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

From Databases to Web-Bases: The Araneus Experience

G. Mecca[1], P. Atzeni[2], P. Merialdo[1,2], A. Masci[2], G. Sindoni[2]

RT-DIA-34-1998 **May 1998**

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

[2] Dipartimento di Informatica e Automazione
Università di Roma Tre
00146 Roma – Italy

{mecca,atzeni,merialdo,masci,sindoni}@dia.uniroma3.it

This work was partially supported by Università di Roma Tre, MURST, and Consiglio Nazionale delle Ricerche.

ABSTRACT

The ARANEUS project aims at developing tools for data-management on the World Wide Web. Web-based information systems deal with data of heterogeneous nature, mainly database data and HTML documents. We have implemented a system, called a Web-base Management system, for managing such repositories. The system is designed to support several classes of applications: (i) high-level access to data in the Web; (ii) design, implementation and maintenance of Web sites; (iii) cooperative applications on the Web. We discuss the lessons learned from our experiences with the system, ranging from database-style query interfaces to popular Web sites, to the design and implementation of several sites, among which an integrated Web museum, which correlates data coming from several virtual museums on the Web.

1 Introduction

The enormous growth of the World Wide Web suggests that it will soon become not only a uniform interface for sharing data, but also a standard, world-wide distributed computing platform. Next-generation information systems are likely to be based on HTTP-like protocols, hyper-textual front-ends and platform-independent programming languages. This will clearly have a strong impact on the role played by *data* in such systems. Traditional concepts, such as *data-independence* from applications, design methodologies, and even the concept of DBMS need to be reconsidered in this new framework. In our perspective, database management systems will evolve into new and more sophisticated forms of repositories capable of dealing with these new requirements in data manipulation.

We call a *Web-Base* a collection of data of heterogeneous nature, and more specifically: (*i*) highly structured data, such as the ones typically stored in relational or object-oriented database systems; (*ii*) semistructured data, in the Web style. We can simplify by saying that it incorporates both databases and Web sites. We call a *Web-Base Management System (WBMS)* a system for managing such Web-bases, i.e, a system providing functionalities for both database and Web site management. It is natural to think of it as an evolution of ordinary DBMSs, in the sense that it will play in future generation Web-based Information Systems the same role as the one played by database systems today. Coherently with the nature of the Web, the system should be fully distributed: databases and Web sites may be either local or remote resources.

We have developed a system, the ARANEUS *Web-Base Management System* (a demo of which will be given at SIGMOD'98 [47]), which meets the above requirements. Original features of the system are: (*i*) a data model called ADM for Web documents and hypertext; (*ii*) several languages for wrapping, querying, creating and updating Web sites; (*iii*) methods and techniques for Web site design and implementation.

The goal of this paper is the discussion of the various experiences we made in developing the ARANEUS WBMS and using it as a support for Web-based applications of various kinds. The focus of the paper is not on the technical novelties of the system, which have been described elsewhere [22, 21, 23], but on an *a posteriori* evaluation of choices, on the lessons learned, and on the evolution of the system due to our experiments with it.

In Section 2, we introduce the architecture of the system and briefly summarize its main modules. Then, in the following Sections, we concentrate on the discussion of our experiences with the system. We do not formally present the data model and the various languages, but rather try to give an intuition of their functionalities by means of examples.

Our experiences refer to three main classes of applications that a WBMS should support: (1) *queries*: the system should allow to access data in a declarative, high-level fashion; this means that not only structured data, but also Web sites can be accessed and queried; (2) *Web site design, implementation and management*: in fact, the process of designing, implementing and maintaining new Web sites is a critical one, that can greatly benefit from the adoption of database techniques; (3) *integration of Web sites*: the two activities above can be coordinated in order to establish Web-based cooperative applications, i.e., applications that extract data coming from existing sites, correlate them, and present them in integrated form in new sites to be browsed by final users.

In Section 3, we illustrate the main problems concerned with wrapping and querying Web sites, discussing our experiments with the data model, various approaches to wrapping a data-source, and several alternative paradigms we have tested for expressing queries on a Web site.

In Section 4, we concentrate on the process of Web-site design and implementation. Here, the focus is on the benefits of adopting a specific design methodology, in the spirit of information systems [24], for Web site design. Also, we compare push and pull approaches to page generation, and some fundamental technical issues related to maintaining a site.

Finally, experiences at points above are somehow summarized in Section 5 by a site-integration application, the *Integrated Web Museum* [8], developed using data coming from the Uffizi [14], Louvre [10] and Capodimonte [4] Web sites. Here, we thoroughly discuss how the integration process may deal with heterogeneities between the original sites, and compare materialized vs. virtual approaches in the context of view maintenance.

Related work is discussed in Section 6. Conclusions are reported in Section 7.

2 The Araneus Web-Base Management System

The Araneus WBMS introduces a number of new tools and techniques for managing Web-bases. The overall architecture is shown in Figure 1. The system is implemented in Java and runs on any Java-enabled platform.

The **User Interface** is completely written in HTML, so that end-users and administrators can access the system from any client on the network. Figure 2 shows the main menu of the system. It can be seen from the figure that the system allows to manipulate data coming from different sites. Sites are divided into *external sites*, i.e., sites administered by third parties, over which the WBMS has no direct control, and *local* or *internal sites*, i.e., sites created and administered by the system, over which it has complete control. Operations allowed on external sites are essentially queries and local warehousing of site data. Furthermore, in addition to queries, also updates and restructurings can be done on local sites.

For each site, the system manipulates data of heterogeneous nature, and essentially HTML pages and database tables. With respect to external sites, database tables contain data extracted from the site and materialized locally to serve as a basis for further computations. In the case of local sites, tables are used to manage data to be published on the site. Due to this heterogeneous nature of Web-bases, several data models and languages are used. In the following, we briefly discuss the key features and the corresponding components of the system.

2.1 Data Models

In our perspective, structured data are essentially database tables; hence, the data model used to describe structured data is the relational model, and the corresponding language is SQL. Note that other models for structured data, for example object-based, were possible. However, the adoption of a relational model allows us to leverage standard, wide-spread and robust technology and to accomplish a real platform independence. In fact, we assume that the system has access to a (possibly remote) DBMS—as shown in Figure 1—and uses it to store and manipulate tables.

