ROMA TRE

Università degli Studi di Roma Tre
Dipartimento di Informatica e Automazione
Via della Vasca Navale, 84 – 00146 Roma, Italy.

Design and Maintenance of Data-Intensive Web Sites

Paolo Atzeni†, Giansalvatore Mecca‡, Paolo Merialdo†

RT-DIA-25-1997 **June 1997**

† Dipartimento di Informatica e Automazione
Università di Roma Tre
00146 Roma – Italy

‡ Università della Basilicata,
D.I.F.A. – via della Tecnica, 3
85100 Potenza, Italy

http://poincare.inf.uniroma3.it:8080

ABSTRACT

Many Web sites include significant and substantial pieces of information, in a way that is often difficult to share, correlate and maintain. In many cases the management of a Web site can greatly benefit from the adoption of methods and techniques borrowed from the database field. This paper introduces a methodology for designing and maintaining large Web sites based on the assumption that data to be published in the site are managed using a DBMS. We see the process of designing the site as the result of two intertwined activities: the *database design* and the *hypertext design*. Each of these is further divided in a *conceptual design* phase and a *logical design* phase, based on specific data models. A new logical data model, called ADM, is used to describe the structure of a Web hypertext. It is page-oriented, in the sense that the main construct is the one of page-scheme, providing an intensional description of a class of pages in the site. Based on the ADM scheme of the site, we introduce a language, called PENELOPE, that allows to automatically generate HTML pages starting from the database content. PENELOPE is also able to correlate different pages in a complex hypertext using a suitable URL invention mechanism to guarantee reference integrity. ADM and PENELOPE strongly support site maintenance: the first provides a concise description of the site structure; it allows to reason about the overall organization of pages in the site, in order to evaluate the effectiveness and efficiency of the chosen structure, and possibly to restructure it; at the same time, PENELOPE alleviates the burden of managing HTML files by hand, and guarantees link consistency in presence of updates and reorganizations.

Contents

List of Figures

1 Introduction

Because of the popularity of the World Wide Web (Web), the need to organize large amounts of data in hypertextual form is increasing. In fact, since the Web is becoming a major computing platform and a uniform interface for sharing data, many organizations have found interest in delivering information through the Web, both in Internet and in intranet environments. Because of this growth of interest in the Web, many Web sites now contain valuable pieces of information, that can represent a precious resource for their owners. Thus, great attention needs to be devoted to their design and maintenance. In fact, whenever a site is to contain significant amounts of data, its design becomes a complex process that involves at least two aspects. On the one hand, there are features of the design process that strictly concern data management; this is especially true when it is necessary to manage large collections of data, coming from databases and corporate information systems. On the other hand, the hypertext structure should be carefully designed in order to provide a page organization to be effectively browsed by the users. Clearly, a careful design needs to properly coodinate this two aspects.

To support the process of designing these large, data-intensive Web sites, several methodologies (e.g. [18, 22, 28]) have been recently proposed in the context of hypermedia design. These methodologies represent a first step towards the solution of the problem. They provide specific data models and methodological steps to assist in the process of designing a hypermedia application.

However, we believe that the design process must also support the activity of *maintaining* sites. Searching any of the popular Web index servers, such as Altavista [1], it can be seen that a large number of sites, after being established, are almost completely abandoned, that is, they present numerous errors and inconsistencies due to poor maintenance; also, their information is often obsolete. In many cases, this is due to the fact that managing large collections of files, the HTML pages, is an activity that does not scale up well: due to the URL-based identification mechanism used in HTML, missing pages and inconsistencies in links often occur. The problem is even more apparent when the site undergoes a major *restructuring*: all URLs and page structures have to be changed in a coherent way—if this has to be done manually a tremendous effort is needed and errors are likely to be introduced. We believe that the site designer should be freed from the burden of generating and maintaining HTML pages; for example, once the overall structure of the hypertext has been decided based on the underlying database resources, pages should be *automatically* generated; moreover, the system should also take care of managing URL references, enforcing consistency even in the presence of restructurings.

This paper presents the ARANEUS *Web Design Methodology*, a thorough and systematic design process to organize and maintain large amounts of data in a Web hypertext. Our approach builds on known hypermedia design methodologies [18, 22, 28], and extends them in several ways, in order to automate the implementation and maintenance of the site and to allow for integration or interaction with other information systems. In fact, although many ideas developed in this paper can be considered of general applicability, we mainly focus on the case in which data to be published in the site are kept in a (relational or object-oriented) database. In fact, database management systems provide robust technology for managing large collections of data in a flexible and efficient way; we thus aim at leveraging this technology to improve Web management and maintenance. The database may be either pre-existing, or designed together with the Web site.

Thus, the overall methodology is based on two well distinguished and yet tightly interconnected processes: the *database design* process and the *hypertext design* process. For each of these processes, we distinguish a *conceptual* and *logical* design phase, in the spirit of databases [11]. This allows to isolate system features from higher-level perspectives, and facilitates the restructuring process.

Each hypertext design phase makes use of a suitable data model; the hypertext conceptual

design is based on a data model called *Navigation Conceptual Model* (NCM), inspired by *RMM* [22]; similarly, the hypertext logical design is based on the ARANEUS *Data Model* (ADM) [10]. The use of a specific *logical* data model for Web hypertexts is a distinctive feature of our approach: differently from other models, ADM allows to describe the structure of a site based on *page types* and links, thus providing a concise representation of the hypertext organization. Moreover, once the ADM scheme of a site has been derived, a specific language, called PENELOPE [10], can be used to map database structures to hypertextual structures and automatically generate HTML pages based on the database content. PENELOPE statements specify how pages should be built starting from database tables; it generates a complex hypertext, taking care of assigning URL to pages and enforcing consistency between references.

We consider ADM and PENELOPE important tools for maintaining a site: in fact, the first allows to give a logical description of the site structure; this has several consequences: first, by carefully isolating logical features from physical ones, we introduce a form of *hypertext data independence*, similar to the one usually advocated for relational databases; moreover, the hypertext logical scheme allows to reason about the overall organization of pages in the site, in order to evaluate the effectiveness and efficiency of the chosen structure, and possibly to restructure it. At the same time, PENELOPE alleviates the burden of managing HTML files by hand in presence of updates and reorganizations. In this way, the designer can concentrate on the hypertext organization and use PENELOPE to guarantee that the site is consistently updated.

This work is part of the ARANEUS Project [2], currently under development at Università di Roma Tre. ARANEUS aims at developing tools for mananaging Web data in the spirit of databases. In this respect, the methodology presented in this paper fits in a larger framework, whose goal is to provide tools and techniques supporting the development of Web-based cooperative applications, i.e., applications that analyze and integrate data coming from different Web sites.

The rest of the paper is organized as follows. Section 2 contains a general description of the methodology and a comparison with related literature. The subsequent sections are devoted to the presentation of the various steps of the methodology. Section 3 briefly recalls relevant issues related to the database conceptual and logical design activities, and introduces the University Department Web site example we will refer to throughout the paper. Section 4 introduces NCM and the hypertext conceptual design phase; this is the basis for the hypertext logical design phase, based on ADM, discussed in Section 5, where also techniques for restructuring a site are presented. In Section 6 we briefly discuss the need for a presentation design phase and in Section 7 concentrate on techniques for automatically generating HTML pages.

2 Overview

The overall methodology is shown in Figure 1. Both the database and hypertext design are in turn divided in a *conceptual design* phase and a *logical design* phase, based on specific data models.

2.1 The ARANEUS Design Methodology: Phases and Models

The starting point for the design process is a database conceptual design phase generating an *Entity-Relationship (ER) conceptual scheme* [11], describing the organization of the underlying domain in an implementation-independent manner. From this, the database logical (and possibly physical) design can be derived based on standard techniques [11, 14]. Note that, if the database is pre-existent, these phases can be omitted, provided a conceptual scheme of the application domain is available; otherwise, a reverse-engineering phase may be necessary in order to obtain a conceptual scheme starting from the existing database.

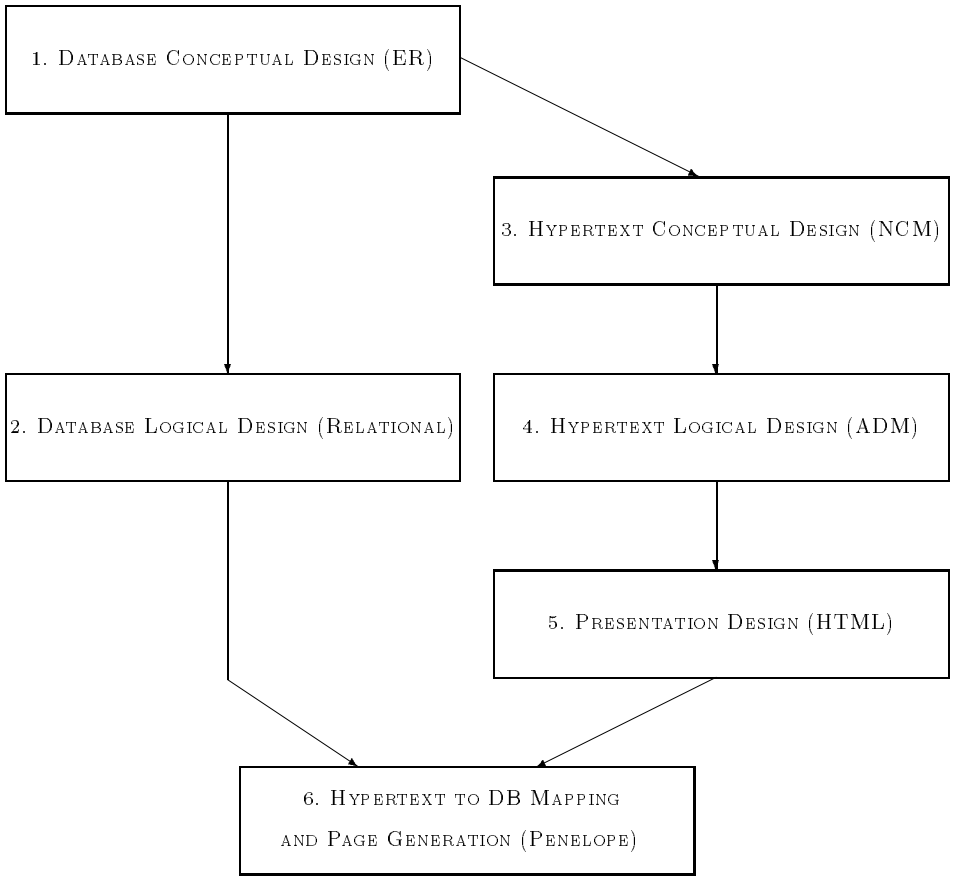


Figure 1: The Araneus Design Methodology

Our approach inherits from RMM [22], in the fact that the ER scheme also provides the basis for hypertext design. However, in order to better isolate the structural properties of the resulting hypertext, we distinguish two different description levels: the *hypertext conceptual level* (phase 3) and the *hypertext logical level* (phase 4). At the conceptual level, we try to abstract the hypertext features, i.e., the hypertext nodes and the paths to navigate between them; this is done using a specific data model, called ncm, inspired by RMM [22]. As a subsequent step, in order to detail the hypertext organization in terms of pages and links, a logical design phase is necessary; this is based on adm, the Araneus Data Model for Web hypertexts.

Finally, once the database and the hypertext logical design are complete, in order to phisically generate HTML pages, the hypertext structure needs to be mapped to the database structure (phase 6); to do this, page attributes are associated with the corresponding attributes coming from tables or views over the database. Penelope can then be used to implement the mapping and generate actual pages based on the database content.

In the rest of this section, we discuss the hypertext design phases in more detail. Note that a specific phase (phase 5) is needed in order to design the presentation, i.e., the layout according to which pieces of information in a page will be displayed. However, in the following we mainly concentrate on data-management issues and do not develop further on this activity.

Hypertext Conceptual Design

The *hypertext conceptual design* aims at describing how application domain concepts are to be organized in hypertextual form; this activity is independent of the physical implementation, and concentrates on two essential aspects: *(i)* deciding which conceptual entities will be part of the hypertext, i.e., will correspond to hypertextual "nodes"; *(ii)* choosing the paths to navigate between entities, i.e., how it will be possible to explore the network of nodes. Moreover, at this level, also the *hypertext access structure* should be described, i.e., how entities are to be aggregated in order to provide a hierarchical organization of concepts to be explored by the user.

