the generation algorithm determines the structure of the requested document or element-grouping. This refers to the contents (for example attributes or adds), the navigational design (links, submitters and access structs), the layout specification (for example headers, footers or elementFormats) and the temporal design. Third, the corresponding SQL queries for retrieving the specified data have to be generated. In the last step, these queries are executed in order to retrieve the application data from the database and the document is created accordingly.

The following subsections describe the document generation algorithms for each type of HMT document in detail.

5.2.1. General document types

General document types or element-groupings are not bound to a certain base ER type, so they cannot contain attributes. Nevertheless, the third step of the document generation algorithm (querying the application database) cannot be ignored, because general documents might have internal access structs with a total reference addressing data from the application database.

After user authorization, the layout information, header and footer of the current document type are retrieved from the repository. This is the only step which does not apply for the generation of element-groupings, because these are only parts of hypermedia documents and therefore do not have this kind of information. Since header and footer of a general document type are general element-groupings, the generation algorithm is applied recursively.

Next, all adds, element-groupings, links and access structs of this document type together with the corresponding elementFormats are retrieved and sorted according to their position. The content of an add is simply displayed without further processing using the corresponding elementFormat as described in section 5.1.4. Links in general documents are always navigational links and may be internal or external. For external links, the link target (which is a URL string) can directly be used to create a hyperlink. The target URL of internal links has to be computed from the target document type and perhaps the target element-grouping specified.

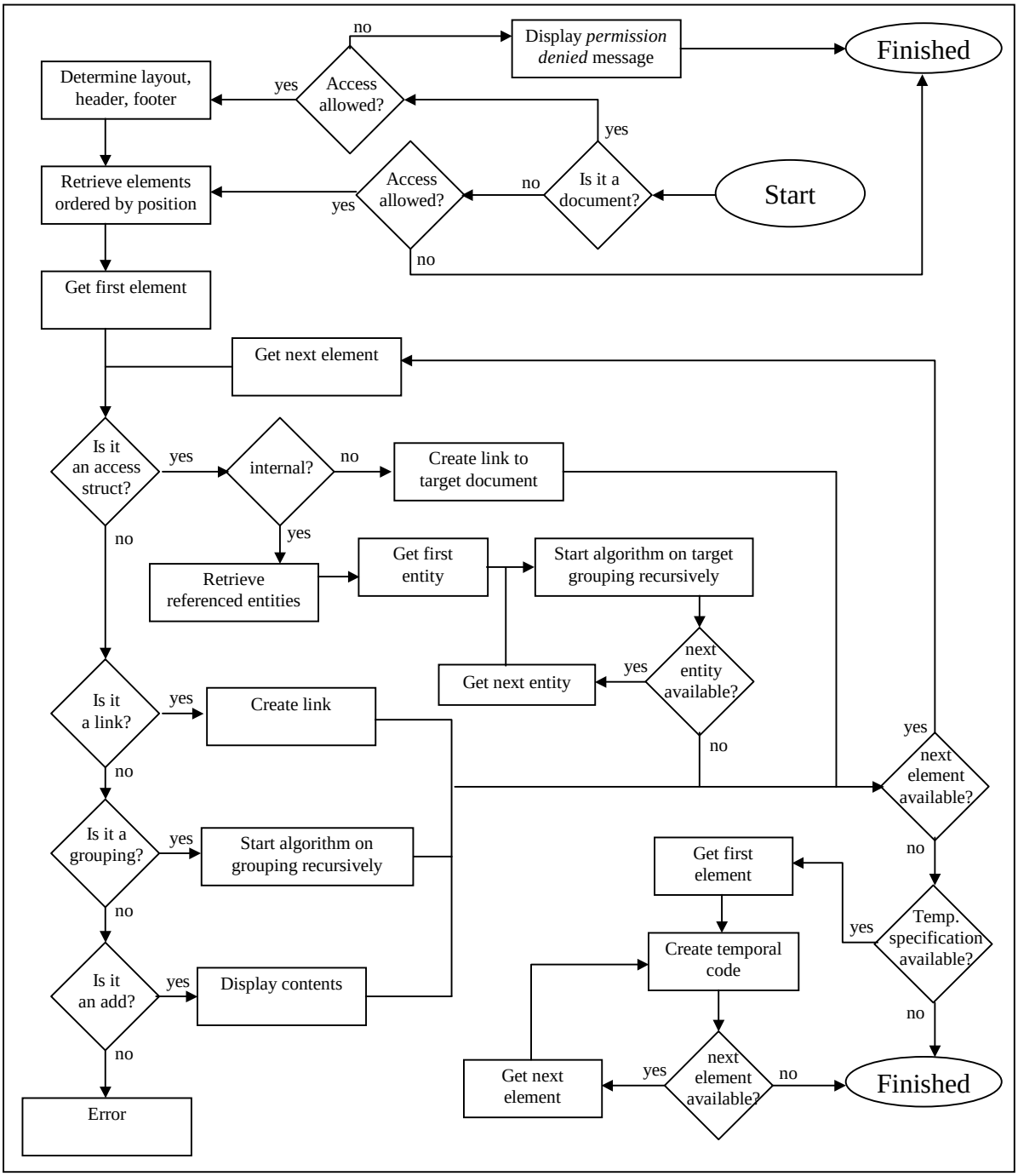

Figure 5-12: page generation algorithm for general document types

Access structs in general document types may be internal or external: External access structs are simply mapped to a hyperlink with the access struct's label as the anchor and the specified result document type as its target. As an additional parameter, the link has to provide either the identifier of the access struct used, or all information needed to create the corresponding SQL query when creating the target document. While the first approach is easier to implement, the second one leads to a better performance, because the repository has not to be queried again for generating the SQL query retrieving the relevant entities to be displayed on the target document. For internal access structs, the set of referenced entities has to be

calculated and displayed at once. In both cases, the algorithm generating the hypermedia output consists of two steps: First, the keys of the referenced entities have to be calculated according to the reference type of the access struct. Total references address all entities of a given entity type, whereas conditional references result in a SQL query where the condition is used as the selection operation. The third type of reference, relationship reference, is not allowed for general document types or general element-groupings. In the second step, the target document or element grouping of the access struct is used to generate the presentation for each of these entities, that means the generation algorithm is called recursively.

Finally, the temporal specification of the document or element-grouping has to be created. After the starting element has been identified, the times of presentation of all other elements are calculated and specified in the language of the hypermedia system used. Figure 5-12 shows the complete generation algorithm for general document types and element-groupings.

5.2.2. Input document types

One major difference between input document types and general document types is that input document types are assigned a base ER type. In addition to the elements already described in the previous section, they are allowed to contain a subset of the set of all attributes of their base ER type. For each attribute of this subset, some kind of input field is generated where the user can enter his data. If the input document is used for an update operation (that means it has been referenced from a result or detail document type), the attribute values of the current entity are used as default values for the corresponding input fields. In a similar way, previously specified values for any access structs are used as preselected default values in the corresponding input form.

Additionally, input document types or element-groupings have a form submitter, which is used to initiate the storage process and to reference a detail document to be displayed afterwards. The storage process has to insert a complete entity into the corresponding database table, no matter which subset of the set of attributes has been presented on the input document. This means that a value has to be provided for each attribute independent of whether it has been included in the input document or not. For attributes not included in the input document, this will be either a null value or a default value the administrator has specified as described in section 4.5.3.3.
Another difference can be found in the treatment of access structs. Input document types are not allowed to have external access structs, because an input form is always limited to one document. If an input form would be divided into several parts on different documents, HMT had to provide its own transaction management and a HMT implementation (and its usage) would be much more complicated. Therefore, distributed input forms are currently not supported, but considered to be an aspect of future research.
While internal access structs in general documents reference a set of related entities of the target entity type, access structs in input documents offer all entities from the target entity type for selection. This means that an internal access struct does not allow to insert an entity into the target ER type, but to relate an already existing entity from the target ER type to the

current entity of the document's base ER type. This is typically done by assigning the meta type popup to one of the attributes of the target element-grouping. For example, an input document type for employees might contain an internal access struct on departments with a target element-grouping consisting of the attribute name (of the department) with meta type popup. On such an input document, a new employee entity can be defined and the corresponding department can be chosen from the popup menu, but the access struct can't be used to insert a new department entity into the database. For that purpose, a departments input document had to be created.