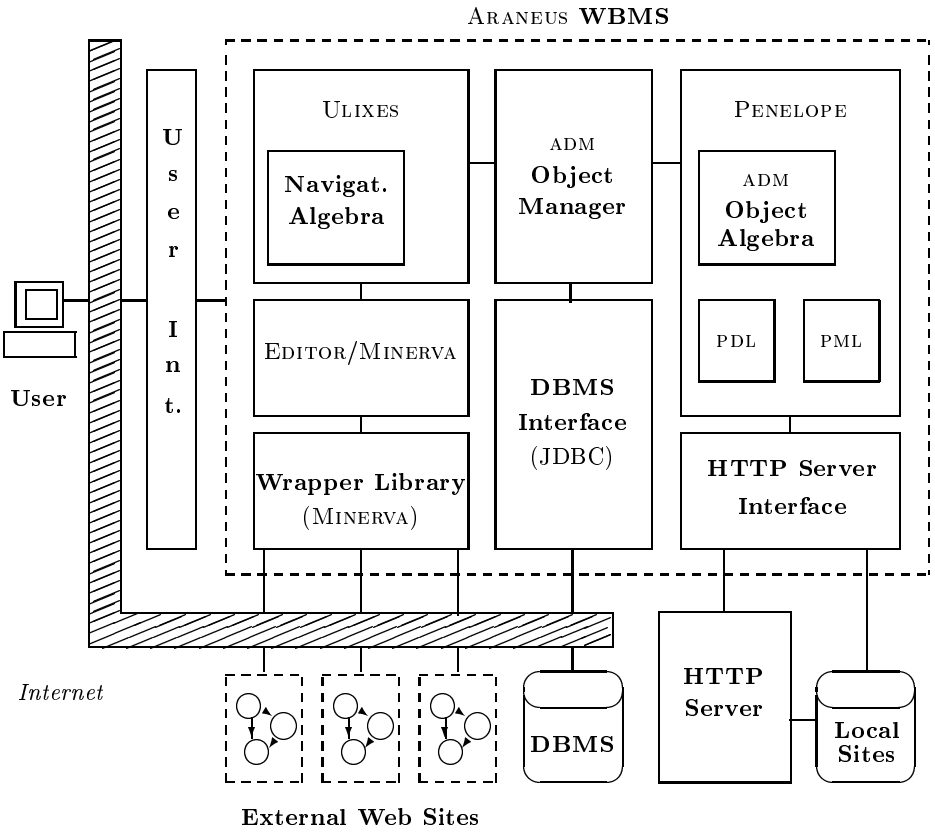

Figure 1: Architecture of the Araneus WBMS

Strictly speaking, the DBMS is not part of the WBMS itself; it is more appropriate to say that the WBMS relies on a DBMS to handle tables. Any table-based DBMS—relational or object-oriented—fits in the system; the standard SQL-based protocol JDBC [9] is used to communicate with the database.

A new model, called ADM [22] has been developed for handling Web documents. ADM represents a key component of the system. It is an ODMG-like [29] model for describing Web-hypertexts. Each page is seen as a URL-identified nested object. Similar pages are grouped into *page-schemes*, which recall the notion of relation scheme or class in databases. Inheritance is not provided, in favor of *heterogeneous union*, which in our experience better models semistructured hypertexts. Also, other types such as *forms* are introduced to model Web-specific features. A *Web site* is a collection of page-schemes connected by links. Figures 3 and 7 show a graphical representation of some ADM schemes, with an explanation of the graphical primitives. Further examples of ADM schemes can be found on our Web site [2].

In the system, ADM objects are handled by the ADM **Object Manager** module (see Figure 1); it manipulates nested objects by decomposing them and using the DBMS to store them in flat tables.

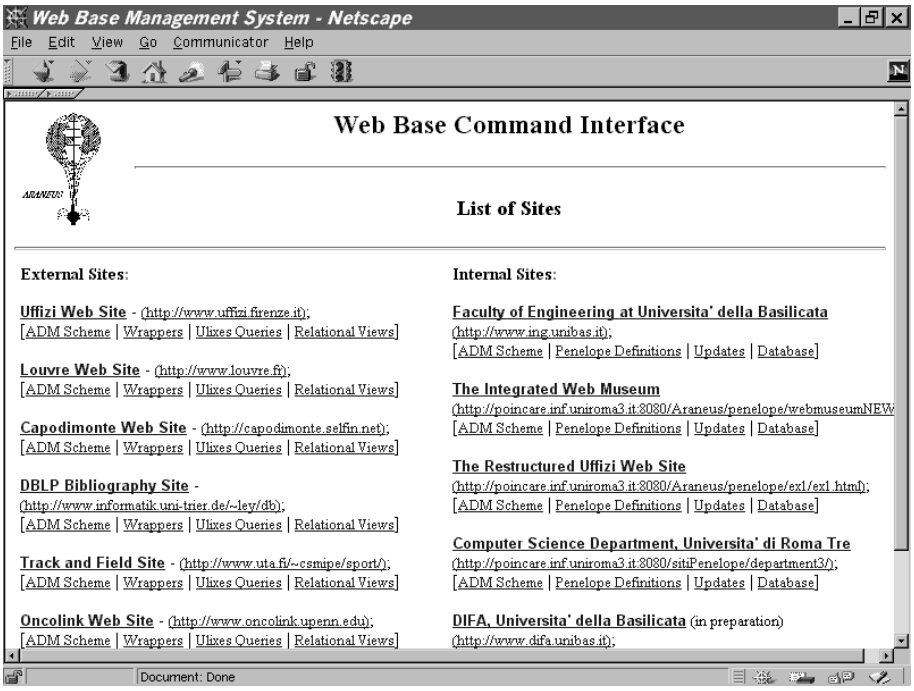

Figure 2: WBMS Menu Page

2.2 Queries over Hypertexts

The system allows to query Web-sites using a suitable language called ULIXES [22], which essentially implements a **Navigational Algebra** [48]. External sites need to be wrapped in order to extract data from pages and see them as instances of page-schemes. Wrappers are written using a text-management language, called EDITOR [21]. The system uses an extensible **Wrapper Library** to store wrappers; whenever a page of a certain page-scheme is to be accessed, its source is downloaded from the network and then processed by the corresponding wrapper, which extracts attribute values and stores the resulting ADM object using the **Object Manager**.

2.3 Hypertext Creation and Management

New sites can be created and administered using the system. Here, the idea is that data to be published on the site are stored in the database. Pages in the site can be either materialized or kept virtual. The module that supports the process of defining and maintaining new sites is called PENELOPE [22]. It incorporates the ADM **Object Algebra**, a nested relational algebra with URL invention that can be used to generate ADM views, and therefore HTML pages, over database tables. These views are declaratively defined using the PENELOPE *Definition Language* (PDL), and maintained using the PENELOPE *Manipulation Language* (PML). When a new site is to be generated, its ADM scheme is first designed. This process is supported by a specific design methodology [23]. Then, the ADM structure is mapped to the database using PDL, which provides a `Define-Page` command to describe the structure of a page-scheme in terms of database tables; page-schemes can be arbitrarily nested and linked using the ADM **Object Algebra**. Then, the

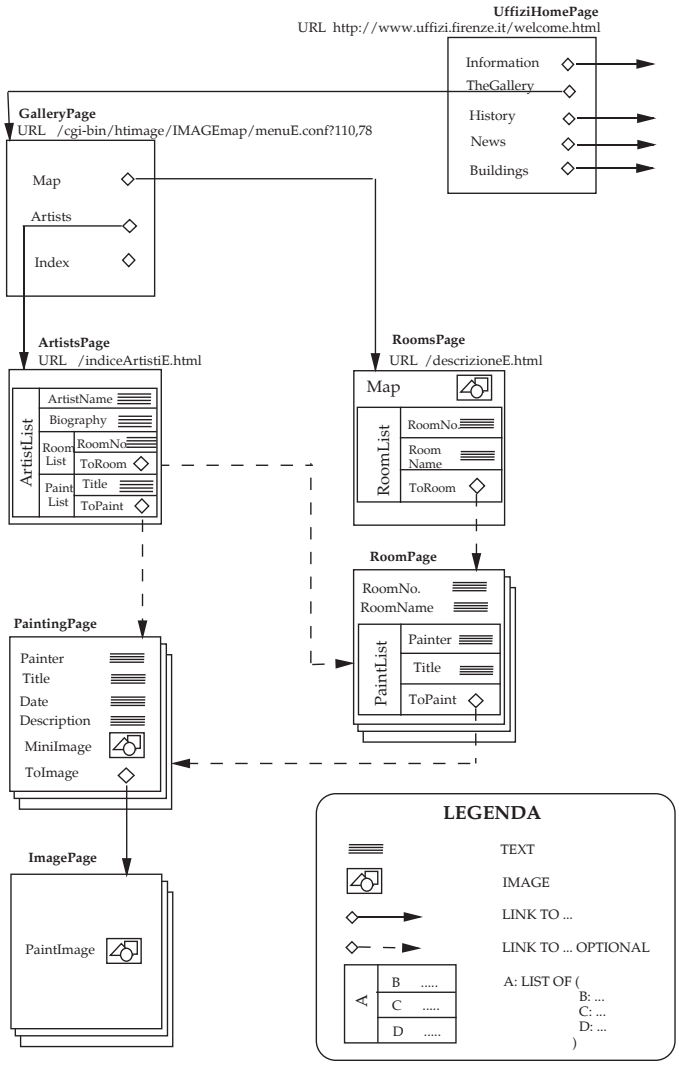

Figure 3: ADM Scheme for the Uffizi Web site

actual site is generated using PML. The new pages may include links to existing pages, and so the new site can be linked to existing ones.