To give an example, consider a University Department Web site; suppose you have designed an ER scheme describing entities (`PROFESSOR, COURSE, ROOM, PUBLICATION, RESEARCH-GROUP` ...) and relationships (`PROFESSOR-OFFICE, COURSE-ROOM` ...). When starting the hypertext design, at the conceptual level, one should first consider how concepts will be organized in the site; in this case, one has to decide which of the entities will be represented in the hypertext; in our example, one could think of having `PROFESSOR` and `COURSE` but not `ROOM` (possibly absorbed by the other two). Then, paths through concepts have to be designed, transforming ER undirected relationships into *directed* relationships; for example, one could decide to allow the navigation from `PROFESSOR` to `COURSE` and not vice versa. Finally, entities should be aggregated in order to impose a suitable access structure; for example, `PUBLICATION` and `RESEARCH_GROUP` could be aggregated under a common node called `RESEARCH`, whereas `PROFESSOR` and `COURSE` under `EDUCATION`; then, in turn `RESEARCH` and `EDUCATION` could be aggregated under a node called `DEPARTMENT`, a sort of root concept from which all Department activities can be reached. This results in a conceptual scheme of the hypertext. The data model used at this step is called *Navigation Conceptual Model* (NCM). It is a variant of RMM, the data model proposed in [22].

Hypertext Logical Design

On the other side, the *hypertext logical design* describes how *pages* are to be organized in the Web site; this phase is specific of the Web framework, and aims at detailing the structure of the hypertext as shown by the browser. At this level, we ignore the physical implementation of pages, and concentrate on abstracting their logical features, i.e., relevant pieces of information in the page (e.g., text or images) and their organization (e.g., flat page, list or nested list). To do this, the hypertext conceptual scheme is translated into a logical scheme, based on the ARANEUS *Data Model* (ADM). The main construct of ADM is the one of *page-scheme*; each page-scheme is a type, describing the structure of a set of homogeneous pages in the site. Each page is seen as an object with an URL plus a set of attributes; these can be either *simple*, like text, images, sounds, or *complex*; complex attributes are lists of tuples, possibly nested. Page-schemes are connected using links, used to describe navigations in the site.

The logical design phase consists in mapping the conceptual hypertext scheme to an ADM logical scheme: in this process, entities and aggregations are translated to page-schemes and directed relationships to links. We adopt a two-step approach: *(i)* we first discuss how NCM constructs can be mapped to page-schemes, in order to obtain a first logical skeleton; *(ii)* then, we reconsider such skeleton, and introduce a set of transformations that can be applied on the ADM scheme in order to improve the overall efficiency and effectiveness. The reason for adopting such a composite process is to emphasize the flexibility of the ADM data model in reorganizing the designed hypertext. Clearly, reorganizations may occur in the site lifetime, and usually require a great effort in order to correspondently maintain the system. ADM is a natural framework for reasoning about the actual organization of the site, and provides useful support to these activities.

We consider this translation to the logical level as being very important in this context; in

fact, essential design choices can be made only at this level. To give an example, consider again the Department Web site: starting from the site conceptual scheme, in order to have a complete description of the site structure, it is necessary to describe how *pages* are to be organized; for example, one may think of generating a page for each instance of the `PROFESSOR` entity, whereas a single page could contain a list of all instances of the `COURSE` entity. These choices are crucial in designing the final site organization and can strongly influence both the user interaction with the site, and the performance of the system. In fact, ADM schemes offer a concise representation of the site, based on which the effectiveness and efficiency of the chosen structure can be evaluated, possibly to restructure it.

Hypertext to DB Mapping and Page Generation

In this phase, a mapping between data in hypertext pages and values in the database is established. To give an example, suppose we have a page, called `PROFESSOR_PAGE` for each professor in our hypertext logical scheme, reporting the name and all courses given by that professor; suppose also data about professors is stored in the database in a table called `PROFESSOR` and data about courses in a table called `COURSES`; in order to fill-out the page we need to access data in both tables; more specifically, we need to map the `NAME` attribute in page `PROFESSOR_PAGE` to the corresponding attribute in table `PROFESSOR`, and the list of courses to tuples in table `COURSES`.

Note that, in general, a *view design phase* is necessary to do the mapping. In this phase, views over the database are used to isolate portions of the database relevant for the Web site; in fact, the site is usually built on a specific *external scheme* on the underlying database. Such an external scheme is implicitly chosen during the hypertext conceptual and logical design phases. In essence, we can consider the final hypertext itself as a *hypertextual view* over the original database. This is especially true if data come from a pre-existent database: in this case, data in the hypertext may be organized in a significantly different way with respect to the starting database. As an example, assume in the hypertext we decide to distinguish graduate courses from undergraduate ones; presumably, this distinction is not in the database, and must be introduced by generating two suitable views, one for graduate and one for undergraduate courses, based on the type of the course; then, we can map each list of courses to the appropriate view.

In ARANEUS, the hypertext to database mapping is based on the hypertext and database logical scheme. We associate with each page-scheme in the ADM scheme a set of database tables and views, containing all relevant information for the corresponding pages. Then, each page attribute is mapped to attributes in the associated tables. A specific programming language, called PENELOPE can be used to this end. PENELOPE programs automatically generate HTML pages based on the page-scheme structure and on the associated template. Note that two different choices are possible here: *(i)* pages can be either generated starting from the database, materialized and stored in HTML files, or *(ii)* can be dynamically generated upon request. These two alternatives are sometimes called *push* and *pull* approach, respectively [30]. The first alternative has clear advantages in terms of performance, since it cuts database access costs, but requires to update the hypertext periodically to reflect database changes. PENELOPE supports both push and pull solutions: it can generate and materialize hypertext pages starting from the database content, or be used to dynamically generate pages using CGI-scripts. Moreover, it also allows for intermediate solutions, in which some of the pages, those considered more stable, are materialized, and some others, considered somehow more volatile, are dynamically generated upon request. Note that PENELOPE greatly simplifies site maintenance: on the one hand, updates to the underlying database are directly reflected on the site; on the other side, URLs are kept consistent also in the presence of page insertion or deletion (this is based on the specific URL invention mechanism used by PENELOPE,

borrowed from object-oriented databases [20]).

At the same time, PENELOPE keeps track of the logical organization of the site, in order to generate a highly structured repository, which can be queried using high level query languages. This is important for several reasons, the main one being that data in the site can be effectively used only if they can be extracted and manipulated using high-level tools. Pages generated by PENELOPE are enriched with suitable *meta-information*: in essence, meta-tags are used to document the organization of data inside the page, i.e., the logical page-type. Suppose, for example, that the ADM scheme of our Department contains a page-scheme `PROFESSOR`, describing the structure of professors' home pages. Suppose also these pages have attributes `Name` and `Phone_Number`. Meta-tags are used to mark these relevant pieces of information inside the HTML source; each meta tag is embedded inside a HTML comment, so that it is completely transparent when the page is displayed by a browser. At the same time, however, it greatly facilitates data access: in order to extract from the home page of a Professor the name or the phone-number, it is sufficient to access the page, and then search for the appropriate meta-tags, using an ordinary parser or tools specifically designed for wrapping HTML pages (e.g. [8, 19]) and extract the string between them.

2.2 Related Work

Various methodologies have been recently presented in the context of hypermedia design, including *HDM* [18, 17], RMM [22] and *OOHDM* [28]. All of them introduce models for the description of hypermedia applications, the essential constructs being the ones of *entity*, *link* and *menu*, the latter used to represent access structures.

Both RMM and OOHDM organize the design process in specific phases: *(i) conceptual data design*, i.e., a conceptual description of the domain of interest, based on ER or on object-oriented data models; *(ii) navigational design*, based on specific data models; *(iii) interface design*; *(iv) implementation.* ARANEUS builds on these proposals, with several differences; first, we see the design process as the result of a strict interconnection between a database and a hypertext design activity; thus, we clearly distinguish the database design process from the hypertext design process; moreover, we isolate conceptual aspects of the hypertext from logical aspects, whereas the data models used in [22, 28] mix together conceptual constructs, like the one of *entity* or *directed relationship* with logical (or even "physical") aspects, such as *guided tours* or *indexed tours*; finally, we specifically focus on *maintenance* aspects and try to provide tools supporting site reorganizations.

The distinction between conceptual and logical aspects is essential in our approach. In fact, our Navigation Conceptual Model (NCM) is a variant of RMM, in which some of the constructs (like *guided tours*) are not included. On the contrary, we use a specific page-oriented data model for the description of the site at a *logical* level, which is absent in other proposals. This fact has several consequences. For example, in RMM it is not possible to specify whether each instance of an entity should correspond to a single page or all instances should be in a single page. Moreover, it is difficult to reason about performance issues, since, for example, there is no notion of *form*, a construct very common in Web sites. At the same time, the absence of a concise description of the page structure makes restructurings more difficult.

In this respect, HDM [18, 17] provides a first solution by distinguishing two different design activities: *authoring in the large* and *authoring in the small.* Authoring in the large aims at describing the overall organization and behaviour of a hypermedia application, whereas authoring in the small deals with specific details in the application. This is somehow similar to our distinction between *conceptual* and *logical* level. HDM mainly focuses on authoring in the large, and only some constructs (*node type, frame, slot*) are given for authoring in the small; however, this is not sufficient for a complete description of a page structure, since, for example, there is no complex data type to

model lists or nested lists inside pages.

Differently from the above approaches, which do not support the automatic generation of pages, the organization of our methodology aims at supporting not only the design process, but also the maintenance process: on the one side, the conceptual and logical description of the site represent an essential documentation, based on which the overall quality of the chosen structure can be evaluated, both in terms of effectiveness and performance, possibly allowing for re-organizations; on the other side, PENELOPE guarantees that consistency in the site will be maintained, even in presence of restructurings.

Several commercial database systems (see, for example, [4, 3]) now provide functionalities for the automatic generation of pages. However, they mainly allow for dynamically generating a single page at a time, containing a set of database tuples (the so-called *pull* approach [30]); usually, the skeletons of pages are kept inside the database; pages contain specific HTML tags, which specify that, in order to fill one page, an SQL query is to be run against the database: once the query has been executed, the resulting table is inserted in the body of the page returned to the user. The language PENELOPE described in this paper can be used to this end; however, pull approaches often suffer poor performances, due to the need to open a database connection whenever a page is to be generated; to improve performance, PENELOPE also supports *push* solutions: it can materialize a whole hypertext based on the database content, storing pages in HTML files. Another important difference is that our approach is based on a specific data model, used to describe the hypertext organization at a logical level, whereas the issue of hypertext structure is not addressed in commercial products.

A language for automatic generation of pages for Web sites has been recently proposed in the *STRUDEL* project [15, 16]. This is based on a different approach, in which graphs in the spirit of UnQL [12] are used to describe the structure of a site; thus, there is no conceptual nor logical scheme of the site as intended in this paper.

Other relevant works are concerned with specific aspects of the design process, like the site deployment over the network [27] or the interaction with the site [29]. Both of these proposals use RMM as a design methodology.

3 Phases 1 and 2: Database Conceptual and Logical Design

We assume that data to be published on the site is to be managed using a DBMS. Thus, there is a tight interconnection between designing the hypertext and designing the underlying database. Note that the database can be either pre-existent or to be implemented. If the database is to be explicitly set-up, a conceptual and a logical design phase are needed. These phases are clearly not necessary if an existing database is to be used.

In either case, the starting point for our approach is a conceptual description of the domain of interest, based on an *Entity Relationship (ER)* [11] scheme, which is rather standard in database design; however, the design process could be easily extended to any conceptual data model. The example we shall refer to throughout the paper consists in the design and implementation of (a portion of) a University Department Web site. The corresponding ER scheme, presented in Figure 2,[1] describes information about people and activities in the department. It represents a conceptual description of the application domain that is "neutral" with respect to any actual implementation, based on *entities* and *relationships* among entities. We assume that the ER scheme has been designed based on standard techniques and do not develop on this issue, to concentrate

[1] The conceptual and logical schemes are clearly based on some simplifying assumptions - for example in choosing keys - and consider only a subset of relevant features of a Department database.

on the hypertext design activity.