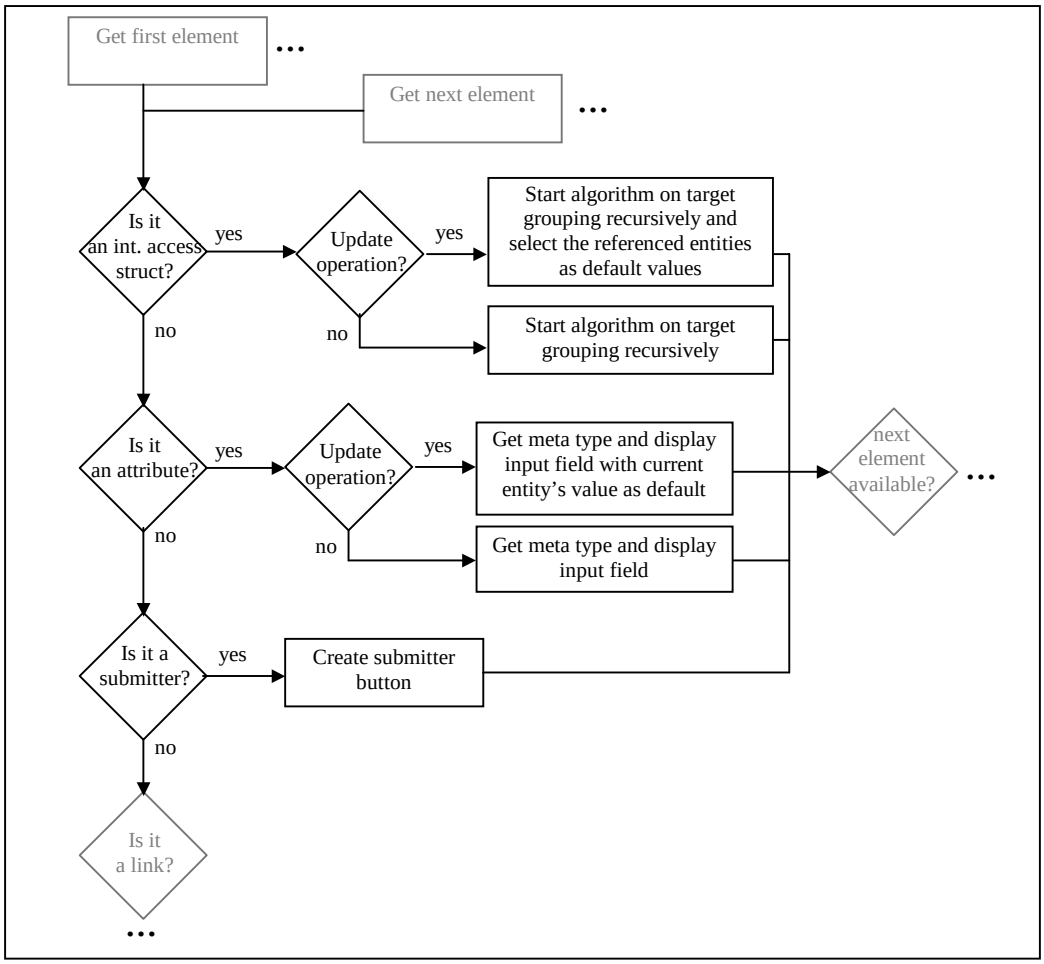

Figure 5-13: specific aspects of the input document generation algorithm

Since most parts of the document generation algorithm for input document types can be found exactly identical in the algorithm for general document types described in the previous section, Figure 5-13 only shows the components that are specific for input documents.

5.2.3. Query document types

Query documents are created using the same algorithm as for input documents, but there are some small differences: First, query documents are never related to a certain entity like input documents can be (for update operations). Thus, we do not have to differentiate the operation

mode for attributes and access structs. Second, the semantics of internal access structs is slightly different. An internal access struct on input documents offers all entities available from the target ER type for selection. An access struct on query documents only offers the (distinct) set of entitles of the target ER type which are referenced by entities from the current base ER type. This constraint is obviously useful and necessary, because the selection of any other entity of the target ER type not contained in the set of referenced entities would produce no query results. To avoid user frustration, these impossible combinations of entities should not be offered for selection.
Third, it is left to the implementation to decide whether additional input fields are offered for each attribute where the user can select how to combine multiple search expressions, for example with a logical *and* or a logical *or* operation, or if one of those is selected to be the default operation. And last but not least, the submitter of a query document type always references a result document type as its target instead of a detail document type. This is necessary because an input document type always addresses (stores) one entity, whereas a query document will often lead to a set of results to be displayed.
Since most differences between query and input documents concern semantics and not the generation algorithm, we refer to Figure 5-13 for a graphical specification of the query document generation algorithm.

5.2.4. Detail document types

The document generation algorithm for detail document types extends the algorithm for general document types by adding the option *attributes* to the list of elements of a document.

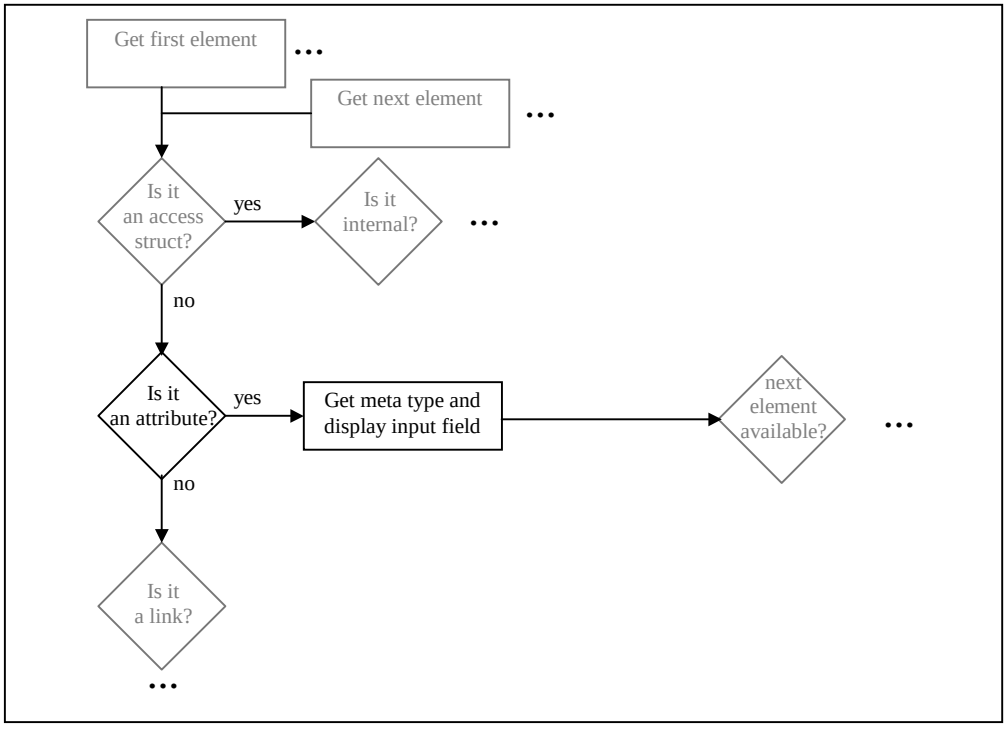

Figure 5-14: specific aspects of the detail document generation algorithm

The content of an attribute is displayed by using its meta type as specified in section 4.5.2. Figure 5-14 shows the modified section of the detail document generation algorithm.

5.2.5. Result document types

The algorithm for the generation of result documents is different from the other algorithms described so far.

The most significant difference lies in the fact that result documents are the only HMT documents presenting more than one entity of their base ER type. This means that the specification of a result document type or element-grouping is applied on every entity to be displayed. The only exception to that rule is the (optional) query element-grouping used to refine the query which is only displayed once and not for each entity.

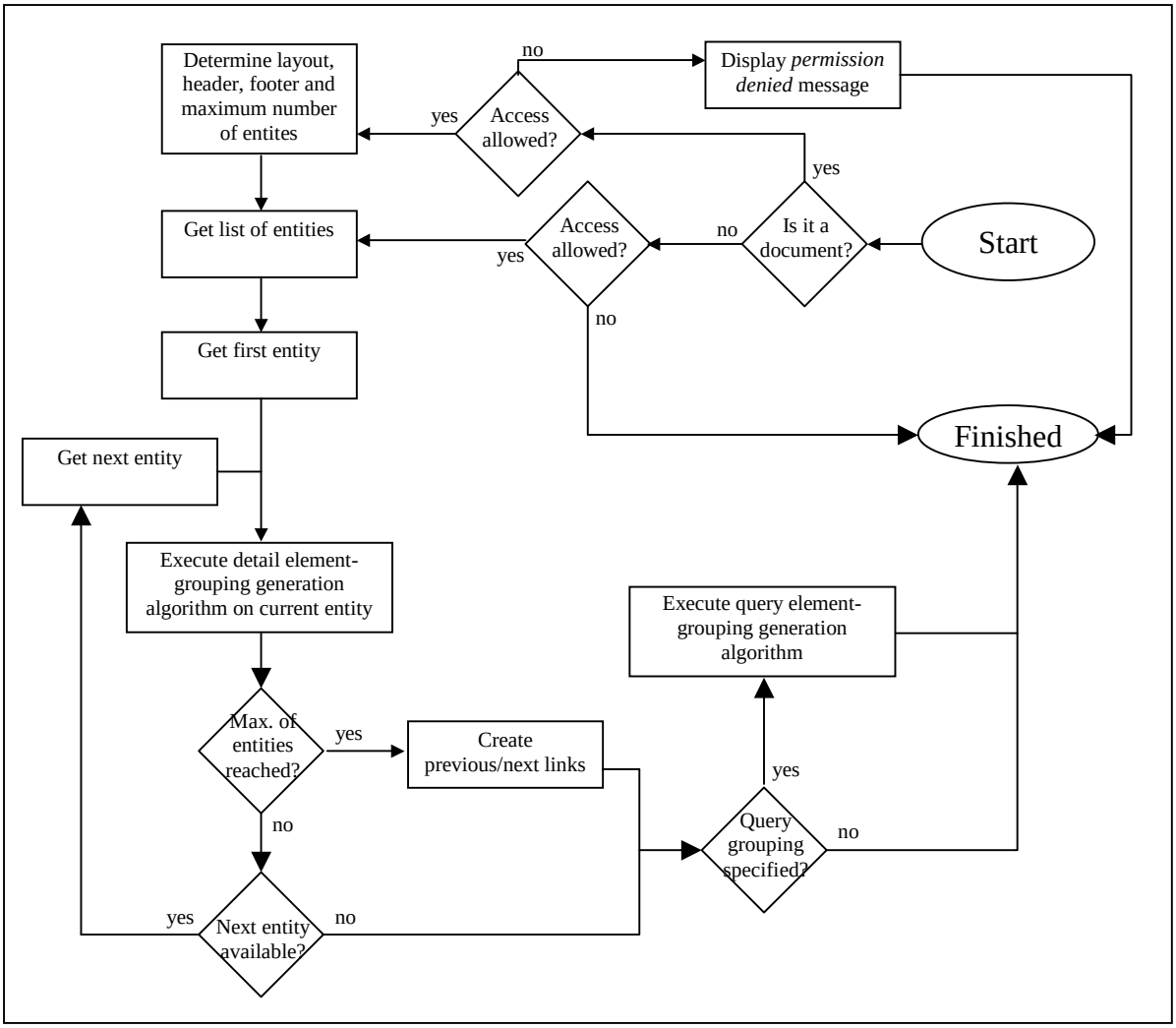

Figure 5-15: specification of the result document generation algorithm

Since there may be specified a maximum number of entities per document, the algorithm has to limit the presentation of entities accordingly and provide links for addressing the documents containing the previous or next portion of results.

