Semistructured and Structured Data in the Web: Going Back and Forth

Paolo Atzeni,[1] Giansalvatore Mecca,[2] Paolo Merialdo[1]

[1] Università di Roma Tre [2] Università della Basilicata

email: {atzeni,mecca,merialdo}@inf.uniroma3.it

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

1 Introduction

Database systems offer efficient and reliable technology to query structured data. However, because of the explosion of the World Wide Web [11], an increasing amount of information is stored in repositories organized according to less rigid structures, usually as hypertextual documents, and data access is based on browsing and information retrieval techniques.

Since browsing and search engines present important limitations [8], several query languages [19, 20, 23] for the Web have been recently proposed. These approaches are mainly based on a loose notion of structure, and tend to see the Web as a huge collection of unstructured objects, organized as a graph. Clearly, traditional database techniques are of little use in this field, and new techniques need to be developed.

In this paper, we present the approach to the management of Web data as attacked in the Araneus project carried out by the database group at Università di Roma Tre. Our approach is based on a generalization of the notion of *view* to the Web framework. In fact, in traditional databases, views represent an essential tool for restructuring and integrating data to be presented to the user. Since the Web is becoming a major computing platform and a uniform interface for sharing data, we believe that also in this field a sophisticate view mechanism is needed, with novel features due to the semi-structured nature of the Web. First, in this context, restructuring and presenting data under different perspectives requires the generation of *derived Web hypertexts*, in order to re-organize and re-use portions of the Web. To do this, data from existing Web sites must be extracted, and then queried and integrated in order to build new hypertexts, i.e., *hypertextual views* over the original sites; these manipulations can be better attained in a more structured framework, in which traditional database technology can be leveraged to analyze and correlate information.

Therefore, there seem to be different view levels in this framework: *(i)* at the first level, data are extracted from the sites of interest and given a database structure, which represents a first *structured view* over the original semi-structured data; *(ii)* then, further database views can be built by means of reorganizations and integrations based on traditional database techniques; *(iii)* finally, a *derived hypertext* can be generated offering an alternative or integrated hypertextual view over the original sites. In the process, data go from a loosely structured organization—the Web pages—to a very structured one—the database—and then again to Web structures.

In this paper, we present a set of tools that support such a view definition process. Since step *(ii)* is based on rather standard database techniques, we focus on the other two steps. A key point in our approach consists in the use of a specific formalism, the Araneus *Data Model* (ADM), for describing the structure of a Web hypertext. Based on the data model, we propose two languages to define views over the Web: the Ulixes language, to define database views over a site, and, symmetrically, the Penelope language, to generate a derived hypertext from a database.

To extract data from a Web site, as a very first step we derive a *scheme* of the site. The use of a specific data model is central in our approach: in fact, in order to reason about hypertextual data, we need to describe its logical organization, abstracting from the physical organization (the pages). We say that ADM is a *page oriented* model, in the sense that the main construct of the model is that of *page scheme*, used to describe the structure of a set of homogeneous pages in a site; the main intuition behind the model is that an HTML page can be seen as an object with an identifier, the URL, and several attributes, one for each relevant piece of information in the page. The attributes in a page can be either *simple*, like text, images, binary data or links to other pages, or *complex*, that is, lists of items, possibly nested. ADM also provides *heterogeneous union* and a *form* type, specifically needed to model the organization of Web pages.

In order to see pages as instances of page schemes we apply suitable text extraction procedures based on the Editor language [9], a new language for searching and restructuring text. Editor programs are a procedural language for manipulating text, based on "cut & paste" operations. Each page is seen as an object with several extraction methods, one for each attribute of the page; these methods access the HTML source of the page and extract the value of the corresponding attribute. In essence, EDITOR programs act as "wrappers" for pages.

Based on these ideas, the scheme of a Web site can be seen as a collection of page schemes, connected using links. This structured view of the site abstracts some of the properties of the pages, reflecting the user's perspective, and provides a high-level description to be used as a basis for successive manipulations.