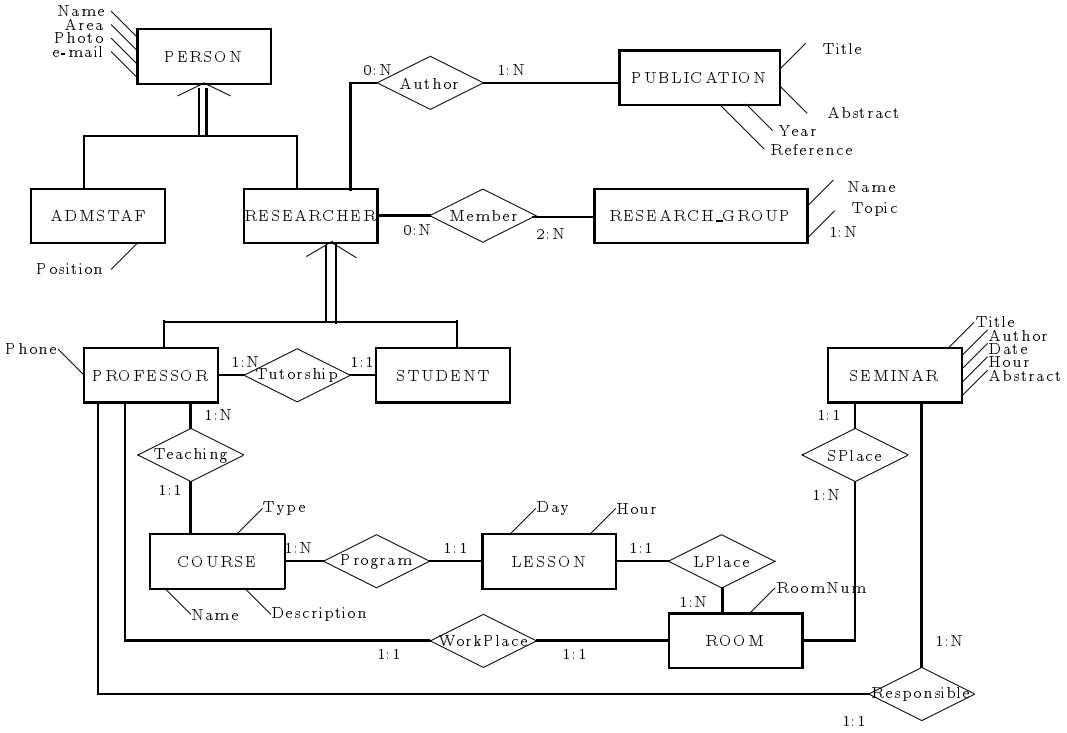


Figure 2: Department ER scheme

In our example, we also suppose that a relational database is available, containing the following relations (keys are underlined):

```
PROFESSOR(Name, Area, Photo, e-mail, Phone, RoomNum)
STUDENT(Name, Area, Photo, e-mail, Tutor)
ADMSTAFF(Name, Area, Photo, e-mail, Position)
COURSE(Name, Type, Description, Instructor)
LESSON(CourseName, Day, Hour, RoomNum)
SEMINAR(Title, Author, Date, Hour, Abstract, Responsible)
PUBLICATION(Title, Abstract, Year, Reference)
RESEARCH-GROUP(Name, Topic)
PERSON-IN-GROUP(Name, Group)
PUBLICATION-AUTHORS(Name, Title)
```

4 Phase 3: Hypertext Conceptual Design using NCM

In our perspective, describing the conceptual properties of an hypertext involves three main activities, as follows:

- describing the conceptual *entities* that are part of the hypertext, i.e., the hypertext *nodes*; each entity describes a real world object present in the hypertext, and can be seen as an aggregation of several *attributes*; this is coherent with the fact that, similarly to a database, a hypertext is based on a model of the underlying domain; note that not all ER entities have to be also hypertext nodes; for example, with reference to Figure 2, `PROFESSOR` and

`PUBLICATION` will most probably be hypertext nodes, i.e., relevant classes of objects in the final hypertext, whereas `ROOM` may not;

- describing the hypertextual paths to navigate between entities, i.e., how it is possible to explore the network of nodes; with respect to traditional ER relationships, the description of paths between entities requires to introduce a notion of *direction*, because in the hypertext a navigation always goes from a source to a destination node;

- describing which *access paths* should the hypertext provide starting from a source node in order to locate information of interest; this corresponds to organizing entities in a hierarchy of concepts to be explored by the user.

NCM is a data model to describe a hypertext conceptual scheme.

4.1 NCM: A Conceptual Data Model for Web Hypertexts

NCM is inspired to RMM [22], although depurated of all non-conceptual features. There are two main classes of constructs in NCM, *nodes* and *links*. There are three different kinds of nodes, and two different kinds of links. Among these constructs, *macroentities*, *union nodes* and *directed relationships* aim at describing a hypertextual view of the underlying domain, whereas *aggregation nodes* and *aggregation links* are used to design the hypertext hierarchical access structure (the corresponding graphical symbols are reported in Figure 3).

- **Macroentities**: a macroentity is a node corresponding to a class of objects in the application domain; macroentities have attributes, which can be either simple or complex (a complex attribute is in turno composed of simple or complex attributes, that is, nesting is allowed); each attribute has an associated cardinality, and thus can be either mono or multivalued. For each macroentity (at least) one *descriptive key* must be indicated. A descriptive key of a macroentity is a set of attributes with two properties: *(i)* it is a key (in the usual sense) for instances of the macro-entity and *(ii)* it is explicative about the corresponding instance, i.e., the user should directly infer the meaning of its values. Consider again the Department Example: a descriptive key for a `SEMINAR` macroentity is made of the `Title` and the `Author` of the seminar: although the title alone may suffice to identify different seminar, it does not convey enough meaning about the corresponding seminar, so that also the name of the author is needed to satisfy the second property.[2]

- **Union Nodes**: a union node corresponds to the union of several (possibly heterogeneous) hypertext nodes. In conceptual data models *IS-A* relationships are used for several purposes [5, 21]: besides specifying inheritance constraints, they describe different roles that a type can play, and provide types corresponding to the union of other types. This latter aspect is very important also in a hypertextual framework; in fact, in this context, typing is usually rather loose, and structurally similar navigations often allow to reach objects with different properties. Union nodes in NCM are used to model types corresponding to the union of different macroentities.

- **Aggregation Nodes**: aggregation nodes and aggregation links are the NCM primitives that allow to model the hypertext access structure. An aggregation node describes an aggregation

[2] As for external identifiers in the ER model [11], *external descriptive keys* may be allowed in NCM, i.e., keys comprising not only attributes of the specific macroentity, but also attributes of other macroentities reachable from the given one via directed relationships.

or classification of macroentities; aggregation links go from the node to the corresponding macroentities. Aggregations may either correspond to inheritance hierarchies (see for example, `PEOPLE` as an aggregation of `PROFESSOR` and `STUDENT`), or be explicitly introduced in order to better organize information inside the hypertext. Note that, while directed relationships describe navigations between instances of macroentities, following aggregation links corresponds essentially to selecting different classes of concepts among those present in the hypertext.

- **Directed Relationships**: a directed relationship describes how it is possible to navigate in the hypertext from a source node to a destination node based on their conceptual relationships; cardinality constraints associated with the source node are used to model the number of instances of the destination node associated with one instance of the source node; as a special case, *symmetric* directed relationships are used to indicate that navigation between the two nodes can proceed in both ways;

- **Aggregation Links**: an aggregation link is used to connect an aggregation node to the the corresponding hypertext nodes.

4.2 From ER to NCM

During the hypertext conceptual design phase, the NCM scheme is derived from the ER scheme. The approach we present may be considered "bottom-up" [11], in the sense that hypertext constructs are derived from corresponding constructs in the ER scheme. The main methodological steps are summarized in Table 1. In the following, we discuss each design steps, trying to present in more detail NCM constructs, their relationships with similar ER constructs, and the main design guidelines underlying the translation of the ER scheme into a conceptual hypertext scheme. Examples mainly refer to the NCM scheme of the University Department, shown in Figure 5.

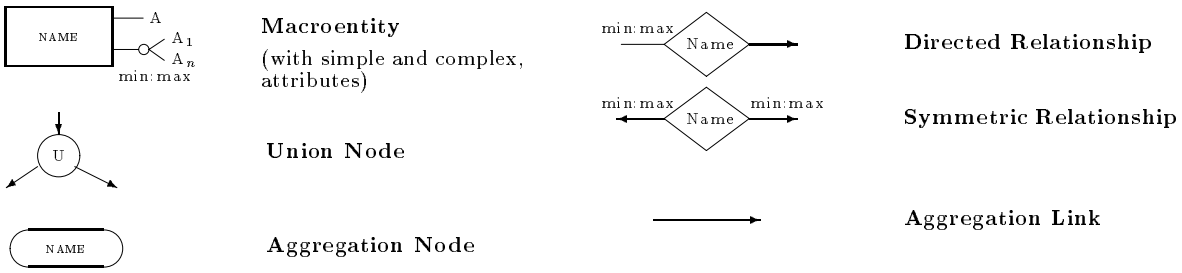


Figure 3: Graphical Representation of NCM Constructs

Step 3.1: Designing Macroentities

Macroentities are intensional descriptions of classes of real world objects to be presented in the hypertext. They indicate the smallest "autonomous" pieces of information which have an independent existence in the hypertext. Examples of macroentities, in our department Web site are `STUDENT` and `COURSE`. Elementary properties of macroentities are described by *attributes*, which can be either *simple* i.e., atomic, or *complex*, i.e., structured; cardinalities may be associated with attributes, so that they can be either mono or multi-valued [11].

Table 1: **Hypertext Conceptual Design Steps:**

Step 3.1: **Designing Macroentities**	macroentities are chosen among entities in the ER scheme; a macroentity may either correspond to a single entity, or to a view over the ER conceptual scheme, i.e, to a portion of the scheme involving several entities and relationships;
Step 3.2: **Designing Union Nodes**	union nodes model types corresponding to the union of different macroentities; they are mainly obtained by mapping ER hierarchies, or may be explicitly introduced to model navigations involving different macroentities;
Step 3.3: **Designing Directed Relationships**	directed relationships model navigations between hypertext nodes, based on conceptual associations; they are a directed counterpart of (undirected) relationships in the ER scheme;
Step 3.4: **Designing Aggregations**	aggregation nodes and aggregation links are used to classify and organize hypertext concepts in order to impose a hierarchical access structure; some aggregations directly come from IS-A hierarchies in the ER scheme, others need to be explicitly introduced.

Macroentities are naturally derived from ER entities or, in some cases, from *views* over entities and relationships. In particular each entity representing objects which are considered relevant and independent in the hypertext, originates a macroentity. In our example, `STUDENT`, `PUBLICATION`, `ADMSTAFF` and `RESEARCH_GROUP` macroentities are derived from the correponding ER entities. For simplicity, we name macroentities with the same name as the corresponding entity. In other cases, macroentities are derived from views over several entities and relationships; as an example, consider the fragment of scheme about courses, lessons, and rooms; in this case, `ROOM` and `LESSON` are not considered as autonomous concepts in the hypertext design. Thus, we choose to incorporate them in suitable macroentities: the first one, called `COURSE` contains cumulative information about courses, lessons and rooms; the second one, called `SEMINAR`, about seminars and rooms; the third one, `PROFESSORS`, about professors and their offices; this essentially corresponds to designing a *view* [11] over the ER scheme for each of these macroentities (see Figure 4). Note that the view definition process may give rise to multivalued attributes, for example to model all lessons associated with an instance of macroentity `COURSE`.

Step 3.2: Designing Union Nodes

Union types naturally arise when translating ER hierarchies. Consider for example the ER fragment describing publications and their authors: the author of a publication may be either a professor or a student. In order to model this fact, a type corresponding to the union of `PROFESSOR` and `STUDENT` is needed. In other cases, they can be explicitly introduced in order to model navigations involving distinct macroentities.