Figure 5-15 shows the document generation algorithm for result documents. To keep the diagram simple and readable, the action *´execute detail element-grouping generation algorithm on current entity´* hides the complete detail document generation algorithm (see Figure 5-12 and Figure 5-14).

5.3 Automatic query generation

The previous two sections introduced the HMT meta tables and specified the algorithms for the creation of hypermedia documents based on the HMT repository. The SQL queries retrieving the information from the meta tables are generated straightforwardly and don't have to be discussed explicitly, and the same holds for the queries retrieving information about a certain entity, which has to be displayed on a details document. Even the generation of the SQL query retrieving a set of entities referenced by an access struct on a details document is trivial and needs no further investigation. But the more complex, interactive query and input document types show a variety of aspects suggesting a closer examination of the query generation process.

5.3.1. Query generation for input document types

Input document types may require more than one SQL query in order to store all the information specified by a user. The process of generating the appropriate insert or update statements can be divided into two parts:

First, an entity of the base ER type has to be stored. For this purpose, a SQL query is created where all attributes that appeared in the query document are assigned the corresponding values entered by the user. The remaining attributes are either assigned NULL or a default value specified by the administrator (see section 4.5.3.3).

Definition 5.1 (interpretation of attributes in input document types):

Let $d \in DOC_e^{input}$ be an input document type with base ER type e, $dom(a_1)$ be the domain and $def(a_1)$ be the default value of a given attribute $a_1 \in ATTR_e$ and $fieldValue(a_2)$ be the value of the input field for an attribute $a_2 \in ATTR_d$, then the function v: $ATTR_e \rightarrow \bigcup_{a \in A_e} dom(a)$ assigning each attribute of e a value is defined as

$$v(a) = \begin{cases} fieldValue(a) & if\ a \in ATTR_d \\ def(a) & if\ a \notin ATTR_d \wedge def(a) \in dom(a) \\ NULL & otherwise \end{cases}$$

Second, one insert or update query has to be created for each access struct in order to store the connections defined by the user on the input document. This is the common way of specifying relationships in a RDBMS: Depending on the cardinality of the relationship type, either the current entity (for n:1 relationship types), an entity of the target ER type (for 1:n relationship types) or entities of a third ER type (for m:n relationship types) have to be updated or inserted accordingly. In order to preserve consistency of the data source, all queries have to be executed within one transaction.

5.3.2. Query generation for query document types

The algorithm for the automatic generation of SQL queries from query documents is more complicated than the algorithm for input documents, because query documents may have nested access structs and additional features like multi-attribute input forms (see section 4.5.3.1). Moreover, query document types have to produce a single SQL query calculating the result set if the results are to be displayed in a certain order, whereas input documents may generate a number of separate requests.

The evaluation of a query document d generates a SQL statement applying the following rules:

The SQL query generated from a HMT query document *d* with base ER type e_d contains always a projection on the key attribute(s). This is the only projection needed, because the query is only used to identify the set of relevant entities. The presentation of these entities is handled by the corresponding result document type, which generates its own SQL statement.

Definition 5.2 (general form of SQL queries resulting from query document types):

Let $d \in DOC_e^{query}$ be a query document type with base ER type e_d and key_1 .. key_n be the key attributes of e_d, then the generated SQL query has the general form

$$query(d, e_d) = \pi_{key_1,...,key_n}(e_d)$$

Attributes of the base ER type having distinct labels lead to a selection operation on these attributes with the values of their corresponding input fields. It is left to the implementation to decide whether the operator Θ is replaced by a logical *and* or a logical *or*. Attributes with empty input fields are ignored.

Definition 5.3 (interpretation of attributes with distinct labels in query document types):

If a query document type *d* contains attributes with distinct labels, the generated SQL query is extended as follows:

$$query(d, e_d) = \pi_{key_1,...,key_n}(E_1(d, e_d)) \text{ where}$$

$$E_1(d, e_d) = \sigma_{a_1 = fieldValue(a_1)\Theta...\Theta a_n = fieldValue(a_n)}(e_d) \text{ with } a_i \in ATTR_d, \Theta \in \{and, or\}$$

$$and \quad fieldValue(a_i) \neq NULL$$

Multiple attributes having identical labels lead to a set of selection operations on these attributes, each with the value specified in the shared input field. Again, the selection of the

connecting operator is left to the implementation of a HMT system, so θ can either be a union or an intersection. All attributes of a shared input field are ignored if the corresponding field is empty.

Definition 5.4 (interpretation of attributes with identical labels in query document types):

If a query document type *d* contains attributes with identical labels, the generated SQL query extends the specification of Definition 5.3 by adding the expression E_2:

$$query(d, e_d) = \pi_{key_1,...,key_n}(E_1(d, e_d)\ \theta\ E_2(d, e_d))\ where$$

$$E_2(d, e_d) = \sigma_{a_1=v\Theta...\Theta a_n=v}(e_d)\ with\ a_i \in ATTR_d\ and\ v = fieldValue(a_i),\ v \neq NULL$$

Access structs are mapped to joins with the target ER type. The query generation algorithm is executed recursively on the target element-grouping specified:

Definition 5.5 (interpretation of access structs in query document types):

If a query document type *d* contains access structs, the generated SQL query extends the specification of Definition 5.4 in the following way:

$$query(d, e_d) =$$

$$\pi_{key_1,...,key_n}(E_1(d, e_d)\ \theta\ E_2(d, e_d)\ \theta\ query(t_1, e_d \rhd\lhd e_{t_1})\ \theta ...\theta\ query(t_n, e_d \rhd\lhd e_{t_n}))$$

where t_1 .. t_n are the target documents or element-groupings of the access structs $s_1..s_n$ of document *d*.

5.4 Summary

This chapter has presented the HMT meta schema, which is used for storing HMT application schemas.

Section 5.1 introduced the basic meta schema of about 14 meta tables and more than 40 relationship types, which can be considered as a basis for the implementation of a HMT CASE-tool. Additional meta tables or extensions of existing HMT meta tables can (and probably will) be made by actual CASE-tool implementations in order to provide more user friendly and powerful interfaces for administration and presentation.
However, the basic HMT meta schema described here is capable of representing any application designed with HMT as introduced in chapter 4, including all features like access restrictions, query refinement, temporal design or adaptive documents.

Afterwards, section 5.2 discussed the document generation algorithms for the five kinds of document types (general, query, result, detail and insert). We presented in detail how hypermedia documents are created using the HMT meta schema. Although the five document generation algorithms show similar structures, some essential differences could be identified. Especially the result document type with its query refinement capabilities requires an individual solution.

Finally, a special aspect considering the two interactive document types (query and input document type) has been discussed. The information gathered by documents of this kind needs to be converted into a SQL query, which is either querying or updating the underlying database. The corresponding query generation algorithms have been described in section 5.3.

The following chapter will describe the implementation of a prototype HMT CASE-tool for the Hypermedia Modeling Technique based on the meta schema presented in this chapter.

CHAPTER 6

THE WEBCON PROTOTYPE

The best way of verifying a design methodology is the successful development of a corresponding CASE-tool. Based on the experiences from several online projects [Ab00, Mli00, Ttm00] using earlier versions of a repository-based design tool, the WebCon toolkit has been completely re-implemented in order to support the design of web applications with HMT.

One of the main objectives for this project was to achieve maximum flexibility regarding platform, database vendor and web server by using standardized interfaces and techniques. Although these standard interfaces often have a lower performance compared to proprietary interfaces, this disadvantage can usually be neglected because mostly response times and transmission rates of the web build the bottleneck of such applications. The initial WebCon version described in this chapter focuses on the generation of WWW applications, but can easily be extended in order to support different hypermedia formats.

Section 6.1 will present the architecture of the system, followed by a detailed description of the actual implementation with its features in section 6.2. After a discussion of open issues in section 6.3, the chapter ends with a short summary in section 6.4.

6.1 Architecture

One of the major drawbacks of the first versions of WebCon [ZS98, SZ99] was its dependency of platform and database system. The CGI binary for the gateway was written in C for the Solaris platform and the only database system supported was TransBase [Tra00]. A quick installation of the corresponding information systems on other platforms, for example in order to provide a standalone version for fairs or symposiums on a PC, could not be managed.

In order to avoid these problems, the current version of WebCon is designed to be completely independent from platform, web server and database system. The tool is implemented in pure java using the standard java servlet API. Servlets are server-side java programs that are called by the web server. Since each web server either supports servlets directly or at least can be extended by so-called servlet-engines, arbitrary web servers can be used with WebCon. In our test environment, we used the Apache web server and the JServ servlet engine.

For accessing the DBMS, we use the Java Database Connectivity (JDBC) [Dic97] interface. This guarantees maximum flexibility regarding the database system, because practically all database vendors meanwhile offer JDBC drivers for their products.

The WebCon toolkit consists of two major parts: the administration component and the page generation component. The administration component is used for designing a HMT schema of the application and mapping it to the HMT repository. It analyzes the data dictionary of the database in order to get information about the tables, attributes, primary and foreign keys. The page generation component queries the WebCon repository for the structure of the pages and retrieves the corresponding application data. Figure 6-1 shows a graphical description of the WebCon architecture.