3 Querying the Web

The first experiences with our system were aimed at investigating the chances of re-applying traditional database concepts, such as the ones of *data-model*, *scheme* and *query* to the Web. In the attempt to find a reasonable database-like abstraction from Web data, we had soon to face the need of choosing the right perspective under which we wanted to consider the Web. It is in fact apparent that the richness of the Web leaves great freedom in choosing the level of abstraction for looking at data. It has also been recently argued [42] that a full-fledged data-model for the Web should encompass descriptions of data at different levels, as follows;

- early attempts [41, 49] at defining query techniques for the Web essentially considered the *whole Web as a huge database*; at this level, the graph abstraction seems to be the only reasonable one, since heterogeneities are too big to abstract any common features beside the link topology;

- an alternative is to consider *the Web as a collection of sites, each site being a separate database*; this view of the Web is justified by the fact that Web sites are often quite specific in their offer of data and services, and therefore their description can benefit from more accuracy in the semantics of data and their relationships;

- another extreme may concentrate on *single documents, i.e. HTML pages, each seen as a separate data source*; in some cases, this is also reasonable, since there are pages containing complete databases (for example, stock prices or flight schedules) that might be worth querying by themselves. This approach is substantiated by a large body of research in the field of document query languages [46, 37, 17].

It can be seen that each of these levels represents somehow a compromise between richness of information and accuracy of the description: seeing the Web as a database allows in fact to capture its treasure of information, but at the same time it forces to shift towards a very high-level description, which captures little of the semantics of data. On the contrary, the description of a single page may be very accurate, but shows little of its connections with the rest of the Web.

When we first started our study, we concentrated on the "Web site as a database" perspective, which in our opinion represents a good compromise, and investigated how much of the database inheritance could be re-used in this context.[1] As a first consequence of this, our study completely ignored the problem of *finding relevant data* in the Web, on which other approaches had concentrated so far. We considered the case in which a user has already identified a bunch of sites containing information of his/her interest, and wants to be able to flexibly access and manipulate data in the site, perhaps to be used as a basis for further computations.

3.1 The Data-Model

One of the key features of our approach was the attempt to re-consider in the Web world the cardinal notion of database scheme as an intensive, compact description of a larger extension of data. Now, the Web contains a plethora of sites, of very different nature. Clearly, only sites whose informative content is of a certain size (in the order to hundreds of pages or above) are worth applying database-style techniques (small sites can be well explored using a browser). Hence, *large Web sites* became the target of our analysis, as a preferred counterpart of databases on the Web. Interestingly, we soon realized that, although the Web as a whole may be well considered a very unstructured data-source, from the logical viewpoint large Web sites are often very structured, and therefore well suited to be described using database techniques. There is a number of reasons behind this; the main one is that it is very hard to maintain sites with a poor organization; also,

[1] Although our data model was primarily conceived as a tool to describe Web sites, it has been shown in [22] that is generalizes other data models proposed for the Web, and therefore can be used to give a description of the whole Web and as a basis to ask Web-wide queries.

users find very difficult to localize data of interest inside these sites, and tend to prefer ones with a cleaner structure. This has justified the flourishing of a large number of tools for the automatic generation of HTML pages based on common specifications.

We have so far developed descriptions for several Web sites, coming from different domains, ranging from museums [14, 10, 4], to bibliography servers [11, 45], to sport sites [15, 3]. From the logical viewpoint, all of these sites have a rather tight structure, and can be described in a very compact way using database structures. Note that none of these sites come from an underlying database. Clearly, in other database-generated sites we have examined [13, 5, 16, 1] the structure is even tighter. Based on the analysis of these sites, we refined our definition of the ADM data model, adopting a subset of ODMG, and enriching it with union types in place of hierarchies and forms. In fact, there is no clear counterpart to inheritance hierarchies in this field, but on the contrary *union types* are essential to model heterogeneity among pages; this is especially true for links, whose target page may in some cases belong to the union of several page-schemes (example: for a paper, a link to a conference or a journal page). Also, the form-based access mechanism in the Web, often used as an alternative to links, needed to be incorporated in the data model.

The ADM description of a site is derived *a posteriori*, on the basis of a "reverse engineering" process. So far, this is done mainly by hand, with the support of several tools, but we are now studying techniques to automate this process. What we do is essentially to run robots on the site to map the site content and graph topology; then, based on the graph, we examine a sample of pages, to identify their attributes. Each set of logically homogeneous pages is described by a page-scheme. Even for very large sites, this activity can be completed in a few hours. This of course does not conclude the reverse engineering phase, since appropriate wrappers need to be written in order to build, from HTML pages in the sites, an internal representation of data as instances of the data model. The process of writing wrappers for a Web site can be much more delicate and demanding than expected.

3.2 Wrapping

Wrapping a site consists essentially of mapping logical access to attribute values in a page at the ADM level, to physical access to text in the HTML source. The natural candidate to write wrappers for HTML documents seemed at first to be context-free grammar parsers. These have in fact been used with success in other, more controlled frameworks [17, 26]. Unfortunately, we soon realized that HTML pages are far too complex to be captured by ordinary parsers. In fact, although sites have often a rather tight *logical* structure—i.e., pages can be easily split in homogeneous sets, all pages in a set having essentially the same attributes—they are often very heterogeneous from the *physical* viewpoint, that is, the actual HTML of two apparently similar pages may be radically different. There are a number of reasons for this, the main one being that HTML documents often present heterogeneities and exceptions, or even *errors*; in fact, browsers do not parse the HTML sources they access, and, even in the presence of errors, they manage to display the corresponding page anyway; as a consequence, it is rather frequent the case of pages that do not fully comply to HTML grammar rules. Just to mention one example, in one of the sites we have examined, some of the HTML sources were actually binary files in Microsoft Word format; nevertheless, browsers

somehow managed to display the page content, with minor noise.

We soon realized that more flexibility with respect to the one granted by grammars was needed. So, we resorted to a *procedural language*: EDITOR. EDITOR [21] is a language to search and restructure textual documents. It is implemented as a Java-based abstract data type for managing documents, upon which searches can be performed using simple patterns, and restructuring by cutting and pasting regions among documents. Being procedural, EDITOR behaves well in presence of exceptions, which can be explicitly caught in the control-flow of the language, and managed separately on the basis of a case-by-case analysis. All of our wrappers are currently written in EDITOR, which has proven to be a very flexible means to wrap textual data sources.