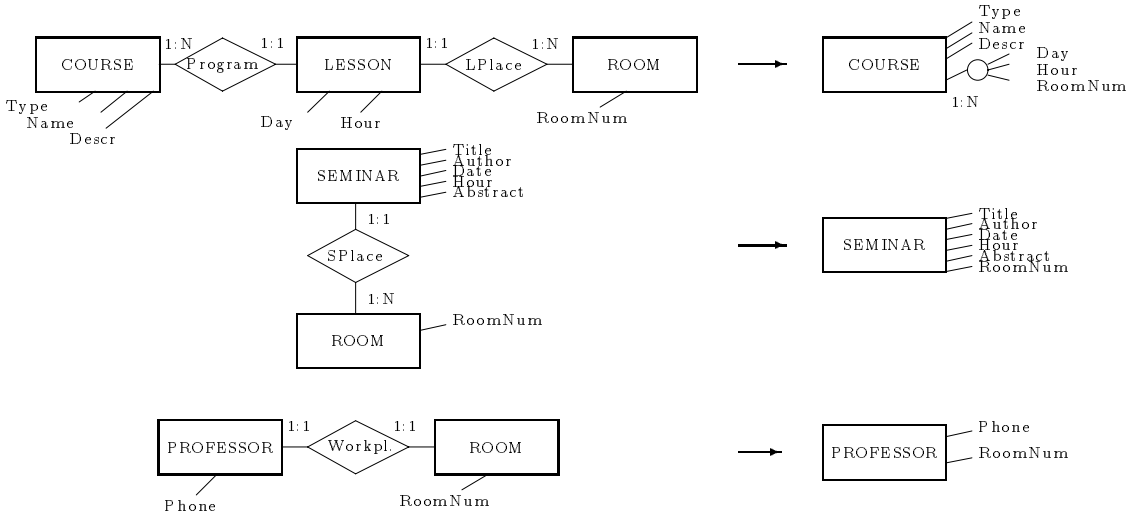


Figure 4: Views over the ER scheme

Step 3.3: Designing Directed Relationships

Directed relationships describe navigations between macroentities, or between a macroentity and a union node, based on conceptual associations. A directed relationship is described by its source and target nodes, and by cardinality constraints. Note that a *symmetric* directed relationship, that is a relationship where the two partecipating macroentities can be both source and target nodes, can be considered as composed by two asymmetric directed relationship.

In general, directed relationships are derived from ER relationships. In our example, a directed relationship from macroentity `SEMINAR` and `PROFESSOR` in the NCM scheme models the fact that, in our hypertextual perception of the application domain, it is possible to navigate from seminars to their respective organizers (and not vice-versa); similary, a symmetric directed relationship between `PROFESSOR` and macroentity `COURSE` is used to describe the fact that it will be possible to navigate the association between the two concepts in both ways (see Figure 5).

Step 3.4: Designing Aggregations

In many cases, aggregations are a natural counterpart of ER hierarchies (see for example, `PEOPLE` as an aggregation of `PROFESSOR` and `STUDENT`); in other cases they may be explicitly introduced in order to better organize information inside the hypertext (for example, professors, seminars and administrative staff are aggregated under a concept of "general Department matters").

Note that, in order to better organize concepts, aggregations may also aggregate other aggregation nodes: in our example, `RESEARCH` is an aggregation node corresponding to `PEOPLE`, `PUBLICATION` and `RESEARCH GROUP`; in turn, `PEOPLE` aggregates professors and students. A root aggregation node is usually present to represent the hypertext entry point (`DEPARTMENT` in our example).

Sometimes, aggregation of a macroentity is only partial, that is, only a subset of instances of the macroentity is to be aggregated; this can be modeled in NCM by attaching *labels* to aggregation links; each label is associated with a predicate on instances of the destination node and is used to specify that only instances satisfying the predicate are considered as part of the aggregation. In our Department example, we may think of aggregating under the name `EDUCATION` macroentities `COURSE` and `PROFESSOR`. However, it is reasonable to distinguish graduate courses from undergraduate ones;

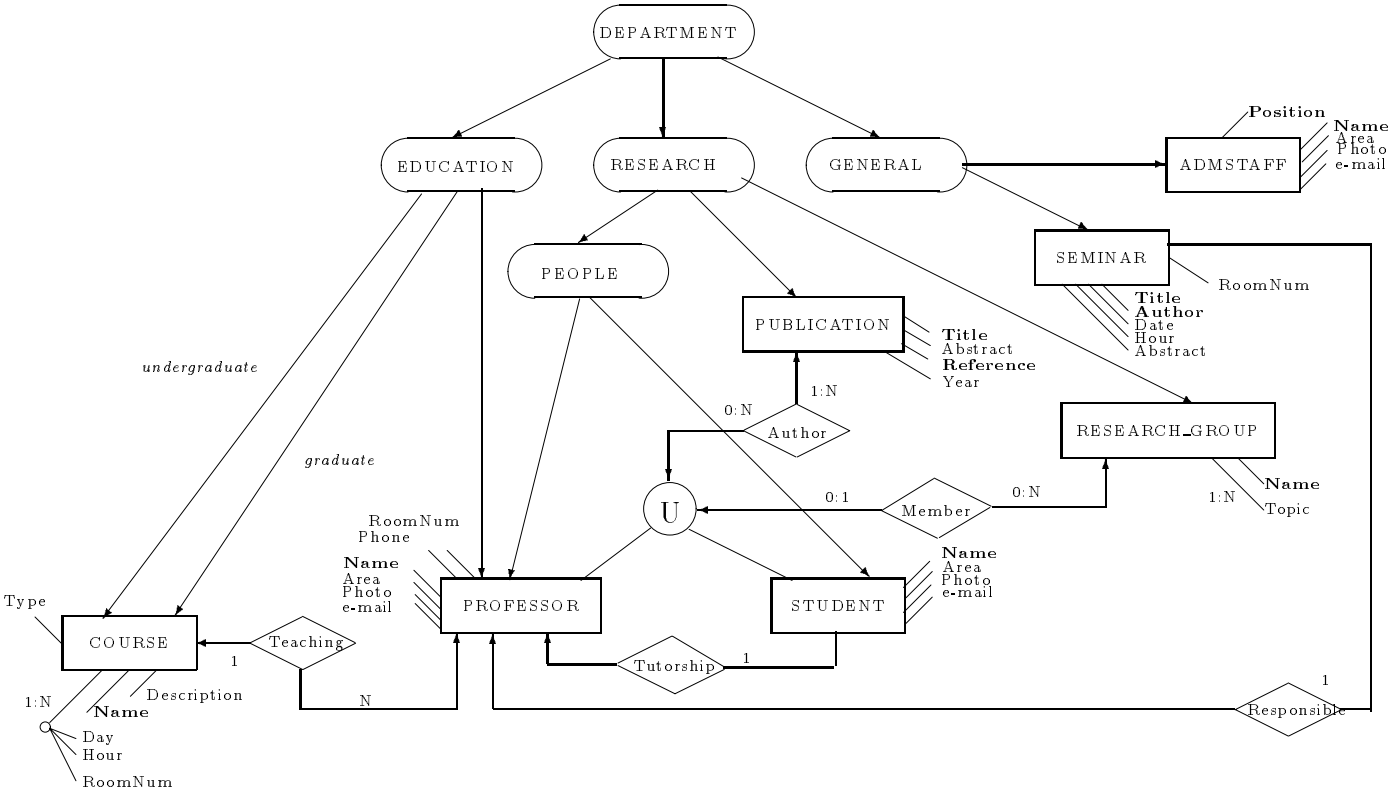


Figure 5: The Department NCM scheme

links with different labels ("undergraduate" and "graduate") are used to this end (see Figure 5).

5 Phase 4: Hypertext Logical Design using ADM

The core of our methodology is the development of the ADM [10] scheme, which represents an abstract description of actual pages in the Web site. In fact, coherently with their conceptual nature, NCM schemes describe how concepts can be navigated in the target hypertext. On the other side, actual Web hypertexts are essentially graphs of pages: these two ways of organizing information can be rather far away from each other, and specific methodological steps are needed in order to derive the second from the first. The hypertext logical design phase is concerned with these aspects. It is based on a logical hypertext data model, called ADM [10], which allows to describe in a tight and concise way the structure of HTML pages by abstracting their logical features. In fact, we say that the model is *page oriented*, in the sense that it recognizes the central role that pages play in this framework.

5.1 ADM Page-schemes

The fundamental feature of ADM is the notion of *page-scheme*, which resembles the notion of relation scheme in relational databases or class in object-oriented databases: a page-scheme is an intensional description of a set of Web pages with common features. An instance of a page-scheme is a Web page, which is considered as an object with an identifier (the URL) and a set of attributes. Moreover, there is one specific aspect in this framework with no counterpart in traditional data

models. On a page-scheme a special constraint can be specified in order to model an important case in this framework: when a page-scheme is "unique", it has just one instance, in the sense that there are no other pages with the same structure. Tipically, at least the home page of each site falls in this category.

Page features are described by means of attributes, which may have simple or complex type. Simple attributes are monovalued and correspond to atomic pieces of information, like *text*, *images* or other multimedia types. *Links* are simple attributes of a special kind, used to model hypertextual links; each link is a pair *(anchor, reference)*, where the *reference* is the URL of the destination page, possibly concatenated to an *offset*, inside the target page-scheme, and *anchor* is a text or an image. Anchors for links may either be constant strings, or correspond to tuples of attributes; to give an example, in the Department Web site, we might want to link each course page with the page of the corresponding professor; to do this, we can add a link attribute to the course page, where the reference is the URL of the corresponding instructor page, and the anchor is made of two text attributes, namely the first and last name of the instructor.

Complex attributes are multivalued attributes and represent (ordered) collections of objects, that is, *lists* of tuples. Component types in lists can be in turn multivalued, and therefore nested lists are allowed. It should be noted that we have chosen lists as the only multivalued type since repeated patterns in Web pages are physically ordered. Note that attributes may be labeled "optional" in order to allow null values.

ADM also provides two more constructs, that we consider particularly relevant in the Web framework. First, ADM, like NCM, also provides a *heterogeneous union* type, in order to provide flexibility in modelling, according to the heterogeneous nature of the Web. Moreover, an important construct in Web pages is represented by *forms*. Forms are used to execute programs on the server and dynamically generate pages. ADM provides a *form type*: in order to abstract the logical features of an HTML form, we see it as a *virtual list of tuples*; each tuple has as many attributes as the fill-in fields of the form, plus a link to the resulting page;[3] such lists are virtual since tuples are not stored in the page but correspond to submissions of the form. Thus, forms and lists are tightly related in ADM; however, the distinction between the two has important implications on the way data can be accessed in the site.

We represent an ADM scheme as a directed multigraph; nodes in the scheme graph are page schemes; the symbol for a unique page-scheme recalls a single page, whereas that for non-unique page-schemes recalls a "stack" of pages; edges are used to denote links. The graphical representation of ADM constructs is shown in Figure 6.

The hypertext logical design phase aims at producing an ADM scheme starting from the NCM hypertext conceptual specifications. The hypertext logical design can be seen as a two-step process: first, based on a sequence of systematic, simple steps, a skeleton of the ADM scheme is built by mapping NCM to ADM contructs; then, this first ADM scheme can be further restructured in order to improve effectiveness and efficiency. The main methodological steps are shown in Table 2. In the following, we first discuss how NCM constructs can be mapped to page-schemes, and then how to restructure the resulting ADM scheme.

5.2 From NCM to ADM

A translation from NCM to ADM essentially consists in mapping NCM nodes to page-schemes and NCM links to ADM link attributes. There are three main methodological steps in this process, as shown in Table 2. In the following, we discuss these steps in more detail.

[3] Forms introduce several specific data types, such as *checkboxes*, *radios* or *selections*. We ignore these aspects here for simplicity, and consider only attributes of type text. All ideas can be easily generalized to the most general case.