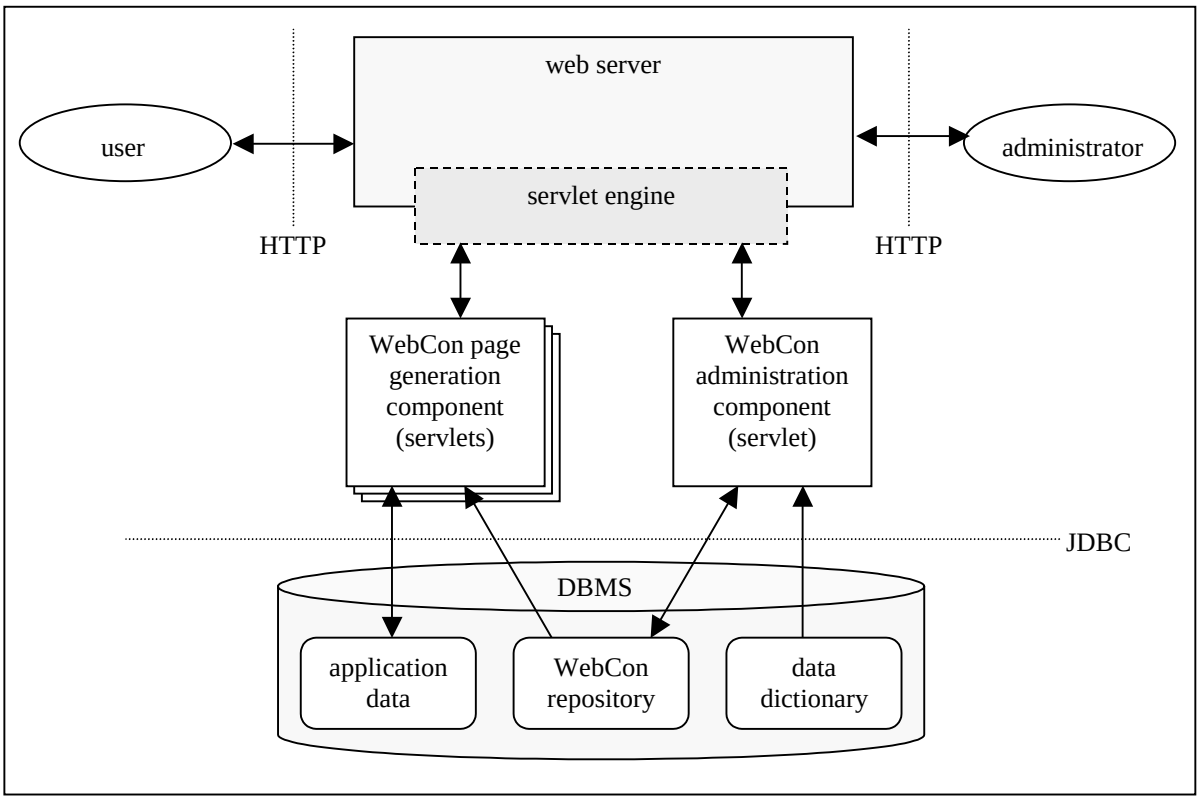

Figure 6-1: architecture of the WebCon toolkit

The current version of WebCon offers online page generation, that means the HTML pages are generated at the moment the user sends his request. Later versions might additionally support offline generation with an algorithm similar to that described in [Som00]. However, result documents always have to be generated dynamically, because their contents can only be determined at runtime after the user has specified his query. In order to optimize the page generation process, caching strategies have been developed, which will be described later in section 6.2.3.

Due to the statelessness of the HTTP protocol, a session management has to be implemented. The maintenance of a session status is necessary for a variety of situations, for example when

splitting long query results into several connected pages that can be browsed by the user. Since cookies are not defined by a standard and are often not accepted by users, we decided to use a page based session management, that means all necessary information is kept within the HTML page and sent to the server at each request. In general, this refers only to two or three variables and therefore does not noticeably reduce response and transmission times of the system.

The following section discusses some implementation details concerning functionality, the HMT repository, authorization, the application generation process and system administration with the WebCon toolkit.

6.2 Implementation

The WebCon prototype has been implemented on a Sun Ultra1 workstation using Java Development Kit version 1.2, the Apache 1.3.12 web server, JDBC and the TransBase v4.3.3 relational database system. Other DBMS like Adabas, MySQL and PostgreSQL have also been tested [See00]. The GNU revision control system (RCS) has been used to guarantee efficient and secure multi-user development.
The WebCon repository is mapped to a relational representation consisting of totally 26 tables. These tables correspond directly to the meta tables described in section 5.1 and do not need to be listed explicitly. The following sections will discuss some interesting implementation aspects of WebCon in detail and specify the HMT functionality supported by our prototype.

6.2.1. Configuration

Various parameters of the WebCon tool can be defined in a configuration file that is read at system startup. It contains the following information:

```
#
# WebCon configuration file
#
JDBC_DatabaseURL=jdbc:transbase://sunwibas13.forwiss.de:4340/hmt
JDBC_User=tbadmin
JDBC_Passwd=admin
Schema_Cache=enabled
Session_Timeout=5
```

The first parameter specifies the JDBC database URL for connecting to the DBMS including the port number, and the values of *JDBC_User* and *JDBC_Passwd* are used for database login. This account must have at least read access for the tables of the application domain and additionally write access for all tables that are to be updated by HMT input documents. The *Schema_Cache* parameter allows switching the schema cache on and off, and the maximum

idle time before automatically closing a user session is defined by the *Session_Timeout* parameter (in minutes).

6.2.2. Authentication and Authorization

Both authentication and authorization are handled by the WebCon system and not by the web server or the DBMS. This provides maximum flexibility and independence from third party systems as described in section 3.3.4. The WebCon prototype uses a fixed account for connecting to the database, which is specified in the WebCon configuration file (see previous section).

User authentication is needed when the user requests a protected document for the first time. In this case, an authentication document is created instead of the requested page. This authentication document is a special kind of general document, which can be specified by the administrator just like ordinary general document types. The only condition for authentication document types is that they must include an add containing the special token [**Authform**]. This token will be replaced at runtime by two input fields for login and password of the user and a submit button for sending the data.

After having entered a valid login/password pair, the user gets assigned a unique identifier (UID) that will be transmitted for each request of the current session. All roles assigned to the user get activated and the originally requested document is now retrieved if at least one of the user's roles has access permissions. Otherwise, an error message will be sent. Subsequent requests for restricted documents do not require another user authentication, because now the user has a UID that is valid for the whole session.

The length of a session is determined by the specification of a maximum idle time interval in the configuration file, that means the maximum period of time after his last request. If the user exceeds that time limit, the session is automatically closed, the user's UID becomes invalid and another request for a protected document requires a new user authentication. Since multiple authentication documents can be designed, the administrator can use different authentication documents within one application.

In contrast to documents, element-groupings with access restrictions do never lead to user authentication. If the user does not have the appropriate permissions or even no UID for the current session at all, the restricted element-groupings will not be displayed (adaptive documents, see section 4.4.2.2).
This implies that the administrator of a WebCon application has to ensure that before accessing an adaptive document, the user has to be lead to at least one restricted document in order to activate user authentication. Alternatively, the adaptive document itself might also be restricted, but then inconsistencies between the roles allowed to access the whole document and the roles used for document adaptation might occur.

6.2.3. Efficient Page Generation

The generation of the HTML pages in WebCon follows the algorithms presented in section 5.2. After the user's authorization has been confirmed, the different elements and access structures of the requested document are identified and mapped to SQL queries retrieving the application data from the database system. The resulting HTML page is sent back to the user.

Crucial for the performance of the page generation process are the SQL queries to the database system. They can be divided into two groups:

- The schema queries are used to retrieve information about the structure of the HTML page from the WebCon repository. For this purpose, the meta tables for every domain and access primitive have to be queried, which leads to currently at least eight SQL queries for every detail document (one for layout, header, footer, attributes, adds, element-groupings, links, access structs and perhaps Guided Tour).
- The data queries are generated dynamically from the results of the schema queries and retrieve the actual data from the database. At least one query is needed for detail documents in order to get the information about the current entity, and each contained element-grouping or access struct requires another query.

The WebCon prototype uses two optimization strategies to improve the performance of the page generation process. First, the number of data queries is minimized by storing the current entity in an entity-cache during the page generation process. Thus, repeated requests for attributes of the current entity can be answered without database access. This is an advantage especially for highly structured documents containing several element-groupings, because otherwise each subsequent element grouping resulted in a recursive call of the page generation process with one additional data query. After the page generation algorithm has finished, the corresponding entity is removed from the entity-cache. This guarantees that always the latest information from the database is presented and changes to the data source become visible immediately without having to clean the application cache manually.

Of course keeping the entity in the entity-cache would improve the overall performance for subsequent requests, but this is only suited for applications with a rather static data source. In that case, however, an offline generation of HTML pages would show the best performance under the same circumstances. Additionally, applications with a large data source would require a very large cache size and efficient replacement policies in order to improve system performance significantly.

The second and even more efficient optimization strategy reduces the amount of schema queries by storing the structure of a document within a so-called schema-cache. The schema-cache is not emptied after the page generation process has finished, thus subsequent calls for the same document type require no schema queries at all. Since the HMT schema of an application is completely independent of the size of the application database and usually much smaller, the required schema-cache size is limited. Only if the HMT schema of the application is changed, the schema-cache has to be emptied. But since the schema of an

application is usually not altered as often as the data source itself, this approach provides a reasonable compromise between high performance and simple administration.