However, it has the usual disadvantage of procedural programming, namely, the lack of declarativity. In fact, the programmer has to explicitly take care of all details of the wrapping, and this in some cases makes the code rather long and boring to write; even more serious, this has a negative influence on the process of maintaining a wrapper. In fact, Web sites are essentially dynamic, and tend to evolve. Although, in our experience, it is quite rare that the overall logical scheme of a site changes—it never happened to the sites we have examined in the last couple of years—it may happen that the *presentation* of data inside pages, i.e., the physical HTML organization, changes—this happened to one of the sites we had wrapped and forced us to maintain our wrappers.

Having a more compact and declarative formalism for defining wrappers became soon a major necessity, in order to be able to easily change and maintain them. This originated MINERVA [31], an attempt to find a compromise between a declarative, grammar-based approach, and the flexibility of procedural programming. MINERVA essentially incorporates an explicit exception handling mechanism inside context-free parsers. An example of wrapper written using MINERVA is reported in Figure 4; it refers to the page containing the list of all conferences in the DBLP site at Trier [45]. The user defines a set of productions, describing the grammar of the document; the system generates the actual wrapper by translating these declarative specifications into EDITOR code. During the parsing, if one of the productions fails, an exception is raised. This can be explicitly captured by the grammar using the `EXCEPTION` clause associated with every production. The clause reports a piece of (procedural) EDITOR code, which is run against the document in the attempt to restructure it and take care of the exception. If this succeeds, the parsing goes on where it was suspended. The same techniques can be profitably used to wrap non-HTML data sources such as BibTex files.

3.3 Query Paradigms and Query Interfaces

Once wrappers for a site have been written, the user can express queries using ULIXES [22]. ULIXES is an SQL-like language for ADM objects, which essentially implements a navigational algebra. Each query returns a set of tuples; these are built by navigating a path in a scheme, with selection and projection conditions. Following is an example of an ULIXES query on the Uffizi Web site [14], whose scheme is shown in Figure 3; the query retrieves the title and position (room name) of all paintings by Michelangelo in the museum. Whenever the query is executed, it starts downloading HTML pages from the site, it wraps them, computes the query and stores the result in a local database table called `MichelangeloPaintings`.

```
PAGE AllConferences

$AllConferences :    *<hr> ( $ConfWithInitial )+

$ConfWithInitial :   <h3><a name=*"> $Initial </a></h3>
                     [<ul>
                           (
                             <li><a href="$ConfURL"> $Acronym </a>
                             $TmpConfName $INSERT_TUPLE
                           )*
                     </ul>]

$TmpConfName :       - $ConfName
EXCEPTION (*(?<li>|</ul>)
                     {
                       $ConfName.reset();
                       $Acronym.copyAll();
                       $ConfName.paste();
                     }

$INSERT_TUPLE :      {
                       $Initial  char(1);
                       $Acronym  char(10);
                       $ConfName char(100);
                       $ConfURL  char(100);
                     }
```

Figure 4: MINERVA code for the list of conferences on the DBLP Site at Trier

```
DEFINE TABLE MichelangeloPaintings (Title, RoomName)
AS           RoomsPage.RoomList.ToRoom → RoomPage.PaintList.ToPaint → PaintingPage
IN           UffiziScheme
USING        PaintingPage.Title, RoomPage.RoomName
WHERE        PaintingPage.Painter LIKE '%Michelangelo%'
```

Implementing the prototype of ULIXES posed several interesting challenges. In fact, the system works on ADM objects that are purely virtual, and have to be constructed on the fly by downloading and wrapping actual pages on the site. When the system tries to cross a link and accesses a new page, due to the presence of union types it cannot statically determine the type—i.e., page-scheme—of the page. Therefore, it is forced to adopt a form of dynamical type casting in order to select the right wrapper and methods to apply to the page. This has a number of subtleties. The solution we adopted was to determine, for each page-scheme, a "certificate", that is, an invariant property of HTML sources that uniquely allows to identify instances of that page-scheme. When a page is downloaded, we check its certificate to determine its type and then correspondingly wrap and store it. Another interesting problem was related to managing forms. In our approach, a form is seen as a virtual list that associates a link to a result page with each set of parameters. The language hides all details relative to crossing forms (parameters are specified as conditions in the

WHERE clause) but this requires a different treatment for different form methods (GET or POST with or without redirection).

Note that, with respect to other languages for Web exploration, like WebSQL or W3QS, ULIXES is non-recursive. Although this prevents, for example, from expressing forms of transitive closure on Web sites, in our perspective it does not represent a serious limitation. In fact, the model it refers to is not simply a graph, but an object-oriented ADM scheme; our experience is that, inside a site, cycles may be present, but they mainly serve for navigational purposes and have little semantics (consider for example all links that from the pages of a site allow to reach directly the home page; we usually don't even model these links in the ADM scheme). Also, our experiments with transitive closure of Web pages proved that query results tend to grow very fast, and usually make the computation unacceptably slow.

ULIXES was primarily conceived as a language for writing applications on the Web. However, it may also be used as a tool for casual queries. This is especially useful for very large Web sites, like the DBLP site, where retrieving data based on complex conditions—for example "*find all authors who had papers in the last three VLDB conferences*"—is hardly feasible by simple browsing. To experiment the effectiveness of the language, we made a prototype available to a number of users. From this experience, we have learned that—although quite intuitive—the syntax of the language discourages non-expert users from writing queries from scratch. Casual users found annoying the need to explicitly specify the path to be navigated in the scheme. We therefore started thinking to alternative query interfaces for the system. Here are several alternatives we have considered.

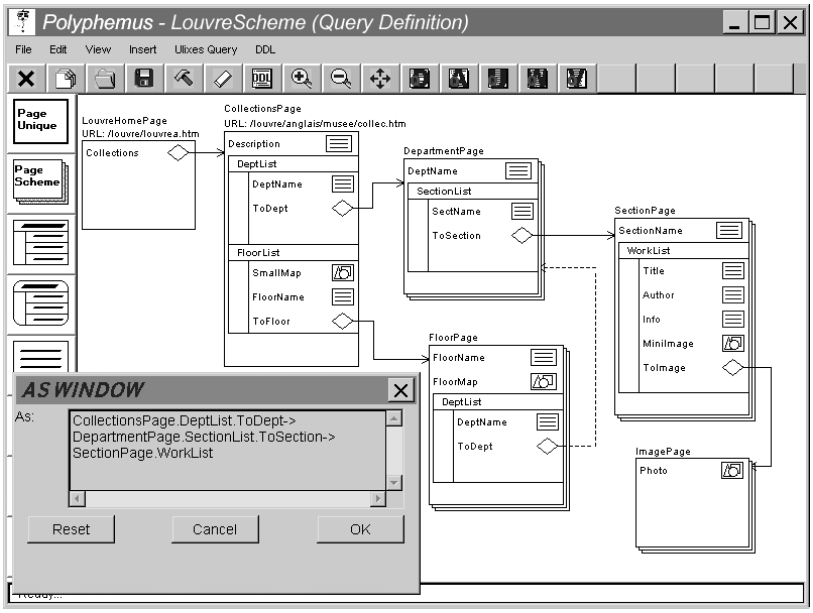

Figure 5: Diagrammatic Queries using POLYPHEMUS

The easiest way to give access to a site is to generate a number of forms corresponding to pre-determined ULIXES queries in which users can simply fill-out their parameters and then run the query; for example, with respect to the DBLP site at Trier, we may allow users to change the

name of the author s/he is interested in, the conference or journal, the year of publication etc. Although very practical, this solution has the clear disadvantage of making the query process very rigid, since only a number of pre-defined paths can be navigated in the site.