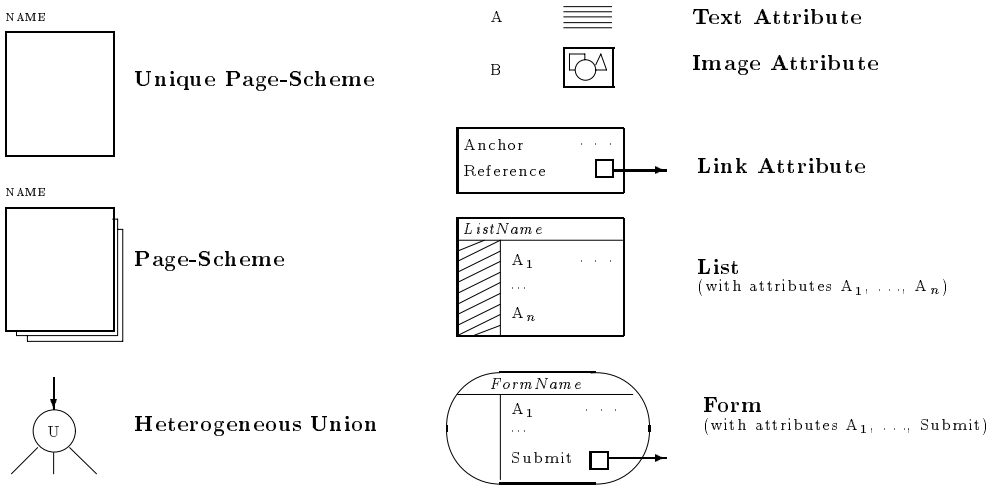


Figure 6: ADM Constructs

Step 4.1: Mapping Macroentities

There are several ways of organizing the presentation of a macroentity instance using Web pages. The most natural way is to use one page for each macroentity instance. As an alternative, several linked pages can be used to present different aspects of the same instance; this is particularly convenient when the corresponding macroentity is particularly rich, and can be seen under different perspectives. At the opposite end, all instances of particular macroentities may in some cases be presented in a single page; although there is no definitive rule, this latter representation seems more appropriate when instances have a particularly concise description. In our example, this may be true for instances of the `SEMINAR` macroentity, for which a single page listing all seminars is a reasonable representation.

Thus, when mapping macroentities to page-schemes, a first design choice consists in choosing which macroentities will be translated on a page-per-instance basis, and for which ones a single page listing all instances is appropriate. In the former case, a page-scheme with the same attributes as the macroentity is generated. In the latter case, the resulting page-scheme will consist of a list of tuples having the same attributes as the starting macroentity. Note that, when translating attributes, NCM single valued attributes are mapped to ADM simple attributes (text, image ecc.) and NCM multivalued attributes are mapped to ADM lists. Figures 7 and 8 show the mapping of macroentities `PROFESSOR` and `SEMINAR` into the corresponding ADM page-schemes: instances of the former will be presented in separate pages, while instances of the latter will be items of a list.

Step 4.2: Mapping Directed Relationships

Directed relationships are mapped to links between page-schemes. As a preliminary step, all NCM symmetric relationships are split in two asymmetric relationships. Then, several design choices are possible, based on the ADM translation of the partecipating macroentities and on the cardinality of the relationship. A new attribute is added to the source page-scheme (i.e., the page-scheme corresponding to the source macroentity): this attribute may be either a link attribute, if the cardinality of the relationship is at most one, or a list of links, if the cardinality is n. The link anchor is in both cases to be chosen among the descriptive keys of the target macroentity; the link reference is based on the URL of the target page-scheme; different choices may be made if instances of the target macroentity have been mapped to pages or items in a list or a page-scheme: in the

Table 2: **Hypertext Logical Design Steps**

Step 4.1: **Mapping Macroentities**	macroentities are mapped to page-schemes; a macroentities is either mapped to a non-unique page-scheme or to a unique page-scheme with a list;
Step 4.2: **Mapping Directed Relationships**	directed relationships among macroentities are mapped to link attributes between the corresponding page-schemes; 1 to 1 relationships map to monovalued link attributes, whereas 1 to N relationships to lists of links; NCM union nodes are naturally mapped using ADM heterogeneous union types;
Step 4.3: **Mapping Aggregations**	aggregation nodes generate more page-schemes, where link attributes are used to model aggregation nodes.
Step 4.4: **Restructuring the Logical Scheme**	the logical scheme generated after the previous steps is analyzed and restructured to improve effectiveness and efficiency.

latter case, an offset inside the page, i.e., a pointer to a portion of the page, may be added in order to reference a specific tuple in the list.

Figures 9 and 10 illustrate this process. In Figure 9, to model the directed relationship from seminars to the corresponding responsible professors, since a single professor is associated with each seminar, a link is added to attribute `SeminarList` in page-scheme `SEMINAR_LIST_PAGE`: it allows the navigation from each seminar (i.e., each item in the list) to the corresponding professor. Note that the link anchor corresponds to the descriptive key of the target macroentity. Then, we consider the 1:n `Author` directed relationship between professors and their publications; the relationship is split in two directed relationships, and each of them is translated separately. In Figure 10, `Author` relationship from professors to publications is mapped to a list of links (`PublList`) in page-scheme `PROFESSOR_PAGE`; each item in the list points to one item in `PUBLICATION_LIST_PAGE`. Finally, the mapping of the two directed relationships between `PROFESSOR` and `COURSE` is shown in Figure 11.

Step 4.3: Mapping Aggregations

Each NCM aggregation node is mapped to an ADM unique page-scheme. NCM aggregation links correspond to link attributes, with some subtleties: if the destination node is in turn an aggregation node or a macroentity mapped to a unique page-scheme, a single link to the corresponding page-scheme is used; the anchor to be used is a constant string, usually corresponding to the target page-scheme name. On the contrary, if the target node is a macroentity mapped to a page-scheme or a union node, an ADM list attribute is added to the source page-scheme; each item in the list is a link to one instance of the destination page-scheme; note that, also in this case, predicates can be used to label ADM links.

An example is given in Figure 12, where aggregation `RESEARCH` is mapped to page-scheme `RESEARCH_PAGE`. Since one of the components nodes is an aggregation node, `PEOPLE`, this is in turn

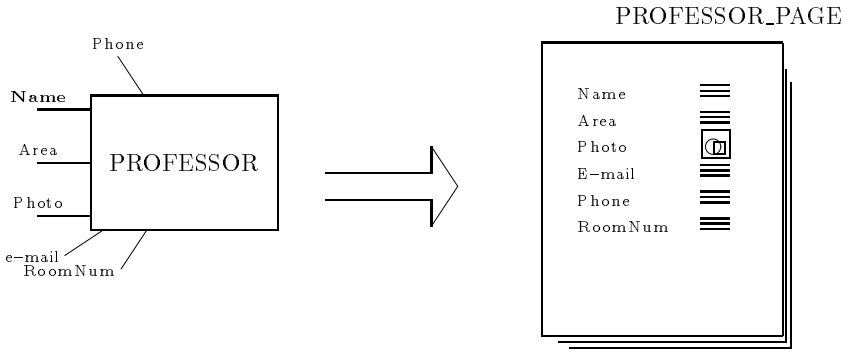


Figure 7: Logical Design Step 1: Mapping Macroentities to page-schemes

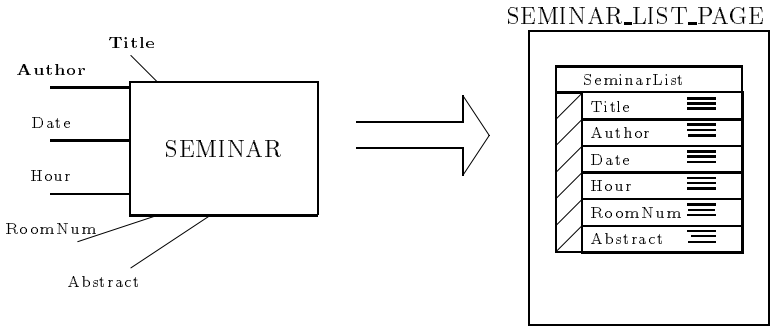


Figure 8: Logical Design Step 1: Mapping Macroentities to tuples

mapped to page-scheme `PEOPLE_PAGE`.

Step 4.4: Restructuring the Logical Scheme

After all NCM primitives have been mapped to ADM constructs, based on the steps described above, a first ADM skeleton results. Althought this logical scheme somehow reflects the conceptual NCM scheme, it may present some limitations. In particular, data may not be sufficiently organized and complex or counterintuitive navigations may be required in order to access information. Moreover, some of the page-schemes can be considered too dense, and thus hardly effective. As an example, when mapping directed relationships to lists of links, lists with a large number of items may arise, thus making the corresponding pages potentially very long and difficult to browse.

In order to solve these problems, the ADM scheme can be restructured, reorganizing information and navigation paths without changing the conceptual hypertext structure. Note that ADM provides a description of the site that is close to the user perception of it. Hence, logical reorganizations of pages are simple and effective. Here, we present several main classes of reorganizations, as follows; it is important to note that, for each of these transformations, also the converse is possible.

Slicing page-schemes Page-schemes corresponding to particularly rich macroentities may be split in two (or more) page-schemes, each presenting a subset of the attributes of the page-scheme; in this way, different perspectives of the same concept can be offered. Note that this activity resembles the *slice design* activity in RMM [22]. Consider for example page-scheme `PROFESSOR_PAGE`; to avoid the generation of large pages, which may be hard to consult for the user, we might think of isolating general information (like name, photo, phone, room number, e-mail address, plus the list of courses) from data concerning research; to do this, the original page-scheme can be split in two, as shown

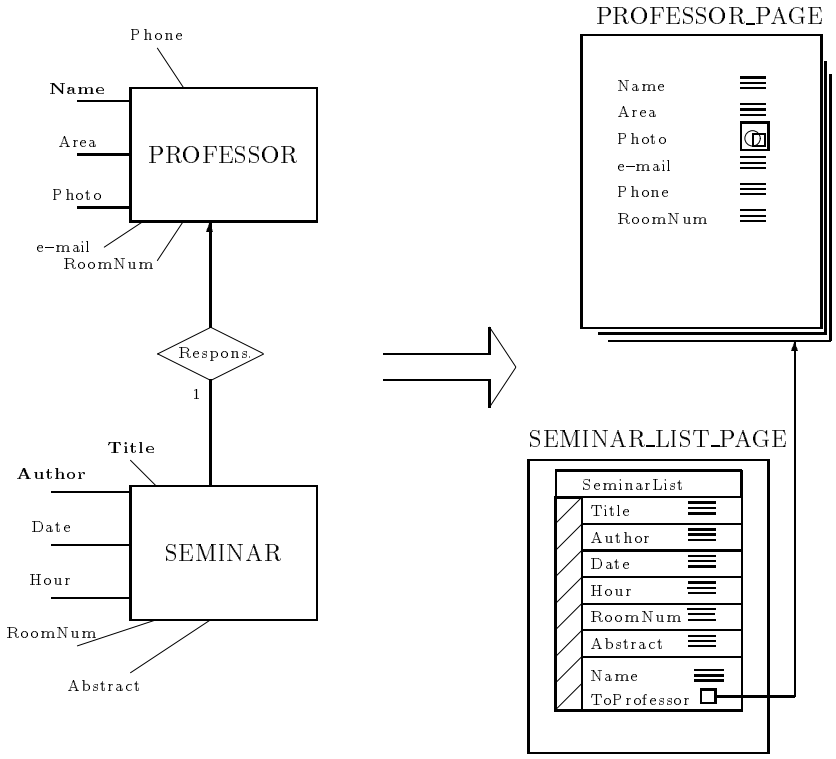


Figure 9: Logical Design Step 2: Mapping Directed Relationships

in Figure 13. Clearly, the two (or more) parts may overlap on some attributes.