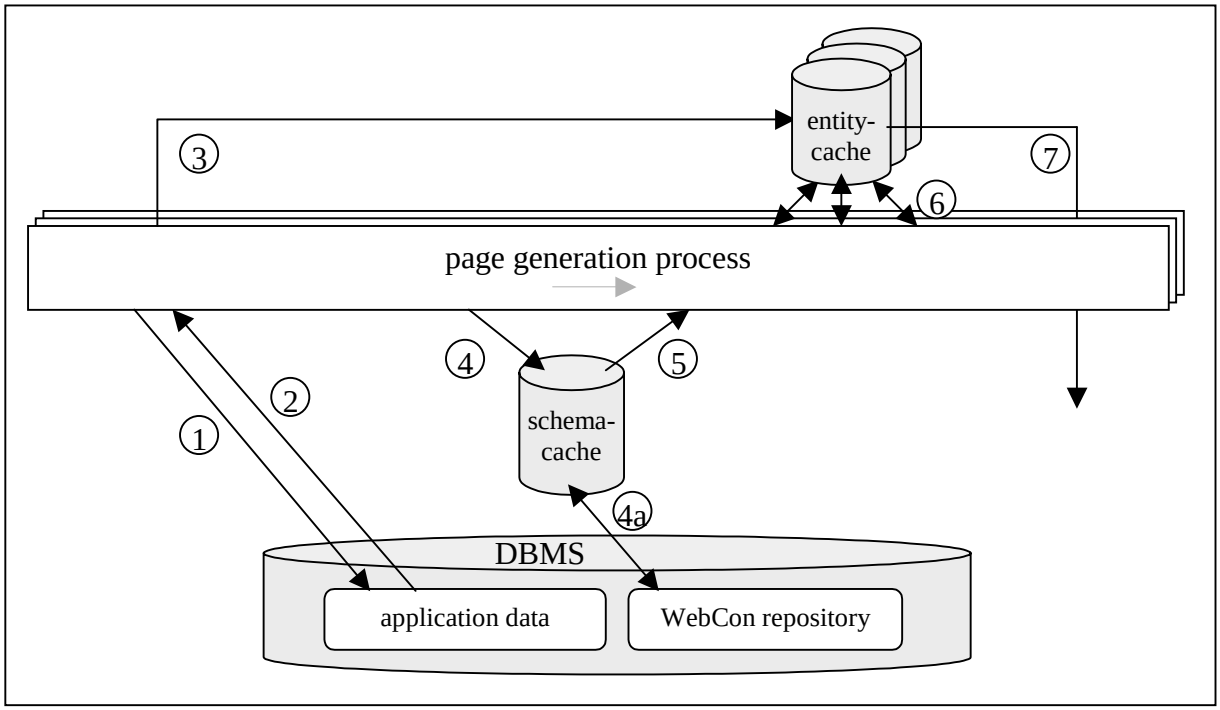

Figure 6-2: caching strategies of the WebCon toolkit

Figure 6-2 shows the use of the entity-cache and the schema-cache. The page generation process first retrieves the current entity from the application database (step 1 and 2) and stores it in the entity-cache (step 3). Afterwards, the structure of the requested page is loaded from the schema-cache (step 4 and 5). If the corresponding document type is used for the first time, its structure is read from the WebCon repository and stored in the schema-cache (step 4a). The page generation process now begins to build the page sending all requests for information about the current entity to the entity-cache (step 6). After having finished the creation of the page, the entity is removed from the entity-cache (step 7).
If several page requests are received at the same time, multiple page generation processes are started. While each of these processes has its own entity-cache (for the current entity), they all share the same schema-cache.

Initial tests of the WebCon prototype have been performed using a copy of the abayfor-online [Ab00] database, which currently contains about 540 research projects and 780 scientists. The sample screenshots of Figure 6-3 show a project overview document containing some information about the mistral research project. Besides the title, image, abstract and other information, a Guided Tour on participating scientists is offered with the button at the top of the page. The second browser window shows the presentation of one of the participating scientists in that Guided Tour. The buttons labeled *previous* and *next* are used for navigating within the Guided Tour, the *back* button is a link leading back to the calling page (in this case the mistral project overview).

Figure 6-3: sample screenshot of two HTML pages generated by WebCon

6.2.4. Administration

The WebCon administration component is used for mapping conceptual HMT schemas to the WebCon repository. The current version is a HTML-based tool allowing administration of all steps of the HMT design process using an ordinary web browser.

The administration tool consists of two frames as shown in Figure 6-4. The frame on the left side of the screen allows selecting the HMT design step. In the screenshot, the conceptual HMT design of a detail document has been selected. The frame on the right side usually provides a linked list of pages for specifying the relevant information. In our example, the page for selecting a detail document for further administration is shown, and the link labeled *[Basics]* at the top of the right window leads to a set of pages allowing the specification of related adds, element-groupings, links or access structs.

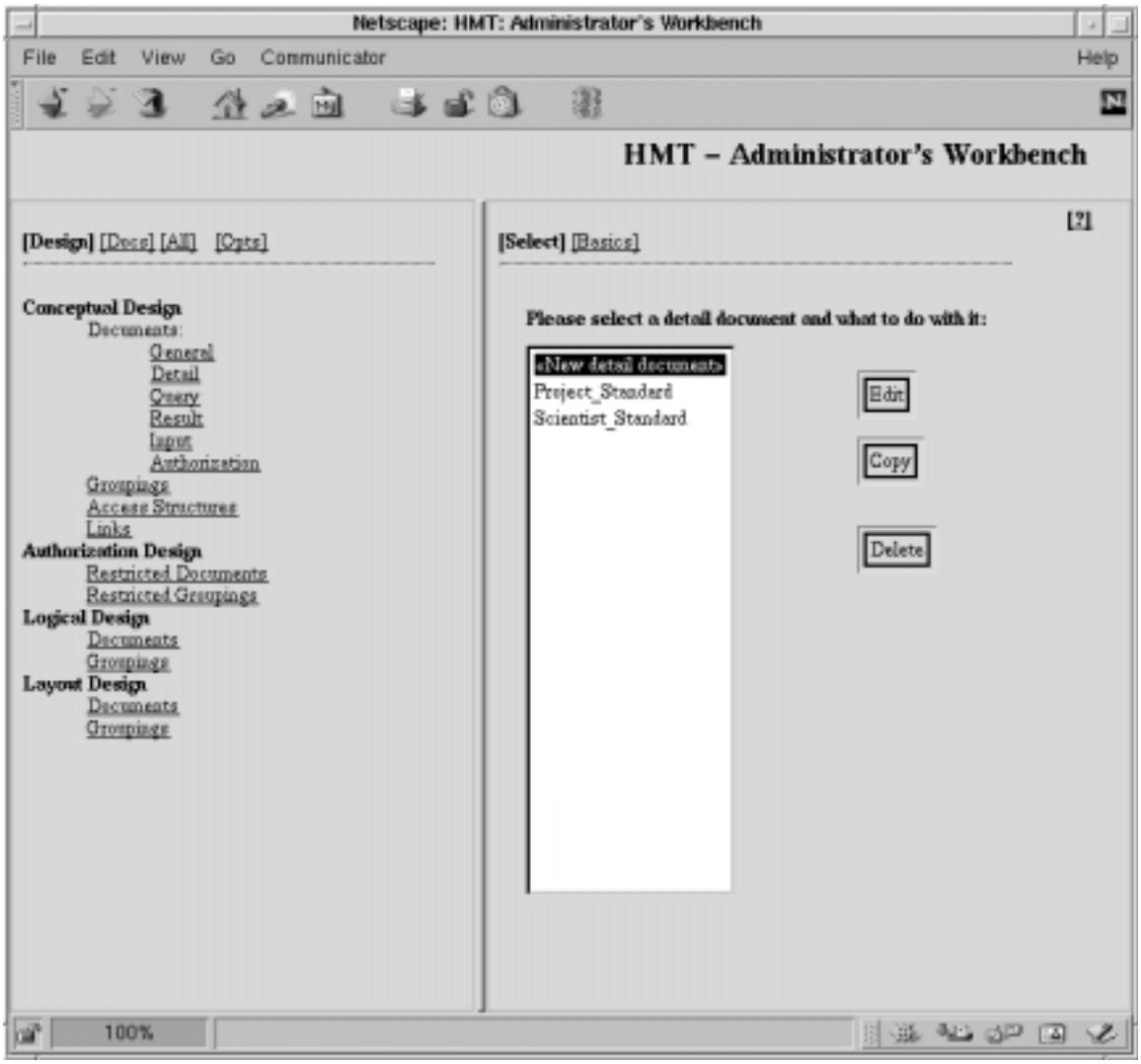

Figure 6-4: screenshot of the WebCon administration tool

Developing the administration tool as a browser-based component provides the advantage of mobile administration of the whole WebCon application without the need of a direct connection to the server (for example with *telnet* or *ssh*). Since no java applets are used, the administration tool is not affected by browser incompatibilities concerning java and the small page sizes even allow using low bandwidth internet connections.

6.2.5. Supported HMT Functionality

The WebCon prototype concentrates on the development of WWW applications using the Hypermedia Modeling Technique. Since the techniques and standards of the World Wide Web (HTML, DHTML, Java Applets) do not provide means for any kind of temporal specification, the WebCon prototype does not support the HMT temporal design step. In contrast to other limitations of the current WebCon prototype, this deficiency cannot be solved unless standards and techniques for the temporal design of WWW applications are available.

HMT Functionality	Description	Prototype Support
General documents	Section 4.3.1	yes
Input documents	Section 4.3.3.4	no
Query documents	Section 4.3.3.1	no
Result documents	Section 4.3.3.2	yes
Result documents special features (query refinement, result set distribution)	Section 4.5.3.2	no
Detail documents	Section 4.3.3.3	yes
Element-groupings	Section 4.3.1	yes
Links (structural and navigational)	Section 4.3.2	yes
Access struct TOC	Section 4.3.2	yes
Access struct Guided Tour	Section 4.3.2	yes
Access struct Guided TOC	Section 4.3.2	no
Access Struct Slide Show	Section 4.3.2	no
Submitter	Section 4.3.2	no
Authorization (document level)	Section 4.4.2.1	yes
Authorization (grouping level)	Section 4.4.2.2	yes
Temporal design	Section 4.5.4	-
Layout design	Section 4.6	yes

Table 6-1: HMT functionality of the WebCon prototype

Concerning the HMT document types, only the three most important ones are supported by the current prototype: general document types, result document types and detail document types. Input and query document types will be included into the next version of WebCon. The functionality of result document types is limited to the basic presentation features; query refinement and distributed result sets cannot be used yet. All other HMT design primitives

like element-groupings, attributes, adds, links and access structs are supported by the current version. This refers also to specific elements like access structs with conditional reference or Guided Tours.
Support for users and roles is available and allows both specifying access restrictions for whole documents and defining adaptive contents. A session management component ensures that the user only has to identify himself once and not for every request.
Table 6-1 shows a summary of the HMT functionality and its coverage by the current WebCon toolkit. The label “yes” identifies already supported features, “no” marks future components and “-“ denotes issues that cannot be addressed with current WWW techniques.

6.3 Open Implementation Issues

The current implementation of WebCon is the first prototype of a HMT CASE-tool. It has helped to identify weak points in the early versions of the HMT meta schema and supported the development of a consistent and efficient relational representation for HMT schemas. Besides improvements regarding efficiency and maintainability of the source code, future versions of WebCon will also provide support for additional HMT functionality not covered so far. The following paragraphs identify the primary development goals for the next versions of WebCon.