Then, we have considered the avenue of a visual paradigm; to do this, we developed POLYPHEMUS, a diagrammatic query interface for expressing ULIXES queries. A snapshot of a POLYPHEMUS screen is reported in Figure 5: it shows the process of expressing a query on the Louvre Museum Web site [10]. The user can specify a query by interacting with a graphical representation of the ADM scheme; a path is selected by simple mouse clicks on page-schemes, and selection and projections can be easily specified along the path using dialog windows. Then, the user can invoke ULIXES to run the query on the site, and browse the results. We are testing the effectiveness of the formalism. It seems that the main advantages of such approach stand in a more natural perception of the way the site is navigated to reach data of interest.

However, a major drawback in terms of performance is that in this way the user is forced to choose a path in the site, and, in the frequent case in which alternative paths to the same data are available, it is not guaranteed that the chosen one is also the optimal one. In fact, as it is discussed in the next section, the presence of alternative access paths is a peculiarity of Web hypertexts. Consider again the DBLP site at Trier and the query "*find all authors who had papers in the last three VLDB conferences.*" The query could be answered by following different paths: (*i*) starting from the home page, follow the link to the list of conferences, from here to the VLDB page, then to each of the last three VLDB conferences, extract a list of authors for each, and intersect the three lists; (*ii*) as above, but go directly from the home page to the VLDB page (there is a link). (*iii*) go through the list of authors, for each author to the list of their publications, and keep those who have papers in the last three VLDB's. If we use number of pages accessed as a rough measure of query execution cost, we see there are large differences among these possible access paths, in particular between the last one and the other two. There are over 16,000 authors represented in this bibliography, so the last access path would retrieve several orders of magnitude more pages than the others.

An alternative and promising approach consists in building relational views over a site, and allow users to express queries over these abstractions. We essentially give users the illusion of interacting with a bunch of database tables, which can be queried using SQL. For example, with respect to the DBLP site, this relational abstraction would include tables like:

```
Conference(ConfName, Location, Year);
Author(AuthorName, Home-Page);
AuthorConference(AuthorName, ConfName);
...
```

In general, a declarative query will admit different translations, corresponding to different navigation paths to get to the data. We leave it to the system to translate these declarative queries into navigation of the underlying hypertext. A specific optimizer, CIRCE [48], generates a number of query plans and, based on a cost model, selects an optimal one.

4 Site Creation and Management

We have experienced and tuned our tools in the generation and maintenance of many Web sites in different contexts. Beside some toy applications, we want to mention two official sites of large size: the Faculty of Engineering at University of Basilicata [6], containing more than 500 pages of information about people, education and initiatives of the faculty; and the Web site of the IV Surgical Clinic, at the Umberto I Hospital in Rome (in preparation).

In both cases, the design was conducted according to the ARANEUS Web Site Design Methodology [23], which is based on a clear separation of data management tasks from page design and implementation. Data management is supported by the DBMS, in which all data to be published on site are stored. The logical structure of pages is designed with the help of ADM. Then, PENELOPE [22] is used to map the hypertext structure onto the database and generate pages.

4.1 Different Design Levels and the Need of a Methodology

A key feature of our approach to Web site design that we want to emphasize here is the clear distinction among three different levels: (*i*) database design; (*ii*) hypertext design; (*iii*) presentation design. The separation is justified by the observation that the three levels are largely independent. For example, differently from databases, hypertexts are very redundant data-sources; the same piece of information can occur several times in a site (consider, for example, the name of a course, repeated in every page in which there is a link to the course page); also, to make browsing more effective, usually several different access paths to the same information are given (access to publications may be either by research topic or by year etc.); on the contrary, we would like to store data with as little redundancy as possible, in order to avoid inconsistencies or update problems. Also, the organization of a site may be restructured to make it more effective even without changing the underlying data. Therefore, the right approach seems to have all data stored in a database, and design the site as a hypertextual view over the database.

This independence is also the reason for which we adopted relational databases instead of object-oriented ones as a repository for data to be published in the site. Since ADM can be considered as a subset of ODMG, the adoption of an OODBMS could have appeared as a more "natural" solution: page-schemes could be implemented by classes and each page stored as an object. However, such a tight coupling of the database with the hypertext structure introduces several problems; first, it forces to have a high degree of redundancy in the database schema to model different access paths to data; then, in case the hypertext undergoes a restructuring, all classes have to be changed in a coherent way, requiring a tremendous and expensive effort; the problem becomes even more evident if the same database is shared with other applications. A possible solution consists in defining views over the OO database, each view corresponding to a page-scheme; however, to the best of our knowledge, none of the commercial OODBMS supports the definition of views yet.

Similarly, the presentation of data in a page may often vary even if the underlying logical structure remains the same. In our approach, we associate an HTML template file with each page-scheme. The HTML template completely specifies the layout of pages corresponding to that page-scheme, and can be changed independently from the page-scheme structure. When PENELOPE generates an instance of a page-scheme, it extracts attribute values from the database and merges

them with the corresponding template. It is worth noting that, in our experience, designing an appealing layout for pages is a complicated activity, some times even more delicate than designing the site itself.

Based on these ideas, it can be seen how the design of a Web site is a complex task, which requires to deal with data under different perspectives, and map the one onto the others. For large and complex Web sites, the complexity of this process can only be reduced by the adoption of a systematic design methodology, i.e., a set of models and design steps that lead from a conceptual specification of the domain of interest to the implementation of the actual site.

4.2 Maintaining the Site and the "Push or Pull" Problem

One aspect, often underestimated, of a Web site life-cycle is the need to maintain the site. Leaving aside major restructurings of the hypertext organization or changes in the presentation, it is still necessary to periodically update the database.[2] Also, updates to the database should be correspondingly reflected on the site.

Note that two different approaches to Web publishing are possible in this context. Database products on the market adopt *pull* techniques, in which pages contain calls to the DBMS, and, when the user requests a new page, such calls are evaluated, the page is generated on the fly and returned to the browser. The main advantage of this approach is that pages always reflect the most recent database state; however, there are at least two limitations associated with it.

- First, if the underlying database has to be used also for other ends—for example, like a repository for a company information system—frequent accesses to the Web site may considerably increase the load on the database and can slow down the overall performance; on the other side, creating a new database especially intended for Web publishing purposes may not be economically feasible and poses further problems to guarantee consistency between the two repositories.

- Second, the resulting Web site is strongly platform-dependent: the HTTP server needs a specific DBMS as a back end to serve pages, which often contain non-standard tags to invoke the execution of scripts; this means, for example, that such a site cannot be mirrored or distributed over the network, nor moved to another platform without also migrating the DBMS.

An alternative is represented by a *push* approach, in which data are materialized in HTML files and 'pushed' to the site. This clearly solves the problems above, since the resulting site is standard and the HTTP server works independently from the DBMS; however, in this case, the management of pages in presence of updates is more complex; in fact, when the database is updated, also materialized HTML files need to be correspondingly maintained to reflect the change.

[2] In many cases, updates cannot be performed by the site designer, but need to be done by authorized end-users. If data is stored in the database, this requires to update the database. Also, for unskilled users, this has to be done without resorting to complex SQL statements. When this problem arose for our faculty site, we immediately saw that the site we had designed was incomplete, since it did not provide support for updates. We decided to enrich the site itself with a collection of password-protected pages containing forms to update the database. In essence, the site became a full-fledged information system, through which users could access, insert, delete and update data.