Based on the scheme, ULIXES and PENELOPE support a two-way data-restructuring process. First, we use ULIXES to build relational views over the Web. Here, the knowledge about the structure based on the ADM description of the site highly facilitates the process of extracting data. To provide a flexible paradigm to access data, we reconsider the issue of *path expressions* [17] in this framework. We introduce the notion of *navigational expression* as a means to express a set of navigations—i.e., paths through pages—in a site; in fact, to access data in the Web, it is natural to start from some entry-point, like, for example, a home page, and navigate until data of interest are found. The main idea, here, is that each navigation can be seen as a tuple of values and each navigational expression as a relation. These relations, which can be locally materialized and queried using any database query language, represent a *database view* over a site.

Once data has been extracted, it can be easily manipulated locally. For example, it can be correlated with local data sets, or perhaps integrated with data coming from different sites, using known techniques (see, for example [24, 18]).

Finally, PENELOPE can be used to present the resulting database as a derived hypertext, that is, a *hypertextual view* over the original data sources that can be explored using a Web browser. PENELOPE uses ADM to describe the structure of the resulting hypertext, and allows to define new page schemes to present data under the form of derived pages, which are correlated using a suitable URL invention mechanism borrowed from object-oriented databases in order to generate a complex hypertext. In essence the derived hypertext offers an alternative and integrated view over the original sites; it can involve both new pages and existing portions of the Web, in order to re-organize and re-use data.

Based on these ideas, the view definition process can be summarized as follows: *(i)* Web sites of interest are identified and relevant data are described using ADM; pages are wrapped using the EDITOR language in order to abstract their representation with respect to the physical structure; *(ii)* the ULIXES view language is used to define a set of database views over the sites of interest; these views can be analyzed, manipulated and integrated; *(iii)* finally, the resulting table-based data are restructured using PENELOPE to generate a derived hypertext to be browsed by the user, possibly involving existing portions of the Web.

We have chosen to model database views as relations; however, all the ideas of the paper can be easily adapted to work with object-oriented or object-relational structures as well.

Organization of the paper.

Section 2 provides a discussion of related work. Section 3 introduces the ARANEUS data model (ADM). The ULIXES view language is discussed in Section 4, whereas the language PENELOPE is introduced in Section 5. Due to space limitations, the presentation is mainly informal. All examples in the paper refer to the Database and Logic Programming Bibliography server at Trier [22]. In order to simplify the presentation, only a fragment of the corresponding site is discussed, and some of the features have been simplified. Moreover, all names in the paper are completely fictional. An on-line demo of a prototype of the view languages can be found at the ARANEUS project home page [1], where more examples on different Web sites are presented.

2 Related Work

The ARANEUS data model can be considered as a subset of ODMG [13], in the sense that the notion of page-scheme can be assimilated to the notion of class. However, there are some important differences, motivated by the nature of HTML documents: first, there is only one collection type in ADM, namely, lists; moreover, inheritance is not present in ADM, whereas heterogeneous union is supported; also, identifiers in ADM – that is, URLs – are visible to the user and can be queried like any other value; finally, ADM provides a *form* construct, which is specific of the Web framework.

Several query systems for unstructured data have been recently proposed. W3QL [19] allows for expressing both structure specifying queries, based on the organization of the hypertext, and content queries, mainly based on information retrieval techniques. Moreover, the language has been designed to be highly extensible, and tools for managing Web forms are presented. WebSQL [23] is another query language in the spirit of W3QL, in which the effort is towards formalization and some interesting ideas are introduced, such as the one of query locality. Other interesting ideas, like the one of restructuring, are introduced in the WebLog query language [20], whose syntax and semantics are based on logic. In all of these languages there is a very simple notion of scheme, and pages are considered within a single type, i.e., as nodes in a graph, with at most a fixed set of attributes. Moreover, the idea of derived Web structures is not investigated. ARANEUS builds on these proposals, trying to generalize them. In fact, if needed, ADM allows to see the Web as a collection of essentially unstructured pages connected using links (see [10]); however, in addition, it can also be used to describe the inner structure of pages by means of a scheme, thus providing a finer description of data organization.

Other proposals, namely Lorel [7] and UnQL [12], aim at querying heterogeneous and semi-structured information. With respect to ARANEUS, these languages adopt a *lightweight* data model to represent data, based on labeled graphs, and concentrate on the development of powerful query languages for these structures; on the contrary, ADM provides more complex structures, such as lists of tuples. Moreover, in both these proposals, there is no notion of scheme similar to the one of ARANEUS. In fact, we consider the scheme very important in order to reason about data organization and to use high-level tools for manipulating data.