Query and insert document types

The main target of the WebCon future development efforts is the support for HMT query and input document types. Right now, only general, result and detail document types can be generated. Query and input document types can rely on an existing basic document framework, which contains components for every HMT design primitive, for authorization aspects, layout, meta schema access and structural caching. The result document type has to be extended in order to be able to handle distributed result sets and query refinement as described in section 4.3.3.2.

Graphical administration interface

The current, browser-based administration tool is very useful for mobile maintenance of an application and for quick administration by experienced users. It is currently being extended to support the entire HMT functionality including user and role administration.
For inexperienced users or very large applications, however, a graphical HMT development interface would be the best solution. It should offer a toolbox with all HMT design primitives and an editor with drag-and-drop functionality for designing conceptual HMT schemas. The other design steps of HMT would be presented as different views on the conceptual schema. In order to be as platform independent as the page generation components of WebCon, the graphical HMT development interface should be implemented completely in java, too.

Version management

Especially for large-scale applications with several administrators, the aspect of version management becomes an important issue. It must be possible to restore earlier versions of the

application in case of errors or misconfiguration. Since all information about the application is stored within the database system, traditional version management systems like RCS or CVS cannot be used with WebCon.
The solution for this problem is the introduction of an additional version attribute in all HMT meta tables. This version attribute has to become part of the primary key thus allowing the storage of several versions of an entry. The CASE-tool then has to provide a possibility to save the current configuration under a certain version number and to restore an older version. If the page generation component is extended to use a specified version of the meta schema configuration, an existing application can stay online when the administrator generates a new version. After having tested this new configuration, the public application can easily be switched by specifying the new version number.

Performance analysis

After having finished the first prototype with full HMT support, a detailed performance analysis has to be made, which refers to several aspects. The code of the page generation component has to be optimized regarding the usage of objects, methods or variables. The schema-cache has to be analyzed in order to identify a measure for the ideal cache size dependent from the size of the HMT meta schema, the physical memory of the host and the replacement strategy used. Other factors influencing the overall performance of the system are the servlet engine used (meanwhile various servlet engines like JRun, JServ or ServletExec are available) and its configuration regarding servlet loading or usage of threads.

Offline page generation

In addition to the online page generation of the current WebCon CASE-tool, future versions should also support offline page generation. This can be useful for optimizing the performance of large applications with rather static data source or for creating offline versions of the application, for example on cdrom.
Therefore, the WebCon toolkit has to be extended by a static-page-generation-component mapping the URLs of dynamic pages to static references. This component should also contain algorithms for incremental administration, because otherwise the whole application has to be re-generated every time the structure or the contents of the application change. An algorithm describing the incremental maintenance of hypertext views has been described in [Som00].

6.4 Summary

This chapter has described the first prototype of a HMT CASE-tool called WebCon. We first discussed the system architecture, which is a server side concept using an arbitrary web server, java servlets and JDBC for the database connection. User authentication, authorization and session management are handled by the WebCon CASE-tool. This architecture provides maximum flexibility regarding web server, platform and database system.

The second subsection presented a description of the most important implementation aspects regarding the WebCon prototype. After a short view on the system configuration, the aspect

of user authentication has been discussed. For that purpose, a special kind of general document can be defined that assigns each user a unique session-id. In HMT, authorization can be used for both access restriction and adaptive documents.
The page generation component uses special caching strategies: the entity-cache and the schema-cache. While the entity-cache is used for minimizing the requests to the data source, the schema-cache reduces calls to the HMT repository.
The browser-based administration component of WebCon allows maintaining the entire application using an ordinary web browser, thus supporting mobile administration.
Finally, the HMT functionality supported by the current version of WebCon has been specified exactly.

The last subsection has identified the primary future development goals of WebCon, which are support for query and insert document types, a graphical design interface, version management, performance analysis and offline page generation.

As a summary, Figure 6-5 gives an overview of all current and some future components of the WebCon CASE-tool. Already existing components are drawn with a gray background, future packages have a white background.

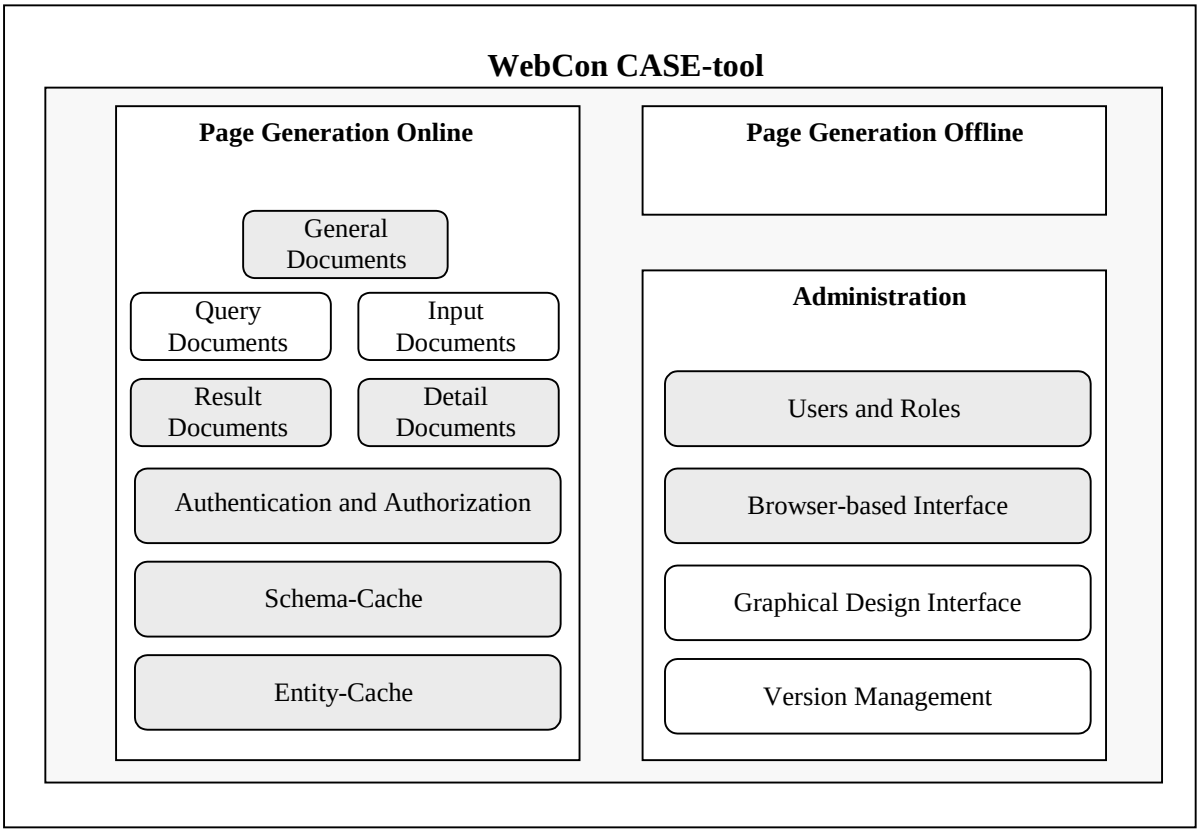

Figure 6-5: overview of the WebCon components

CHAPTER 7

CONCLUSION AND FUTURE WORK

7.1 Conclusion

After hypertext and hypermedia systems have become widely available and accepted, the aspect of database support for these kinds of applications is paid more and more attention to. While modeling methodologies are known in most traditional areas of software development like database design or object-oriented design, the area of database-driven hypermedia applications still lacks a comprehensive design framework covering all relevant issues. The main deficiencies of existing modeling approaches concern the following aspects:

- Support for interactive user interfaces
  In addition to interfaces for “passive” information presentation, a lot of applications require also interactive user interfaces. Examples are search interfaces for directly querying the data source or input interfaces for manipulating the application data.
- Flexible hypermedia formats
  The very short life cycles of standards and techniques in the WWW demand great flexibility regarding the hypermedia target format of an application. This refers not only to the integration of new language extensions or versions, but also to the possibility of switching between completely different hypermedia formats with only minimal efforts.
- Access control
  More and more hypermedia applications contain not only information for the public, but also internal data that should be accessed only by selected users. Not supporting this aspect in the modeling methodology used increases maintenance complexity and error probability of the application
- Personalized information presentation
  Personalization of hypermedia information systems will be a central issue for a variety of future applications. Without support by the application design and maintenance process, personalization can hardly be achieved at reasonable costs.
- Temporal design
  Multimedia elements like graphics, sound or small video sequences become more and more popular as transmission rates in the internet are increased steadily. Although current web standards do not address the issue of temporal relations so far, a comprehensive hypermedia modeling methodology should already provide support for the temporal design of the application.

The Hypermedia Modeling Technique (HMT) has been developed to close this gap. Based on well-known approaches like the Entity-Relationship model (ER) or the Relationship Management Methodology (RMM), HMT offers a six-step design methodology for the development of database-driven hypermedia applications. It is the first hypermedia design framework to address all aspects relevant for typical applications in that area.

The first step in the HMT design process is the requirements analysis. It covers aspects like the definition of the application domain, identification of the intended users, and specification of the system's functionality and usage. The second design step involves the creation of an ER schema of the application domain, which has been identified during requirements analysis. For that purpose, HMT relies on the basic ER model and needs no special extensions. If the hypermedia application has to be built upon an already existing database, these first two design steps can be omitted.

The core hypermedia design starts with the third design step, which is called conceptual hypermedia design. HMT offers a set of design primitives for specifying the contents of hypermedia documents (referred to as information clustering) and the navigation paths between them (referred to as navigational design). Five kinds of document types can be used for information clustering:

- General document types contain no information from the application database, but might include links or access structures leading to other documents. They are typically used to model entry points for the application (homepage, starting page).
- Result and detail document types are used for presenting database contents. While result documents display a list of entities (for example, the results of a query), detail documents show exactly one entity.
- Query document types allow specifying complex search interfaces for the underlying database. These search interfaces can cover several entity types and require no explicit query specification.
- Input document types help to build interfaces for inserting data into the database.

Access primitives like links, submitters or access structs are used to model navigational paths between related portions of information.