Since push techniques are becoming increasingly popular on the Web due to the appearance of *channels*, i.e., sites that periodically deliver pages or portions of sites directly to the client machine, we decided to support both approaches: in a site, pages can be kept virtual, and generated on-the-fly, or materialized in HTML files. In order to enforce consistency between the database and HTML pages, we had to develop a page-update language and a suitable algorithm for incremental page maintenance [54]. The page update language, called PENELOPE *Manipulation Language* (PML), provides two instructions: `Generate` and `Remove`, which can refer to: (*i*) the whole site; (*ii*) all instances of a page-scheme; or (*iii*) pages that satisfy a condition in a `Where` clause. The page-maintenance algorithm takes as input a database update, and returns a minimal set of PML instructions needed to correspondingly update the pages. In essence, when an update to the database is requested to the system, it automatically generates a *mixed transaction*, in which SQL updates to database tables and PML updates to pages are combined in order to guarantee consistency between the two. The transaction is then atomically executed against the database and the Web site.

4.3 Queries and Meta-Information

When implementing PENELOPE, one of our objectives was to be able to seamlessly extend the query process described in the previous section also to internal sites.[3] This somehow required to keep track of the ADM scheme of the site when generating HTML files, and make it accessible to ULIXES without the need of developing ad-hoc wrappers. The way we did it 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 ULIXES in order to dynamically wrap the page and extract relevant pieces of information.

Figure 6 shows a sample HTML source generated by PENELOPE. Some hidden tags have been embedded in HTML comments. 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. In essence, the `PAGE-SCHEME` tag in the page-header represents a DTD for the page, in the spirit of XML, a subset of SGML [40] explicitly conceived for the Web and currently under definition (see [35] for an interesting overview). However, our tagging mechanism is fully embedded in HTML and does not require extensions. Also, it can be a basis for the definition of extensions of other query languages, such as *W3QS* [41] or *WebSQL* [49], in order to exploit the schema information in querying the site.

[3] It may be argued that, since internal sites are generated from a database, the easiest way to query the site content would be to grant direct access to the underlying database. This can be done, for example, by providing suitable forms using which users can express SQL queries directly on the database. This, however, has the same disadvantages we have discussed above with respect to pull approaches (the queries increase the database workload, and makes the site platform dependent).

```
<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">
... omissis ...
<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>
        ... omissis ...
        <!--/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/franksm.html"><!--/ToMemberPage-->
              <!--Name-->Frank Smith<!--/Name--></A></LI>;<P>
          ... omissis ...
        <!--/MemberList--></UL></TR>
</TABLE>
... omissis ...
```

Figure 6: A sample HTML source generated by Penelope

5 Site Integration

The two activities described in the previous sections can be nicely coordinated in a larger framework, aiming at integrating data coming from different Web sites. This form of "data-oriented cooperation" requires to identify a number of Web sites containing homogeneous data, extract pieces of information from the sites, correlate these data and then publish everything in a new site, which offers an integrated and possibly reorganized perspective over the original ones.

This is what we have done in developing the *Integrated Web Museum* [8], a Web site containing data and images about paintings in the Uffizi, Louvre and Capodimonte [4] Web sites. This initiative has had an unexpected success, showing that there is great interest for this kind of application: ever since the site has been indexed by some popular index servers, thousands of accesses per day have been registered.[4]

The museum was developed according to a several-step process that we can summarize as follows: (*i*) *relational view definition and data extraction:* once the original sites have been described in ADM and wrapped, we identify portions of interest and extract them using ULIXES; each ULIXES query defines a relational view over the original sites; (*ii*) *relational view integration:* these views are then processed using the local relational DBMS, to generate an integrated view; also local tables and/or wrapped data sources (files) can be incorporated in the integration process; (*iii*) *hypertextual view definition:* finally, a new Web site is generated as a hypertextual view—defined using PENELOPE—over the integrated relational view. Our approach is therefore to define the global system as a view over the original data sources (see [56] for a discussion of alternative approaches). In the following, we discuss the main issues we have dealt with in developing this application, trying to highlight the problems we faced, and the solutions we adopted.

5.1 Dealing with Schematic and Semantic Heterogeneities

The overall process is rather involved because of the different view levels and models. These levels take care of eliminating heterogeneities between the original data sources. These are essentially of two kinds [52]: *schematic heterogeneities*—i.e., related to the way data are organized at the original data sources—and *semantic heterogeneities*, i.e. related to the way data are represented at the original data sources.

We use our multi-layer view approach to progressively reduce schematic heterogeneities. We want to emphasize the fact that, on the Web, logically similar data may be organized in a radically different way due to presentation choices. Coherently with our approach, we split the two aspects and use the relational level to reason about the logical organization of data, and the hypertext one to reason about the presentation. As an example, it can be seen from Figure 3 that paintings in the Uffizi Gallery are organized by rooms (one page for each room with a list of paintings); in comparison, in the Louvre and Capodimonte sites, they are organized in collections (or departments) (we have omitted the schemes for space reasons; they can be found at [2]). However, beside the specific access paths, the actual data have a very similar logical structure, namely a collection of

[4] Accesses to the Integrated Museum are in fact overloading our HTTP Server, so that we are now considering the idea of moving it to a separate server.

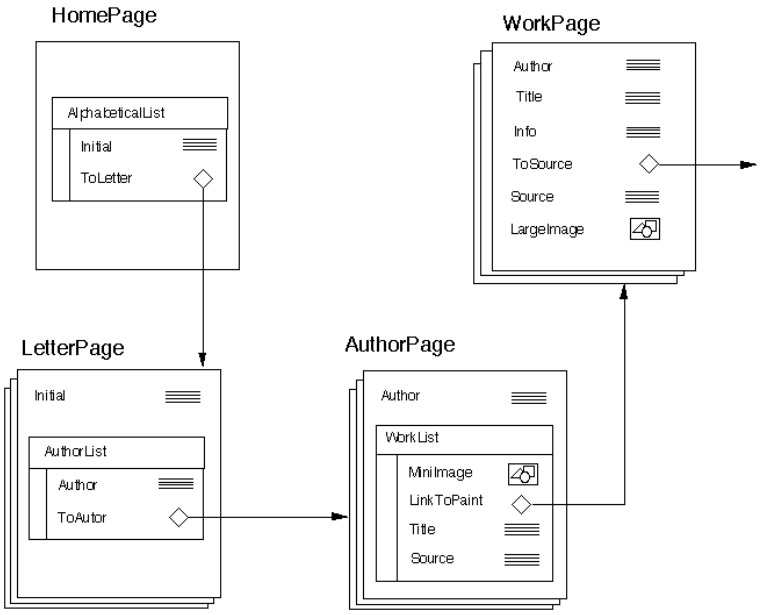

Figure 7: ADM Scheme for the Integrated Web Museum

artists and a collection of works. This form of heterogeneity is dealt with by ULIXES, which takes care of giving a table-based perspective over data in hypertext form.

Once this has been done, integrating the two data-sets amounts to finding the union of two collections. Our approach to this step is rather conservative, in the fact that the logic of the integration process—what is sometimes called the *mediator layer*—is based on SQL, i.e., the integrated relational view is defined as an SQL view over the component tables defined by ULIXES. This, in our experience, is satisfactory in the majority of cases. Then, the final view definition step in PENELOPE takes care of establishing how data will be organized in the integrated site. In this case (see Figure 7) we have chosen still another organization with respect to the ones in the original sites, in which paintings are organized by author, in turn grouped by initial.