Introducing Multi Level Lists and Forms Another common case is the one of lists containing a very large number of items, that make very difficult to data inside a page; in this case, two alternatives are possible: *(i)* the list can be reorganized introducing different levels; or *(ii)* it may be substituted by a form. In the first case, items in the list are grouped based on some criterion, and each group is moved to a different page; the original list will thus contain a fewer numeber of entries, i.e., one for each group of items. With respect to our Department Example, to access a particularly large number of student pages from `PEOPLE_PAGE`, we might think of introducing a multi-level list, dividing students on the basis of their initial. In this way, the original page contains only one item for each initial, pointing to a page containing the list of all students with that initial. As an alternative, very long lists may be substituted using forms; in this way, list items are not to be physically stored in the page, but may be encoded using some program running on the server: when a request is made based on the form, the program is executed and only relevant items are returned. In the Department example, we decide to substitute the list of students in page `PEOPLE_PAGE` with a form such that, by specifying the name of a student, the corresponding page is returned. This transformation is shown in Figure 14.

Other Transformations In other cases, it may be necessary to flatten nested lists. In fact, nested lists may sometimes be considered as difficult to read; in these cases, intermediate pages can be introduced in order to unnest the list: nested levels are substituted by simple links, each pointing to a new page containing all tuples at that level.

In Figure 15 shows the final ADM scheme for the Department example.

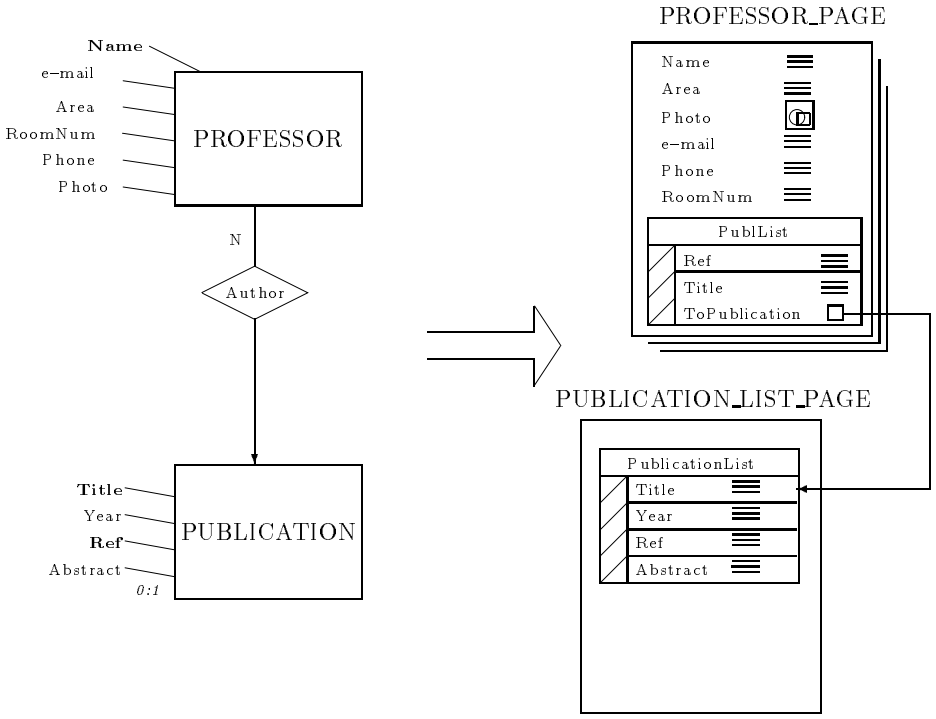


Figure 10: Logical Design Step 2: Mapping Directed Relationships

6 Phase 5: Presentation Design

Once the logical scheme has been developed, a layout should be designed for each page-scheme; this phase, which we call *presentation design*, consists in choosing the right appearance to present pieces of information in a page. Choosing the right page layout is very important in order to make pages attractive to users and to better convey information; however, in the following we mainly concentrate on data-management issues and do not develop further on this activity. We suppose that, as a result of an *presentation design step*, a *page template*, i.e., a single, prototypical page, has been designed for each page-scheme. The template represents at one time a sample of the page layout and a skeleton according to which construct the page. Such a template will be used by penelope to generate instances of the page-scheme based on the corresponding skeleton. It is worth noting that our approach, which clearly distinguishes the page logical features from the presentation and the physical implementation, allows to see the presentation design as an orthogonal activity, whose outcomes can be easily incorporated, using PENELOPE, in the final hypertext.

7 Phase 6: Hypertext to DB Mapping and Page Generation using PENELOPE

As a final step, we need to generate pages starting from the database content. In ARANEUS, this process is supported by a specific language, called PENELOPE [10], which automatically generates HTML source code based on suitable specifications. However, before being able to write PENELOPE statements, we need to establish a mapping between the database and the hypertext; in fact, it is necessary to specify which of the database tables and attributes will contribute to generate pages for each specific page-scheme. As an example, consider pages relative to research groups, described by page-scheme `RESEARCH_GROUP_PAGE` in Figure 15. The page-scheme has the following structure

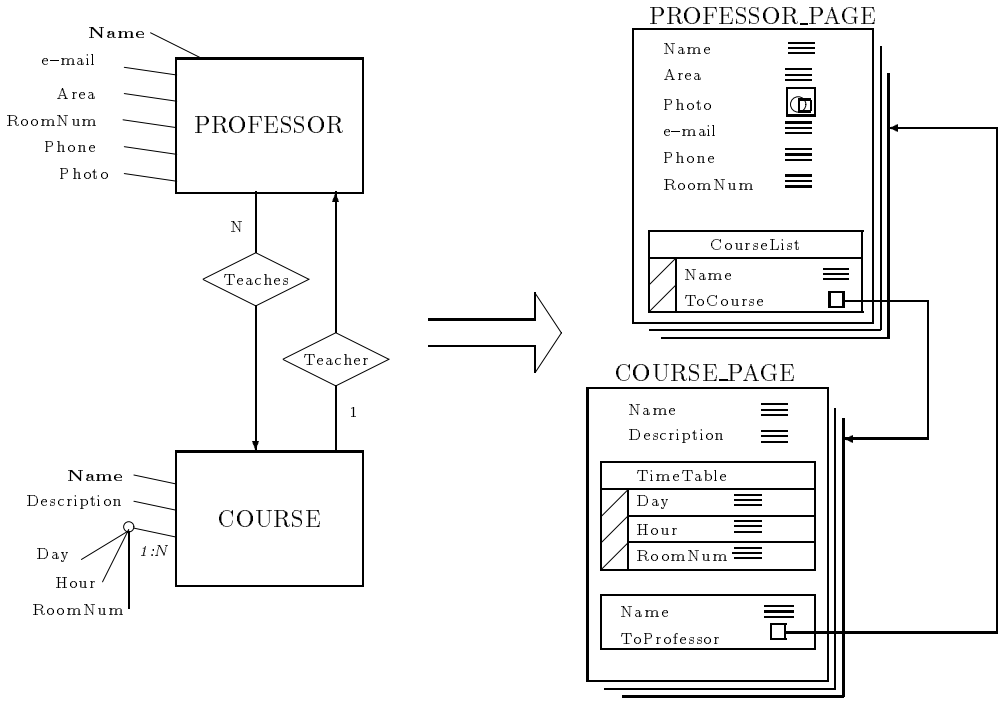


Figure 11: Logical Design Step 2: Mapping Directed Relationships

(described in the ARANEUS Data Definition Language):

```
PAGE-SCHEME RESEARCH_GROUP_PAGE
Name:  TEXT;
       TopicList:  LIST-OF ( Topic :     TEXT;);
       MemberList: LIST-OF ( Member :    TEXT;
                             ToMember :  LINK-TO PROF_GENERAL_PAGE
                                         UNION STUDENT_PAGE;);
```

To generate a page we need to know: *(i)* the research group name; *(ii)* the research topics for that particular group; *(iii)* the names of the group members and the URLs of the respective pages. Thus, the database portion relevant for the page-scheme involves several database tables, namely `RESEARCH-GROUP` and `PERSON-IN-GROUP` (see the database logical scheme in Section 3). In essence, the information in the pages over this page-scheme corresponds to a view over the original database, in which groups and members are joined together, as follows:

```
create view RESEARCH_GROUP_PAGE_VIEW as
select RESEARCH-GROUP.Name as GroupName, Topic, PERSON-IN-GROUP.Name as MemberName
       from     RESEARCH-GROUP, PERSON-IN-GROUP
       where    RESEARCH-GROUP.Name = PERSON-IN-GROUP.Group
```

Once we have designed the view, to be able to fill-out actual pages based on the database content, we need to associate each attribute in the page-scheme with the corresponding view attribute; more important, we need to specify how URLs are to be generated starting from database values, in order to assign a distinct URL to each new page, and to be able to link pages together. In ARANEUS, this is process is based on the PENELOPE language, which allows to specify the mapping between the hypertext and the database, in order to generate actual pages.

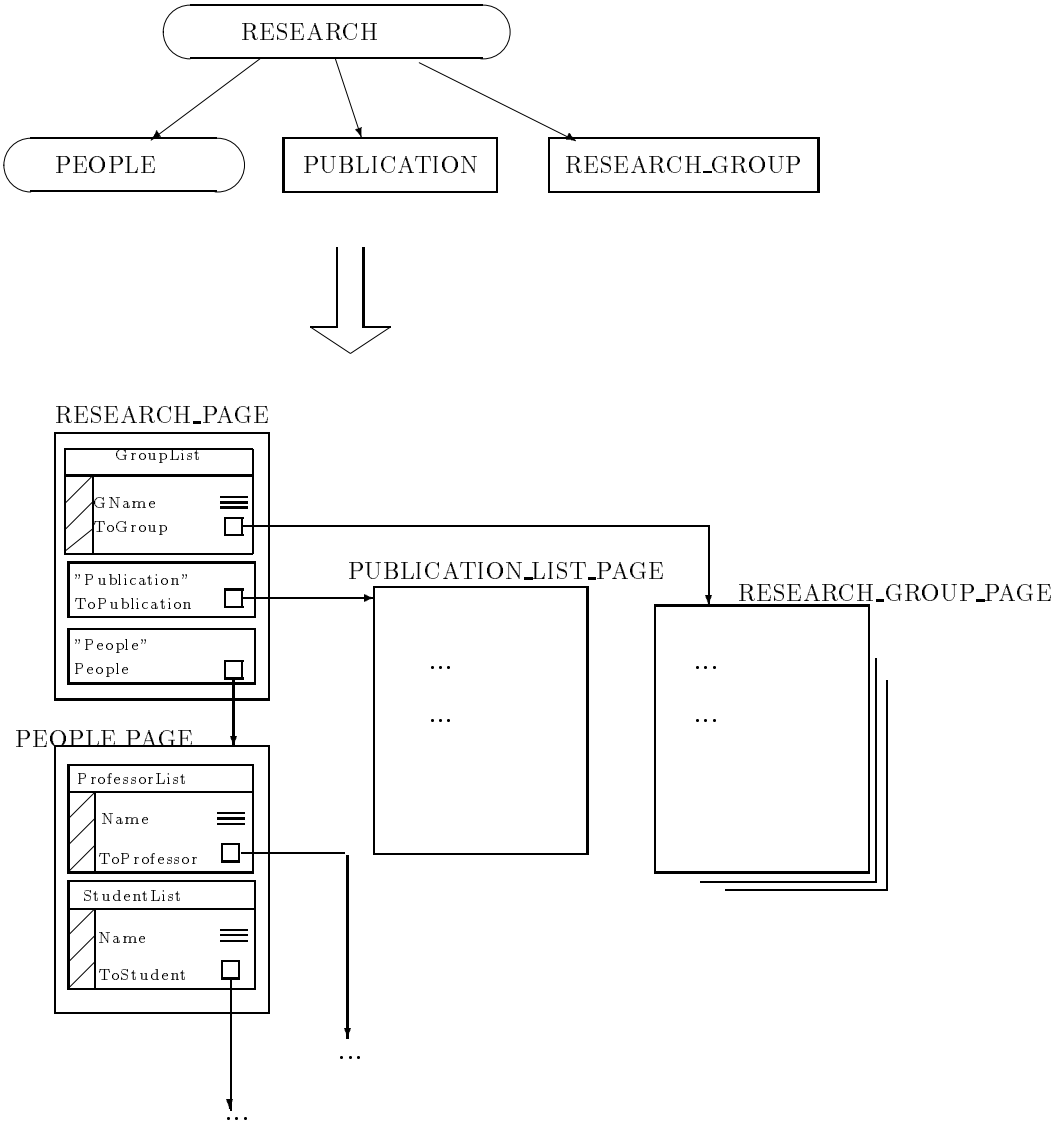


Figure 12: Logical Design Step 3: Mapping Aggregations

7.1 PENELOPE: Syntax and Semantics

PENELOPE is a language for automatically generating complex hypertexts starting from a database. It supports both *push* and *pull* [30] solutions: it can either generate and materialize hypertext pages starting from the database content, or be used to dynamically generate pages upon request. In this paragraph, we first discuss how pages can be materialized and correlated in a complex hypertext, and then show how this technique can be extended in order to generate single pages upon request.