The conceptual hypermedia schema is used as the basis for the fourth design step, which is the authorization design. HMT uses role-based access control for that purpose and allows access restrictions on two different levels. If whole document types are restricted, the corresponding documents can only be viewed by users having the required role. If parts of document types are restricted, the users will see only those parts of the document they have access permissions for. Using this technique, adaptive documents can be designed very easily.

The logical hypermedia design is the fifth step of the HMT design process. It addresses aspects like the order and labels of the elements within a document, the meta types of the

attributes and the temporal design of the application. A set of seven temporal relations has been introduced for specifying the dependencies between the elements of a document type concerning time.

The last step in the design process is the layout design step. It addresses all aspects regarding the layout of the application (for example background image, text color or font) and depends heavily on the actual hypermedia platform used. Thus it is impossible to specify a general model for this design step. HMT leaves it up to an actual CASE-tool to provide support for this design step and a certain hypermedia platform. Since only the last step in the HMT design process depends on an actual hypermedia platform, applications designed with HMT can easily be created for a variety of different hypermedia formats like HTML, XML or PDF.

For the implementation of a HMT CASE-tool, HMT schemas have to be mapped to a computable format. For that purpose, we have introduced a relational HMT repository that resides in the DBMS containing the application data. The schema of this repository (called HMT meta schema) consists of 14 meta tables and more than 40 relationship types storing all information arising from the HMT design process. This repository can be considered as the minimum meta schema needed for mapping HMT schemas, actual implementations might provide additional meta tables and relationship types.

The HMT approach has been verified by the implementation of a prototype CASE-tool for the generation of World Wide Web applications. It is based on java servlet technology and JDBC database access in order to be as independent from platform and database system as possible. The prototype consists of a full-featured page generator for the creation of result and detail documents and a browser-based administration component for maintaining the HMT repository. Additional components addressing currently missing HMT functionality have been identified and will be part of future developments.

7.2 Future Work

Extensions and new developments in the near future will concern mainly the WebCon implementation. As already identified in the previous chapter, efforts in the areas of input and query pages, the graphical design interface or performance tuning are the most important tasks. Other extensions concern the generation of different hypermedia formats like XML or PDF, the support for analyzing access rates and user behavior, and the materialization of HMT pages.
A very difficult aspect will be the integration of (a subset of) the HMT temporal design functionality into the WebCon toolkit, because the World Wide Web offers no standard supporting that issue so far. For some specific elements like simple HMT Slide Shows, a component for the automatic generation of animated gif images or java applets could be integrated into WebCon. However, supporting the entire functionality of the HMT temporal design step will be the greatest challenge for future CASE-tools.

Regarding the Hypermedia Modeling Technique itself, several topics for future research can be identified:

- Support for extended ER models
  The current version of HMT relies on the basic ER model as described by Chen. Future versions might be extended to support advanced concepts like isa-hierarchies or complex attributes. While some concepts require only additional mapping rules (for example complex attributes), others will lead to the introduction of new or extended design primitives (for example isa-hierarchies).
- Non ER-based design of the application domain
  The HMT design process might be extended to support other techniques than ER for the second design step. For example, the application domain might as well be specified using the Unified Modeling Language (UML). For primitive UML schemas, most of the HMT conceptual design primitives could remain unchanged or require only little modifications, and the remaining design steps (four to six) would not be affected by that extension.
- Specification of advanced presentation characteristics
  The Amsterdam Hypermedia Model provides the abstract channel-concept for the presentation of hypermedia contents. Each channel can be assigned a number of parameters specifying default presentation characteristics for the current media type, for example sound volume, frame rate or scaling factors for graphics. The Hypermedia Modeling Technique might be extended to support a similar concept for the specification of advanced hypermedia presentation characteristics.
- Support for distributed databases
  Currently HMT supports the design of hypermedia applications based on a single DBMS. Little modifications like, for example, the use of database-specific prefixes for entities or attributes could allow the creation of applications using distributed data sources. The corresponding extensions to the HMT meta schema are trivial, but performance considerations will have to get a higher priority for CASE-tool implementations.

REFERENCES

[Ab00] abayfor-online: the information system of the association of bavarian research cooperations.
URL: http://www.abayfor.de/

[ACH+93] Arens, Y., Chee, C., Hsu, C., Knoblock, C.: "Retrieving and Integrating Data from Multiple Information Sources". In International Journal of Cooperative Information Systems (IJCIS), 2(2), pp. 127-158, 1993

[All83] Allen, J.F.: "Maintaining Knowledge about temporal intervals". In *Communications of the ACM* 1983/26, Vol. 11, pp. 832-843

[All99] Allaire Corporation: "ColdFusion 4.5 White Paper". White paper, 1999
URL:
http://www.allaire.com/Documents/Objects/WhitePaper/CF45WhitePaper.doc

[Alt00] Alta Vista ™ search engine
URL: http://www.altavista.com

[AMM97] Atzeni, P., Mecca, G., Merialdo, P.: "To Weave the Web". In *Proceedings of the 23rd International Conference on Very Large Databases (VLDB'97)*, Athens,1997, pp. 206-215

[AMM98] Atzeni, P., Mecca, G., Merialdo, P.: "Design and Maintenance of Data-Intensive Web Sites". In *Proceedings of the International Conference on Extending Database Technology (EDBT)*, Valencia, 1998, pp. 436-450

[Apa00] Apache web server
URL: http://www.apache.org/

[AS00] Adobe Systems: "PDF Reference (Second Edition)". Addison-Wesley Publishing Company, 2000

[ASP00] Microsoft "Active Server pages Tutorial".
URL : http://msdn.microsoft.com/workshop/server/asp/asptutorial.asp

[BCN93] Batini, C., Ceri, S., Navathe, S.B.: "Conceptual Database Design: an Entity-Relationship Approach". Benjamin and Cummings Publ. Co., Menlo Park, 1993

[Ber99] Berg, C.: "Advanced Java 2 Development for Enterprise Applications (Second Edition)". Prentice Hall, New Jersey, 1999

[Blu00] Bluestone Software Inc.: “Bluestone Software Total-e-Server White Paper”. White paper, 2000
URL: http://www.bluestone.com/downloads/pdf/Total-e-Server_WP_Final.pdf

[Boe76] Boehm, B.: “Software engineering”. In *IEEE Transactions on Computers 25*, pp. 1226-1241, 1976

[CGH+94] Chawathe, S., Garcia-Molina, H., Hammer, J., Ireland, K., Papakonstantinou, Y., Ullman, J., Widom, J.: “The TSIMMIS project: Integration of heterogeneous information sources”. In *Proceedings of the 10th Meeting of the Information Processing Society of Japan (IPSJ)*, Tokyo, 1994

[CGI00] The World Wide Web Consortium (w3c): “CGI”
URL: http://www.w3c.org/CGI/

[Che76] Chen, P.S.: “The Entity-relationship Model – Towards a Unified View of Data”. In *ACM TODS*, Vol. 1, No. 1, 1976.

[Chi00] Chimera Software: HotSQL 1.3
URL: http://www.chimerasoft.com/hotsql/

[Cod90] Codd, E.F.: “The relational model for database management : version2”. Addison-Wesley Publishing Company, Inc., 1990

[DB00] Deutsche Bahn AG: Online Fahrplanauskunft.
URL: http://www.bahn.hafas.de

[DI95] Diaz, A., Isakowitz, T.: “RMCase: Computer-Aided Support for Hypermedia Design and Development”. In *Proceedings of the 1995 International Workshop on Hypermedia Design (IWHD’95)*, 1995

[Dic97] Dicken, H.: “JDBC Internet-Datenbank-Anbindung mit Java”. Thomson Publishers, Bonn, 1997

[DM00] Danesh, A., Motlagh, K.A.: “Mastering ColdFusion 4.5”. Sybex, 2000

[EN94] El Masri, R.A., Navathe, S.B.: “Fundamentals of Database Systems”. Benjamin and Cummings Publ. Co., Menlo Park, second edition, 1994

[Exp00] ExperTelligence Inc.: WebBase
URL: http://www.webbase.com/

[FFK+97] Fernandez, M., Fiorescu, D., Kang, J., Levy, A., Suciu, D.: "STRUDEL: A Web-site Management System". In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'97)*, Tucson, Arizona, 1997

[GB98] Gavrila, S.I., Barkley, J.F.: "Formal Specification for Role Based Access Control User/Role and Role/Role Relationship Management". In *Proceedings of the 3rd ACM Workshop on Role-based Access Control*, Fairfax, VA, 1998, pp. 81-90

[GMP95] Garzotto, F., Mainetti, L., Paolini, P.: "Hypermedia Design, Analysis, and Evaluation Issues". In *Communications of the ACM Special Issue*, 38(8), pp. 74-86, 1995

[Goo99] Goodwill, J.: "Developing Java Servlets". SamsPublishing, 1999

[GPS93] Garzotto, F., Paolini, P., Schwabe, D.: "HDM- A Model-based Approach to Hypermedia Design". In *ACM Transactions on Information Systems*, 11, 1 (1993), pp. 1-26.

[GPS93] Garzotto, F., Paolini, P., Schwabe, D.: "A Model-Based Approach to Hypertext Application Design". In *ACM Transactions on on Information Systems*, 11(1), pp. 1-26, 1993

[HBR94] Hardman, L., Bulterman, D.C.A., Rossum, G.: "The Amsterdam Hypermedia Model". In *Communications of the ACM*, 1994/2, Vol.37, pp. 50-62

[Hos97] Hoschka, P.: "Synchronized Multimedia Integration Language". http://www.w3.org/TR/WD-smil-971109

[HS94] Halasz, F., Schwartz, M.: "The Dexter Hypertext Reference Model". In *Communications of the ACM*, Feb. 1994, Vol. 37, No. 2, pp. 30-39

[Htm00] The World Wide Web Consortium (w3c): "HTML" URL: http://www.w3c.org/MarkUp/

[IBM00] IBM: Web Sphere Application Server URL: http://www-4.ibm.com/software/info/websphere/

[IKK97a] Isakowitz, T., Kamis, A., Koufaris, M.: "Extending the capabilities of RMM: Russian Dolls and Hypertext". In *Proceedings of the 30th Annual Hawaii International Conference on System Sciences*, 1997.