In ARANEUS, unstructured HTML documents are analyzed to extract their structure; this has several points in common with the management of textual data in the OQL-doc project [6, 5]. Although OQL-doc is not a query language designed for the Web, it shares with ARANEUS the idea of giving structure to unstructured data, and then use high-level database languages to pose queries. However, in OQL-doc, this activity is based on the use of context-free grammars and *structuring schemes*. When dealing with HTML documents, grammars show important limitations. First, the structure is not always completely defined; moreover, the structure can be irregular; finally, HTML documents often contain *errors*, in the sense that they do not fully comply to HTML grammar rules; missing tags are a common example of these errors. To overcome these problems, we adopt a different approach, based on the EDITOR language [9], which, being procedural, allows for a more flexible manipulation of text than traditional parsers.

The use of path expressions [17] in the ULIXES view language inherits some ideas from all of the previous languages. In fact, in Section 4 we show how it is possible to express some WebSQL queries using ULIXES. However, while all of the previous languages allow for expressing recursive paths, here, for the sake of simplicity, we discuss only simple path expressions without recursion. This is sufficient for expressing most "real-life" navigations. However, the language can be extended with recursion in order to enhance the expressive power.

Other proposals [14, 21] aim at integrating data from the Web. The TSIMMIS system [14] extracts data from heterogeneous (possibly semi-structured) sources, and correlates them in order to generate an integrated database representation. Specific translators are written for the various sources. Information Manifold [21] focuses on databases accessible through an interface based on fill-in forms, and provides a specific support for querying, on the basis of declarative descriptions of the contents. These techniques can be used in ARANEUS in order to correlate tabular data and generate integrated views.

Several commercial database systems (see, for example, [3, 2]) now provide functionalities for the automatic generation of pages. However, they mainly allow for generating a single page at a time, containing a set of database tuples; 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, it can also generate a whole hypertext based on the database content. Finally, the issue of hypertext structure is not addressed in commercial products.

3 ADM: A Logical Data Model for Web Hypertexts

The ARANEUS Data Model (ADM) is a variant of ODMG [13], specifically tailored to the Web context. We say that it is *page oriented* in the sense that it recognizes the central role that pages play in this framework. Each Web page is considered as an object with an identifier (the URL) and a set of attributes. We introduce the notion of a *page scheme*, which resembles the notion of relation scheme in relational databases or class in object-oriented databases, to model sets of homogeneous pages. Attributes of a page may have simple or complex type. Simple type attributes are *monovalued* and correspond essentially to text, images or links to other pages. Beside monovalued attributes, Web pages often contain collections of objects, that is, multivalued attributes. We model them using *lists* of tuples. Component types in lists can be in turn multivalued, and therefore nested lists may arise. It should be noted that we have chosen lists as the only multivalued type since repeated patterns in Web pages are physically ordered.

There are some specific aspects in this framework with no counterpart in traditional data models. First, 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*, which is essentially considered 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; such lists are virtual since tuples are not stored in the page but correspond to submissions of the form. Second, usually a site includes pages that have a special role and are unique, in the sense that there are no other pages with the same structure. Typically, at least the home page of each site falls in this category. For the sake of homogeneity, we also model these pages by means of page-schemes. For example, the DB & LP Bibliography Server [22] provides a page to search for a specific author based on his/her name[1]. The page is unique since no other page in the site has the same structure. Moreover, an essential feature of the

[1] The page URL is `http://www.informatik.uni-trier.-de/~ley/db/indices/a-tree/index.html`

page is the form for searching the author database: by specifying a string corresponding to the name of an author, the list of publications of that author is returned. We see the form as a virtual list of tuples with two attributes: the first is the text entered as a string to search; the second is a link to the page generated by the corresponding search; each tuple associates a result page with a search keyword; tuples in the list cannot be directly accessed, but they must be "built up", in the sense that a keyword has to be specified in order to generate a result page.