The page organization is described in PENELOPE using `DEFINE PAGE` statements. Each define statement is associated with a specific page-scheme, and essentially specifies how to fill-out pages based on attributes of the corresponding database table (view). The page generation process in PENELOPE has two distinctive features: first, a suitable URL invention mechanism borrowed from object-oriented databases is used to correlate single pages and generate a complex hypertext; moreover, in order to keep track of the structure, PENELOPE automatically adds *meta-information* to the derived pages. Meta-tags are used to mark relevant attributes inside pages, so that their

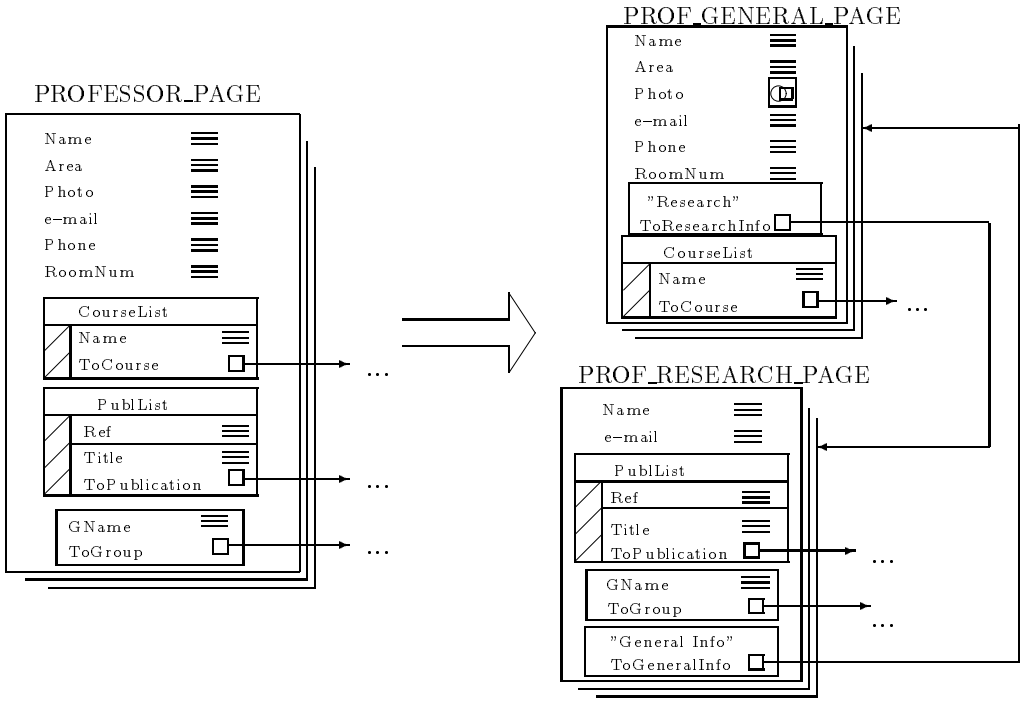


Figure 13: Logical Design Step 4: Slicing Page-Schemes

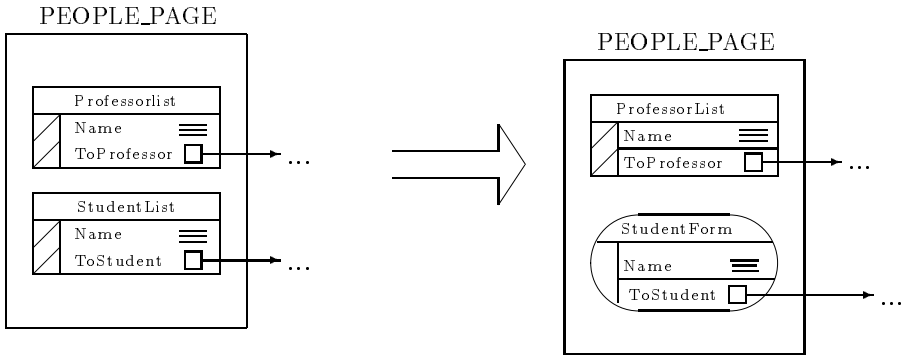


Figure 14: Logical Design Step 4: Introducing Forms

values can be easily extracted from the HTML sources.

The URL invention mechanism is based on the use of *local URLs*. *Local URLs* are used to identify new pages; they can be either constant strings, or strings built using the function symbol `URL` from attributes in relations. For example `result.html` is a constant local URL, whereas `URL(<GroupName>)` denotes a local URL built from values of database attribute `GroupName`. A `DEFINE PAGE` statement has the form:

```
DEFINE PAGE  P [UNIQUE]
AS           S
FROM         R_1, R_2, ..., R_n
IN           DB
```

where: *(i)* P is the page-scheme name; *(ii)* DB is a database; *(iii)* $R_1, R_2, \ldots, R_n$ are tables in DB or SQL views over DB; and *(iv)* S describes the page structure, by specifying the page attributes,

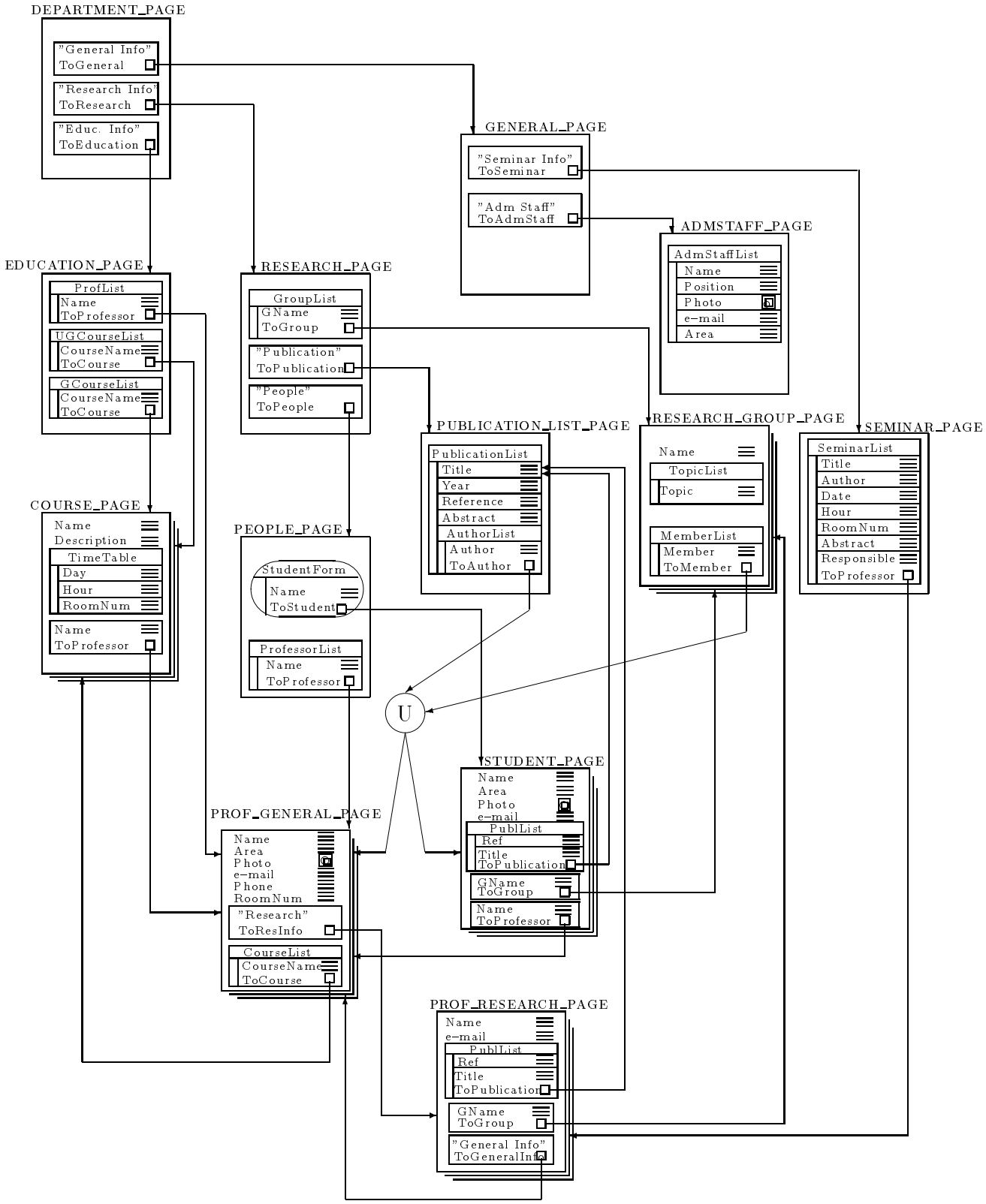


Figure 15: The Department ADM scheme

their type, and their correspondence with database attributes. The presence of the UNIQUE keyword specifies that the defined page-scheme is unique, its absence that it is not unique.[4]

Consider for example page-scheme RESEARCH-GROUP-PAGE in Figure 15. It contains a monovalued attribute, of type TEXT, i.e., the group name, plus two multivalued attributes, corresponding to the list of research topics and group members respectively. The associated database view, called RESEARCH-GROUP-PAGE-VIEW, has attributes GroupName, Topic, MemberName. The following PENELOPE statement is used to generate the corresponding pages:

```
DEFINE PAGE RESEARCH_GROUP_PAGE
AS   URL      URL(<GroupName>);
     GName:       TEXT <GroupName>;
     TopicList:   LIST OF  (Topic:    TEXT    <Topic>;)
     MemberList:  LIST OF  (LINK TO PROFESSOR_PAGE UNION STUDENT_PAGE
                                    (Name:  TEXT    <MemberName>;
                                    ToMemberPage:  REF-TO URL(<MemberName>)));
FROM RESEARCH_GROUP_PAGE_VIEW
IN DEPTDB
```

This statement generates the HTML code for the new pages. It is easy to see that the statement closely resembles the page-scheme structure. Note how the IN clause is used to specify the starting database, which we have called DEPTDB, and the FROM clause indicates which database table(s) or view contains data relevant for the pages. The main part of the statement is the AS clause, describing how to fill-out data in the page. It specifies that a new page with a different URL is to be created for each different research group; clearly, URLs for these pages have to be generated by the system, and each time a page is created, a new, different URL is needed. We use function terms to generate URLs; in fact, term URL(<GroupName>) specifies that the system has to generate an URL for each page, and that the URL must be uniquely associated with a value of attribute GroupName in relation RESEARCH_GROUP_PAGE_VIEW;[5] in this way, a different page will be created for each different research group name.