Semantic heterogeneities are much harder to manage. A key problem we faced was that of inconsistencies between identifiers. For example, Michelangelo is reported as "Buonarroti Michelangelo" on the Uffizi and Capodimonte Web sites, and simply as "Michelangelo" on the Louvre Site. There are less known authors which are reported with three different names in the three sites. Even in different pages of the same site, the same author may in some cases be referenced to with different names. We were forced to manually write conversion tables in order have unique identifiers for each painter. Statistical approaches seem to be necessary in these cases, but they are far from being a final solution to the problem.

5.2 Virtual Views and Materialized Views

One of the questions we soon had to address was the choice between materializing the integrated site or leaving it virtual. Both solutions have advantages and disadvantages, and require radically different technical support. The problem is clearly related to *updates* in the local sites, i.e., insertions

or deletions or updates to pages. Assuming these updates do not change the structure of the site—that is, the ADM scheme remains valid—still the global site should be accordingly updated.

In principle, it would be nice if pages in the integrated museum were left virtual and users could retrieve them based on a purely pull approach. This in fact would guarantee that data in the global site always reflect the last state at the local sites, and no obsolescence can occur. Now, consider a user requesting a page of paintings by Michelangelo from the global site. This request would trigger the execution of a program which performs the following tasks: (*i*) first, it identifies the portion of the local sites relevant for the user request; this can be done by appending selection conditions (stating that the author must be Michelangelo) to the ULIXES queries that navigate the site; (*ii*) then, ULIXES is run to navigate the sites accordingly; pages are wrapped and data are temporarily stored in local database tables; (*iii*) SQL views are computed on the local data to generate an integrated data-set—i.e. a bunch of tuples, one for each painting by Michelangelo—to serve as a basis for the generation of the page; (*iv*) finally, PENELOPE is used to generate the page, which is returned to the user. This process may in some cases require an unacceptable amount of running time. This is due essentially to two factors: first, downloading pages from the network is sometimes slow; second, and more important, to generate a single page in the global site usually many pages need to be downloaded from the local sites. In fact, the original sites may not offer suitable access paths to information of interest: in the Uffizi Web site, in order to retrieve all paintings by Michelangelo one must navigate all rooms, since there is no direct access to paintings based on author names.

We therefore had to adopt a materialized solution, in which pages in the global site are warehoused locally. This drastically reduces the time needed to serve a page, but of course does not guarantee that the global site always reflects all changes in the local sites and poses a serious problem of incremental view maintenance. A reason for which this problem is different from other similar problems is that we have absolutely no control over the local sites. The first problem in maintaining the view is detecting changes in the local sites, because we cannot assume that these are notified by the site managers. Of course, a brute-force approach would do—i.e., periodically navigating the whole local sites, and re-computing the ULIXES views from scratch. This is unsatisfactory for various reasons. In fact, it performs a large amount of work, unnecessarily loading the network and the servers. Usually, changes will affect a limited number of pages with respect to the overall site. We therefore would like to restrict the need to navigate the site to these few pages. A useful technique we use to reduce the network load when checking local sites for changes consists in storing in a local index the date of last update of each page in a site (this is a standard parameter exchanged between server and client during an HTTP connection). Then, when the site is checked for updates, "light connections" to the server are opened to check dates, in order to detect changes without the need of downloading the whole page source. Still, the development of efficient algorithms for change detection and propagation needs to be studied.

6 Related Work

Several systems recently presented in the literature address the problem of managing data coming from the Web. However, they either concentrate on designing query languages for the Web, or on

developing tools for Web site generation.

W3QL [41], WebSQL [49] and WebOQL [19], and, with slightly different focus, Lorel [18] and UnQL [27] are prominent examples of query systems designed for semistructured and Web data. The main difference with respect to ARANEUS stands in the choice of the data model: all of these proposals adopt variants of a simple graph-based data model, and concentrate on the development of query languages for these structures. Moreover, there is no notion of scheme similar to the one of ARANEUS. Other proposals—WebLog [43], ADOOD [36] and FLORID [39]—advocate the use of logic as a formalism for querying the Web.

A notion of scheme similar to the one introduced in ARANEUS has been recently used in WGLog [32], whose aim is at studying graph-based query languages for the Web, and in WAG [28], which also studies mining and integration problems in the Web framework.

Languages and tools for wrapper generation were first studied in the context of textual databases (see, for example, [17]), and then with specific reference to the Web [38, 20, 55]. All of these approaches use variants of grammars to describe patterns in documents. As discussed in the previous sections, grammars are not completely satisfactory in this context. We therefore try to combine the advantages of declarativity with the flexibility of a procedural language for dealing with exceptions.

Other proposals, namely TSIMMIS [30] and the Information Manifold [44] aim at integrating data from heterogeneous sources, including the Web. These techniques can be used in ARANEUS in order to correlate tabular data and generate integrated views. Integration and queries over data coming from the Web is also the goal of the WebSuite project [25], a collection of modules for wrapping, querying, translating [50] and integrating data from the Web.

Web-site generation is another fertile area (see, for example, [33, 53, 51]). These proposals deal with the problem of implementing a Web site as a view over a set of data sources. They mainly adopt a graph-based model, in the spirit of OEM [30, 18], and have no notion of schema of a site. A different approach is the one undertaken in AutoWeb [34], in which a data model inspired by hypermedia authoring is used to describe a site. Deciding the organization of data in the site is an activity supported by a specific design methodology; based on this design phase, data stored in a relational database is translated into HTML pages.

Several commercial database systems (see, for example, [12, 7]) now provide functionalities for the automatic generation of pages. However, also in that case, no data model is used to describe pages and hypertexts. Moreover, these proposals tend to adopt a pull approach to Web publishing, whereas we also support materialized approaches.

7 Conclusions

The ARANEUS Web-base Management System represents a proposal towards the definition of a new kind of data-repository, to serve as a basis for Web-based information systems.

We have presented several experiences maturated in the development of the system, and in its use as a support for implementing a number of applications, trying to give an overview of the problems we have encountered and of the solutions we adopted. We focused on the activities of describing, wrapping, querying, designing, implementing, maintaining and integrating Web sites; for

each of these tasks, we had to reconsider and adapt traditional concepts and techniques developed in the database field.

A number of promising research directions still need to be investigated, as follows.

- **Inferring Structure from the Web**: the reverse-engineering of a Web site would greatly benefit from techniques for automatic schema-finding and wrapping; also, this would reduce the need for schema and wrapper maintenance in the case one site undergoes a major restructuring.
- **Optimization Techniques for Web Queries**: it has been shown in [48] that cost models and optimization strategies for the Web may be significantly different from the ones developed for traditional and object-oriented databases; if the Web becomes the preferred medium for data exchange and access, this issue requires careful investigation.
- **Efficient Algorithms for Change Detection in Web Sites**: if database views are warehoused locally to reduce computation costs, then it is necessary to develop algorithms for efficiently checking Web sites in order to detect updates, i.e., insertions or deletions of pages.
- **Algorithms for Incremental Hypertext View Maintenance**: related to the previous point, if new sites have been built as an hypertext view over existing ones, and pages have been materialized locally, whenever a change is detected in the original pages, the hypertext view should be correspondingly maintained.

References

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

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

[3] Britannica Sporting Record: The Olympic Games. `http://sports.eb.com`. Access to the site is restricted to authenticated users.

[4] The Capodimonte Museum Web site. `http://capodimonte.selfin.net`.

[5] Elsevier Science. `http://www.elsevier.nl`.

[6] Faculty of Engineering at University of Basilicata. `http://www.ing.unibas.it`. In italian.

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

[8] The Integrated Web Museum. `http://poincare.dia.uniroma3.it:8080/penelope/webmuseumNEW/-webmuseum.html`.