[IKK97b] Isakowitz, T., Kamis, A., Koufaris, M.: "Reconciling Top-Down and Bottom-Up Design Approaches in RMM". In *Proceedings of the Workshop on Information Technologies and Systems (WITS97)*, Atlanta, GA, Dec. 1997.

[IKK98] Isakowitz, T., Kamis, A., Koufaris, M.: “The Extended RMM Methodology for Web Publishing“. Working paper IS-98-18, available at http://rmm-java.stern.nyu.edu/rmm/papers/RMM-Extended.pdf

[Inf00] Informix Corporation: “Creating Web-Enabled Database Applications Using the Informix Web Datablade Module”. White paper, 2000
URL: http://www.informix.com/informix/whitepapers/createweb_wp.pdf

[ISB95] Isakowitz, T., Stohr, E., Balasubramanian, P.: “RMM: A Methodology for the Design of Structured Hypermedia Applications”. In *Communications of the ACM*, 38(8), pp. 34-44.

[Jou97] Jourdan, M., et al.: “Authoring Environment for Interactive Multimedia Documents”. OPERA project, INRIA Rhone-Alpes, Montbonnot, France
http://opera.inrialpes.fr/OPERA

[KK92] Kendall, K.E., Kendall, J.E.: “Systems Analysis and Design”. Prentice-Hall, New Jersey, 1992.

[Loe98] Loeser, H.: “Techniken für Web-basierte Datenbankanwendungen: Anforderungen, Ansätze, Architekturen“. Informatik Forschung und Entwicklung, 13(4), pp.196-216, 1998

[LS97] Lupu, E., Sloman, M.: “Reconciling Role Based Management and Role Based Access Control”. In *Proceedings of the 2nd ACM RBAC Workshop*, Fairfax, VA, USA, Nov. 1997, pp. 135-141

[Lyc00] Lycos ™ search engine
URL: http://www.lycos.com

[MG+87] Malone, T.W., Grant, K.R., Turbak, F.A., Brobst, S.A., Cohen, M.D.: “Intelligent Information-Sharing Systems”. In *Communications of the ACM*, Vol. 30 (5), pp. 390-402, 1987

[Mic00] Microsoft: “ISAPI Programming, Microsoft Interactive Developer”.
URL: http://www.microsoft.com/Mind/0197/ISAPI.htm

[Mli00] M-Line: Materialwissenschaftliches Online Informationssystem.
URL: http://www.format.mwn.de

[MMD] Macromedia: Director 4.0 *User’s Guide*.

[Mof98] Moffet, J.D.: “Control Principles and Role Hierarchies”. In *Proceedings of the 3rd ACM Workshop on Role-based Access*, Fairfax, VA, 1998, pp. 63-69

[Net00] Netscape: “Netscape API Functions”.
URL: http://home.netscape.com/servers/index.html

[Net99] Netscape Support Documentation: “Persistent Client State – HTTP Cookies”.
URL: http://www.netscape.com/newsref/std/cookie_spec.html

[NO93] Nyanchama, M., Osborn, S.: “Role-Based Secutity, Object Oriented Databases & Separation of Duty”. In *ACM SIGMOD Record*, Vol. 22, No. 4, December 1993

[NO96] Nyanchama, M., Osborn, S.: “Modeling mandatory access control in role-based security systems”. In *Database Security VIII: Status and Prospects*, Chapman-Hall, 1996

[Odb00] Open Database Internet Connector
URL: http://www.iodbc.com

[Op96] Open Market: “FastCGI: A High-Performance Web Server Interface”. Technical white paper, 1996
URL: http://fastserv.name.net/whitepapers/gcgi-whitepaper.shtml

[Ora00a] Oracle Corporation: “Oracle Internet Application Server 8i (Oracle iAS)”. Technical white paper, June 2000
URL: http://technet.oracle.com/products/ias/pdf/ias_technical_WP.pdf

[Ora00b] Oracle Corporation: “An Overview of iAs Forms Services Architecture”. Technical white paper, June 2000
URL: http://technet.oracle.com/products/forms/pdf/275632.pdf

[Per00] Pervasive Software Inc.: “Tango Enterprise White Paper”. White paper, 2000
URL: http://www.pervasive.com/documentation/docs/w_tango.doc

[RBP+91] Rumbaugh, J., Blaha, J., Premerlani, W., Eddy, F., Lorensen, W.:
“Objectoriented Modeling and Design”. Prentice-Hall, New Jersey, 1991

[RG00] Ratschiller, T., Gerken, T.: “Web Application Development with PHP 4.0“. New Riders Publishing, 2000

[Ris00] Rischpater, R.: “Wireless Web Development”. Apress, 2000

[RJM+93] Rossum, G., Jansen, J., Mullender, K.S., Bulterman, D.: “A Presentation for Portable Hypermedia Documents”. In *Proceedings of the ACM Multimedia ´93*, pp.183-188

[Ro00] Rational Software Corporation: “Rational Rose – a Rational Suite product”. URL: http://www.rational.com/products/rose/index.jtmpl

[San96] Sandhu, R.S.:”Role hierarchies and constraints for lattice-based access controls”. In *Proceedings of the Fourth European Symposium on Research in Computer Security*, Springer-Verlag, Rome, Italy, 1996

[SCF+96] Sandhu, R., Coyne, E.J., Feinstein, H.L., Youman, C.E.: “Role-Based Access Control Models”. In *IEEE Computer*, 29(2), Feb. 1996, pp. 38-47

[SDK96] Song, J., Kim, M.Y., Ramalingam, G.: “Interactive Authoring of Multimedia Documents”. *Research Report RC 20369*, T.J. Watson Research Center, IBM Research Division Yorktown Heights, NY

[See00] Seehafer, R.: “Dynamische Generierung von Webseiten aus HMT-Modellen”. Diploma thesis, Technical University of Munich, 2000

[SM98] Sandhu, R., Munawer, Q.: “How to do Discretionary Access Control using Roles”. In *Proceedings of the 3rd ACM Workshop on Role-based Access Control*, Fairfax, VA, 1998, pp. 47-54

[Som00] Sommer, U.: “Integriertes Management großer Web-Sites auf der Basis datenbankbasierter Modellierungskonzepte“. PhD thesis, Technical University of Munich, 2000

[SRB96] Schwabe, D., Rossi, G., Barbosa, S.: “Systematic Hypermedia Design with OOHDM”. In *Proceedings of the ACM International Conference on Hypertext (Hypertext 96)*, Washington, Mar. 1996.

[SSW80] Scheuermann, P., Scheffner, G., Weber, H.: “Abstraction Capabilities and Invariant Properties Modeling within the Entity-Relationship Approach”. In *Entity-Relationship Approach to Systems Analysis and Design*, Elseveier, Amsterdam, 1980, pp. 121-140.

[Sun00] Sun Microsystems Inc.: “NetDynamics Application Server5.0”. White paper, 2000
URL: http://www.netdynamics.com/products/nd50_wp.pdf

[Syb00] Sybase: “PowerBuilder 7.0 – The bridge from client/server to Web and distributed applications”. White paper, 2000
URL: http://www.sybase.com/content/1002992/pb7ds.pdf

[SZ99] Sommer, U., Zoller, P.: “WebCon: Design and Modeling of Database Driven Hypertext Applications”. In *Proceedings of the 32nd Annual Hawaii International Conference on System Sciences*, 1999.

[TB79] Thomas, E., Biddle, B.: “The Nature and History of Role Theory”. In *Role Theory: Concepts and Research*, Krieger Publishing, 1979

[Teo94] Teorey, T.J.: “Database Modeling & Design”. Morgan Kaufmann Publishers, Inc., San Francisco, 1994.

[Tha91] Thalheim, B.: “Foundations of Entity-Relationship Modeling“. In *Annals of Mathematics and Artificial Intelligence*, 1991

[Tra00] TransAction Software GmbH: “TransBase Relational Database System Version 5.11 – System and Installation Guide”. Munich, 2000

[Tro98] De Troyer, O.: “Designing Well-Structured Websites: Lessons to be learned from Database Schema Methodology”. In *Proceedings of the 17th International Conference on Conceptual Modeling (ER’98)*, Singapore, 1998

[Ttm00] TTM-Line: Traffic and Transport Management Online.
URL: http://ttmline.forwiss.tu-muenchen.de

[TYF86] Teorey, T.J., Yang, D., Fry, J.P.: “A Logical Design Methodology for Relational Databases Using the Extended Entity-Relationship Model”. In *ACM Computing Surveys* 18,2 8June 1986, pp. 197-222.

[Vos91] Vossen, G.: “Data Models, Database Languages and Database Management Systems”. Addison-Wesley Publishers, 1991

[WCO00] Wall, L., Christiansen, T., Orwant, J.: “Programming Perl (3rd edition)”. O’Reilly & Associates, 2000

[Wei82] Weinberg, G.M.: “Rethinking Systems Analysis and Design”. Little Brown and Company, Inc., Boston, 1982

[Xml00] The World Wide Web Consortium (w3c): “XML”
URL: http://www.w3c.org/XML/

[Yah00] Yahoo ™ search engine
URL: http://www.yahoo.com

[Yu97] Yu, J.: “A simple, intuitive hypermedia synchronization model and its realization in browser/java environment”. Technical Note 1997-027, Digital Equipment Corporation Systems Research Center, Palo Alto, CA, October 1997

[Zol00] Zoller, P.: “HMT: Modeling interactive, adaptive Hypermedia Applications”. To appear in : *Siau, K., Rossi, M. (editors): Modelling Methodologies for the next Millenium*, Idea Group Publishing, autumn 2000

[Zol96] Zoller, P.: “Anbindung von Datenbanksystemen an das World Wide Web”. Diplomarbeit, Technical University of Munich, 1996

[ZS98] Zoller, P., Sommer, U.: “WebCon: A Toolkit for an Automatic, Data Dictionary Based Connection of Databases to the WWW”. In *Proceedings of the 1998 ACM Symposium on Applied Computing*, Atlanta, 1998, pp. 706-711

APPENDIX A LIST OF DEFINITIONS

APPENDIX B LIST OF FIGURES

APPENDIX C LIST OF TABLES