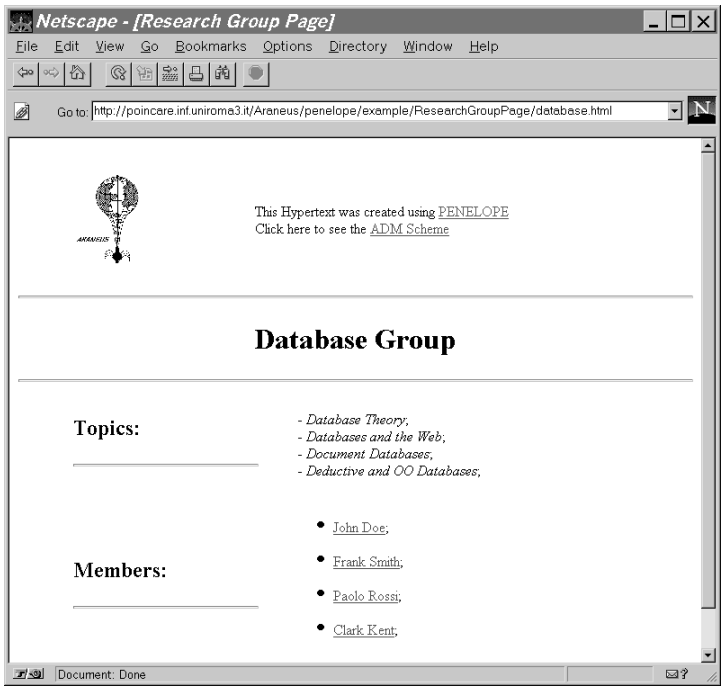
The DEFINE PAGE statement also describes how pages must be filled-out starting from attributes in the relation. For example, the definition of attribute GName of type TEXT specifies that each page will report the research group name, as specified in the ADM scheme, and that the corresponding values come from attribute GroupName of the database view. Note also how local URLs are used to link pages together. In the previous example, we need to specify that, coherently with the page-scheme structure, each page must contain a list of group members, and that, for each member, a link to the corresponding page must be present; these may be either professor or student home pages. In the statement, for each item in list MemberList, we use as an anchor the member name, i.e., a descriptive key of the corresponding page-schemes; then, to link list items to the appropriate pages, we use as a value for the link reference function term URL(<MemberName>); this enforces the correct reference as long as URLs for professor and student pages will be generated using the same function term.

A sample page generated by the previous statement is reported in Figure 16.

So far, we have assumed that pages generated by PENELOPE are materialized and stored in HTML files. However, in some cases, this is not the right solution. In fact, pages containing data to be updated frequently might be out-of-date, and re-generating the pages periodically may require

[4] To avoid inconsistencies, we have to impose some constraints for unique page-schemes; for example, the corresponding local URL must be a constant string; second, the associated page structure must contain a single attribute, which has to be multivalued.

[5] This technique is somehow similar to the use of *Skolem functors* to invent new OID's in object-oriented databases [20].




```
<HTML> <HEAD> <TITLE>Research Group Page</TITLE>
<!--PAGE-SCHEME Research_Group_Page
       GName      : TEXT;
       TopicList  : LIST-OF (Topic : TEXT;);
       MemberList : (Name         : TEXT;
                     ToMemberPage : LINK-TO ProfessorPage UNION StudentPage;)
    END -->
</HEAD><BODY BGCOLOR="FFFFFF">
...
<CENTER><H1><!--GName-->Database Group<!--/GName--></H1></CENTER><HR>
<CENTER><TABLE CELLPADDING=20 COLS=2 WIDTH=90%>
<TR><TD><H2>Topics:</H2><P><HR></TD>
    <TD><!--TopicList-->
        - <I><!--Topic-->Database Theory<!--/Topic--></I>;<BR>
        - <I><!--Topic-->Databases and the Web<!--/Topic--></I>;<BR>
        ...
        <!--/TopicList--></TD></TR>
<TR><TD><H2>Members:</H2><P><HR></TD>
    <TD><UL><!--MemberList-->
          <LI><!--ToMemberPage--><A HREF="/ProfessorPage/johndoe.html"><!--/ToMemberPage-->
              <!--Name-->John Doe<!--/Name--></A></LI>;<P>
          <LI><!--ToMemberPage--><A HREF="/ProfessorPage/franksmith.html"><!--/ToMemberPage-->
              <!--Name-->Frank Smith<!--/Name--></A></LI>;<P>
          ...
        <!--/MemberList--></UL></TR>
</TABLE>
...
```

Figure 16: A sample page with the associated HTML source

some time. Consider for example the seminar list page: seminars presumably need to be updated regularly, to add new ones and delete old announcements. Thus, it would be desirable, instead of storing the seminar list in a HTML file, to generate it upon request starting from the database content. The URL-specification mechanism of PENELOPE can be easily extended in order to achieve this goal. The idea, here, is that, whenever linking the seminar list page, instead of referencing a file name, a URL corresponding to a suitable PENELOPE *query* can be used; this particular query is to be executed using the CGI interface whenever the page is requested and is used to access the database and generate a *single page*[6], instead of the whole site. In this way, the seminar list page may be kept *virtual*, in order to avoid inconsistencies with respect to the database.

Thus, the highest flexibility in the page generation process corresponds to using both materialized and virtual queries, the first for more stable pages, to be updated regularly, and the latter for pages with a higher dynamics.

7.2 Marking the Structure with Meta-Information

A Web site is usually considered as a loosely structured repository. However, in our case, the ADM scheme provides a concise and structured description of the site content, in the database spirit. Thus, we would like to keep track of such structure when generating HTML files, and make it somehow accessible to users. The way PENELOPE does this is by adding *meta-tags* to HTML pages, in order to mark the structure. These meta-tags are embedded in HTML comments, and thus are completely transparent to ordinary Web browsers; however, they can be used by more sophisticated tools in order to extract relevant pieces of information from the page.

Consider for example the HTML source shown in Figure 16. Some hidden tags have been added, beside actual data to be displayed. The first of these tags, in the page header, `<!-- PAGE-SCHEME Research_Group_Page ... -->`, describes the structure of the page-scheme according to which the page is organized. Then, for each attribute in the page, the corresponding value is marked by suitable meta-tags in order to easily recognize and extract the value.

Once the page has been organized in this way, several tools can be used to *query* the site, i.e., to automatically navigate the site to extract information based on high-level queries; this is often desirable when accessing large amounts of data, in order to avoid the usual disorientation associated with browsing; ULIXES [10, 9], is a tool we have explicitly designed to this end; queries over a site can be expressed based on the corresponding ADM scheme using simple path-expressions, in order to access pages and store data in a local database; as an alternative, straightforward extensions of *W3QS* [23] or *WebSQL* [26] could be used, or even a combination of a HTTP robot and grammar parser. In this way, the resulting site is not only a bunch of HTML files, but a highly structured repository that can be either browsed or queried, thus making data access more effective.

Acknowledgments

The authors wish to thank Alessandro Masci and Salvatore Labonia who contributed to the development of PENELOPE and provided useful comments. Thanks also go to Riccardo Torlone, who commented on an early draft of the methodology, helping us to improve the overall organization of the work.

[6] This process is straightforward for unique page-schemes; it is slightly more complex when a specific instance of a non-unique page-scheme is to be generated; in fact, in this case, appropriate selections have to be made on the database in order to extract only data relevant for the page: this can be done by specifying in the `FROM` clause of the `DEFINE PAGE` statement a suitable SQL query based on appropriate attribute values.

References

[1] AltaVista Technology Home Page. http://www.altavista.com.

[2] The ARANEUS Project Home Page. http://poincare.inf.uniroma3.it:8080/Araneus.

[3] Informix Home Page. http://www.informix.com.

[4] Oracle Home Page. http://www.oracle.com.

[5] S. Abiteboul and R. Hull. IFO: A formal semantics database model. *ACM Transactions on Database Systems*, 12(4):297–314, December 1987.

[6] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. Wiener. The Lorel query language for semistructured data. *Journal of Digital Libraries*, 1(1):68–88, April 1997.

[7] P. Atzeni, A. Masci, G. Mecca, P. Merialdo, and E. Tabet. ULIXES: Building relational views over the web. In *Thirteenth IEEE International Conference on Data Engineering (ICDE'97), Birmingham, UK*, 1997. http://poincare.inf.uniroma3.it:8080/Araneus/publications.html.

[8] P. Atzeni and G. Mecca. Cut and Paste. In *Sixteenth ACM SIGMOD Intern. Symposium on Principles of Database Systems (PODS'97), Tucson, Arizona*, 1997. http://poincare.inf.-uniroma3.it:8080/Araneus/publications.html.

[9] P. Atzeni, G. Mecca, and P. Merialdo. Semistructured and structured data on the Web: Going back and forth. In *Workshop on the Management of Semistructured Data (in conjunction with ACM SIGMOD).* http://www.research.att.com/~suciu/workshop-announcement.html, 1997. http://poincare.inf.uniroma3.it:8080/Araneus/publications.html.

[10] P. Atzeni, G. Mecca, and P. Merialdo. To Weave the Web. In *International Conf. on Very Large Data Bases (VLDB'97), Athens, Greece, August 26-29*, 1997. To Appear. http://-poincare.inf.uniroma3.it:8080/Araneus/publications.html.

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

[12] P. Buneman, S. Davidson, G. Hillebrand, and D. Suciu. A query language and optimization techniques for unstructured data. In *ACM SIGMOD International Conf. on Management of Data (SIGMOD'96), Montreal, Canada*, pages 505–516, 1996.

[13] S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. D. Ullman, and J. Widom. The TSIMMIS project: Integration of heterogenous information sources. In *IPSJ Conference, Tokyo*, 1994.

[14] R. A. ElMasri and S. B. Navathe. *Fundamentals of Database Systems.* Benjamin and Cummings Publ. Co., Menlo Park, California, second edition, 1994.

[15] M. Fernandez, D. Florescu, J. Kang, A. Levy, and D. Suciu. STRUDEL – a Web site management system. In *ACM SIGMOD International Conf. on Management of Data (SIGMOD'97), Tucson, Arizona*, 1997. Exhibits Program.

[16] M. Fernandez, D. Florescu, A. Levy, and D. Suciu. A query language and processor for a Web site management system. In *Workshop on the Management of Semistructured Data (in conjunction with ACM SIGMOD).* http://www.research.att.com/~suciu/-workshop-announcement.html, 1997.

[17] F. Garzotto, L. Mainetti, and P. Paolini. Hypermedia design, analysis and evaluation issues. *Communications of the ACM*, 58(8):74–86, August 1995.

[18] F. Garzotto, P. Paolini, and D. Schwabe. HDM – a model-based approach to hypertext application design. *ACM Transactions on Information Systems*, 11(1):1–26, January 1993.

[19] J. Hammer, H. Garcia-Molina, J. Cho, R. Aranha, and A. Crespo. Extracting semistructured information from the Web. In *Workshop on the Management of Semistructured Data (in conjunction with ACM SIGMOD).* `http://www.research.att.com/~suciu/-workshop-announcement.html`, 1997.

[20] R. Hull and M. Yoshikawa. ILOG: Declarative creation and manipulation of object identifiers. In *Sixteenth International Conference on Very Large Data Bases, Brisbane (VLDB'90)*, pages 455–468, 1990.

[21] R.B. Hull and R. King. Semantic database modelling: Survey, applications and research issues. *ACM Computing Surveys*, 19(3):201–260, September 1987.

[22] T. Isakowitz, E. A. Stohr, and P. Balasubramanian. RMM: A methodology for structured hypermedia design. *Communications of the ACM*, 58(8):34–44, August 1995. `http://www.-stern.nyu.edu/~tisakowi/rmm/`.

[23] D. Konopnicki and O. Shmueli. W3QS: A query system for the world-wide web. In *International Conf. on Very Large Data Bases (VLDB'95), Zurich*, pages 54–65, 1995.

[24] L. Lakshmanan, F. Sadri, and I. N. Subramanian. A declarative language for querying and restructuring the Web. In *6th Intern. Workshop on Research Issues in Data Engineering: Interoperability of Nontraditional Database Systems (RIDE-NDS'96)*, 1996.

[25] A. Y. Levy, A. Rajaraman, and J. J. Ordille. Querying heterogeneous information sources using source descriptions. In *International Conf. on Very Large Data Bases (VLDB'96), Mumbai(Bombay)*, 1996.

[26] A. Mendelzon, G. Mihaila, and T. Milo. Querying the World Wide Web. *Journal of Digital Libraries*, 1(1):54–67, April 1997.

[27] G. Poncia and B. Pernici. A methodology for the design of distributed Web systems. In *Conference on Advanced Information Systems Engineering (CAiSE'97)*, 1997.

[28] D. Schwabe and G. Rossi. The Object-Oriented Hypermedia Design Model. *Communications of the ACM*, 58(8):45–46, August 1995.

[29] K. Takahashi and E. Liang. Analysis and design of Web-based information systems. In *Sixth International World Wide Web Conference (WWW'97)*, 97. `http://www6conf.slac.-stanford.edu/`.

[30] M. Winslett. Databases and the World Wide Web, 1997. Tutorial presented at ICDE'97.