[9] JDBC Database Access API. `http://www.javasoft.com/products/jdbc/jdbc.html`.

[10] The Louvre Web site. `http://www.louvre.fr`.

[11] The Oncolink Web site. `http://www.oncolink.upenn.edu`.

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

[13] Springer Science on-line. `http://www.springer.de`.

[14] The Uffizi Web site. `http://www.uffizi.firenze.it`.

[15] World-wide Track and Field statistics on-line. http://www.uta.fi/~csmipe/sport/.

[16] Yahoo Yellow Pages. http://yp.yahoo.com.

[17] S. Abiteboul, S. Cluet, V. Christophides, T. Milo, G. Moerkotte, and J. Siméon. Querying documents in object databases. *Journal of Digital Libraries*, 1(1):5–19, April 1997.

[18] 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.

[19] G. O. Arocena and A. O. Mendelzon. WebOQL: Restructuring documents, databases and Webs. In *Fourteenth IEEE International Conference on Data Engineering (ICDE'98), Orlando, Florida*, 1998. To Appear.

[20] N. Ashish and C. Knoblock. Wrapper generation for semistructured Internet sources. In *Proceedings of the Workshop on the Management of Semistructured Data (in conjunction with ACM SIGMOD 1997)* http://www.research.att.com/~suciu/workshop-papers.html, 1997.

[21] P. Atzeni and G. Mecca. Cut and Paste. In *Sixteenth ACM SIGMOD Intern. Symposium on Principles of Database Systems (PODS'97), Tucson, Arizona*, pages 144–153, 1997. http://poincare.dia.uniroma3.it:8080/Araneus/.

[22] 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*, pages 206–215, 1997. http://poincare.dia.uniroma3.it:8080/Araneus/.

[23] P. Atzeni, G. Mecca, and P. Merialdo. Design and maintenance of data-intensive Web sites. In *VI Intl. Conference on Extending Database Technology (EDBT'98), Valencia, Spain, March 23-27*, 1998. To Appear.

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

[25] C. Beeri, G. Elber, T. Milo, Y. Sagiv, O. Shmueli, N. Tishby, Y. Kogan, D. Konopnicki, P. Mogilevski, and N. Slonim. WebSuite – a tools suite for harnessing Web data. In *Proceedings of the Workshop on the Web and Databases (WebDB'98) (in conjunction with EDBT'98)* http://poincare.dia.uniroma3.it:8080/webdb98, 1998.

[26] G. E. Blake, M. P. Consens, P. Kilpeläinen, P. Larson, T. Snider, and F. W. Tompa. Text/relational database management systems: Harmonizing SQL and SGML. In *First Intern. Conf. on Applications of Databases, (ADB'94), Vadstena, Sweden. LNCS 819*, pages 267–280. Springer-Verlag, June 1994.

[27] 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.

[28] T. Catarci, L. Iocchi, D. Nardi, and G. Santucci. Conceptual views over the Web. In *Proceedings of the Fourth Workshop on Knowledge Representation meets Databases (KRDB'97) (in conjunction with VLDB'97)* http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-8/, 1997.

[29] R. G. G. Cattel. *The Object Database Standard ODMG-93*. Morgan Kaufmann Publishers, San Francisco, CA, 1994.

[30] 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.

[31] V. Crescenzi and G. Mecca. Grammars have exceptions, 1998. Submitted for Publication.

[32] E. Damiani and L. Tanca. Semantic approaches to structuring and querying Web sites. In *IFIP Working Conference on Database Semantics*, 1997.

[33] 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.

[34] P. Fraternali and P. Paolini. A conceptual model and a tool environment for developing more scalable, dynamic, and customizable Web applications. In *VI Intl. Conference on Extending Database Technology (EDBT'98), Valencia, Spain, March 23-27*, 1998.

[35] L. M. Garshol. Introduction to XML, 1998. `http://www.ifi.uio.no/~larsga/download/xml/-xml_eng.html`.

[36] F. Giannotti, G. Manco, and D. Pedreschi. A deductive data model for representing and querying semistructured data. In *Proceedings of the Workshop on Logic Programming Tools for Internet Applications (in conjunction with ICLP'97), Leuven, July 11, 1997* `http://http://clement.info.-umoncton.ca/%7Elpnet/proceedings97/`, 1997.

[37] G. H. Gonnet and F. W. Tompa. Mind your grammar: a new approach to modelling text. In *Thirteenth International Conference on Very Large Data Bases, Brighton (VLDB'87)*, pages 339–346, 1987.

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

[39] R. Himmeroeder, G. Lausen, B. Ludaescher, and C. Schlepphorst. On a declarative semantics for Web queries. In *Fifth International Conference on Deductive and Object-Oriented Databases (DOOD'97), Montreux, Switzerland, December 8-11*, 1997.

[40] ISO. International Organization for Standardization. ISO-8879: Information Processing – Text and Office Systems - Standard Generalized Markup Language (SGML), October 1986.

[41] 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.

[42] D. Konopnicki and O. Shmueli. Bringing database functionality to the WWW. In *Proceedings of the Workshop on the Web and Databases (WebDB'98) (in conjunction with EDBT'98)* `http://poincare.-dia.uniroma3.it:8080/webdb98`, 1998.

[43] 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.

[44] 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.

[45] M. Ley. DataBase systems and Logic Programming bibliography site. `http://www.informatik.-uni-trier.de/~ley/db/index.html`. Also accessible from `http://dblp.uni-trier.de`.

[46] A. Loeffer. Text databases: A survey of text models and systems. *Sigmod Record*, 23(1):97–106, March 1994.

[47] G. Mecca, P. Atzeni, A. Masci, P. Merialdo, and G. Sindoni. The Araneus Web-base management system. In *ACM SIGMOD International Conf. on Management of Data (SIGMOD'98), Seattle, Washington*, 1998. Exhibition Section.

[48] G. Mecca, A. Mendelzon, and P. Merialdo. Efficient queries over Web views. In *VI Intl. Conference on Extending Database Technology (EDBT'98), Valencia, Spain, March 23-27*, 1998. To Appear.

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

[50] T. Milo and S. Zohar. Schema-based data translation. In *Proceedings of the Workshop on the Web and Databases (WebDB'98) (in conjunction with EDBT'98)* http://poincare.dia.uniroma3.it:8080/-webdb98, 1998.

[51] F. Paradis and A. M. Vercoustre. A language for publishing virtual documents on the Web. In *Proceedings of the Workshop on the Web and Databases (WebDB'98) (in conjunction with EDBT'98)* http://poincare.dia.uniroma3.it:8080/webdb98, 1998.

[52] A. P. Sheth and J. A. Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys*, 22(3):183–236, September 1990.

[53] G. Simeon and S. Cluet. Using YAT to build a Web server. In *Proceedings of the Workshop on the Web and Databases (WebDB'98) (in conjunction with EDBT'98)* http://poincare.dia.uniroma3.-it:8080/webdb98, 1998.

[54] G. Sindoni. Incremental maintenance of hypertext views. In *Proceedings of the Workshop on the Web and Databases (WebDB'98) (in conjunction with EDBT'98)* http://poincare.dia.uniroma3.-it:8080/webdb98, 1998.

[55] D. Suciu Ed. Proceedings of the workshop on the management of semistructured data (in conjunction with ACM SIGMOD 1997) http://www.research.att.com/~suciu/workshop-papers.html, 1997.

[56] J. D. Ullman. Information integration using logical views. In *Sixth International Conference on Data Base Theory, (ICDT'97), Delphi (Greece), Lecture Notes in Computer Science*, pages 19–40, 1997.