Other constructs of the model are due to the semi-structured and heterogeneous nature of Web data; besides *optional* attributes, which allow to describe pages with missing features, ADM also provides a *heterogeneous union* type [16], which seems more appropriate to this context than traditional class inheritance [4]. Consider again the page in the DB & LP Bibliography Server [22] to search the author database. The search form has a rather involved behavior: when a string is specified, the database of author names is searched; if a single name matching the query string is found, the author's page with his/her publications is returned; on the contrary, if the query string matches several names, a different page containing the list of all matching names, along with links to the corresponding pages, is returned. Union is essential to model this behavior: in this case, we say that the form returns a link to a type which is the union of two different page-schemes, one for author pages and the other for name-index pages.

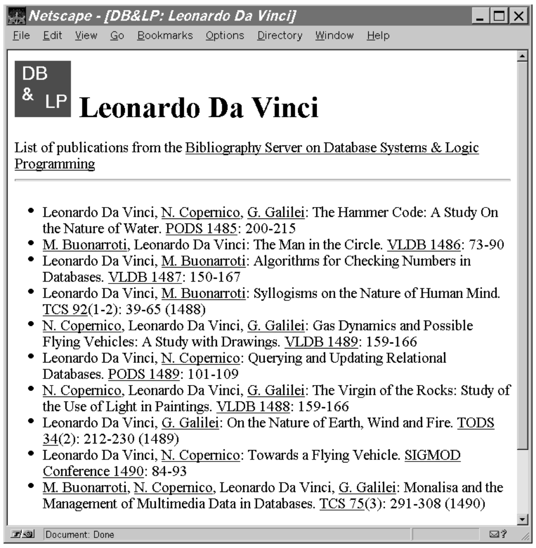

Figure 1: A page containing the publications by Leonardo Da Vinci

We define the notion of ADM *scheme* as a set of page-schemes. We can represent the scheme as a directed multigraph; nodes in the scheme graph are page-schemes; each unique page-scheme in the diagrammatic representation is denoted as a single page, whereas non-unique page-schemes are represented as "stacks" of pages; edges are used to denote links. A fragment of the DB & LP Bibliography server scheme is shown in Figure 2, which also contains an explanation of the other symbols.

Consider for example page-scheme `AuthorPage` in Figure 2; it corresponds to pages such as the one in Figure 1, referring to fictional publications of author Leonardo Da Vinci. For each author in the site there is a similar page, and all of these pages share the same structure. The corresponding page-scheme has one monovalued attribute (the `Name`) plus a multivalued attribute (the `WorkList`), containing the list of publications; this, in turn, is a set of nested tuples. Note how we choose to model information about publications: for each paper, we extract the title, the reference (conference or journal the paper was published in) and the year; moreover, we also have a list of authors; for each author, we report the name and a link to the corresponding page in the site; this link is optional since it is not present for all authors. However, we choose to have a slightly redundant representation, in the sense that we also extract a string containing the names of all authors. This has several advantages in terms of querying the site, allowing to pose conditions on all author names, and can be done since the page-scheme is only one of the possible descriptions of information in the page, reflecting the user's personal view of the site, and can be adapted on the basis of efficiency needs. Each citation also contains a link, `ToRefPage`, to the corresponding conference or journal page; since pages for conferences and journals have different structure, we use a union type to model the link.

Note that, to see actual pages in the site as instances of the page-scheme, we need to access the HTML source and apply suitable text restructuring procedures. Attribute values are extracted from the HTML source using the EDITOR language [9], a formalism for text manipulation. In the current implementation we wrap pages using Java classes; each page-scheme in the site corresponds to a specific class, with one method for each attribute; each method implements an EDITOR program that accesses the HTML source and returns a complex value for the attribute. Based on this perspective, at the instance level, a site can be seen as a graph in which links connect trees corresponding to pages. In fact, each instance of a page-scheme is a tree (because of its nested structure), and may contain links to other instances. Nodes of trees (and therefore of the overall graph) are essentially tuples; each tuple attribute may either have a simple value or be the root of a subtree; optional attributes may have a null value.

It is worth noting that this approach generalizes the models adopted in other query languages for the Web [19, 20, 23], in which pages are considered as essentially unstructured objects. In fact, in [10], we show how it is possible to model the whole Web using ADM, and, based on this scheme, we discuss how to ask very general queries, such as "*retrieve all*

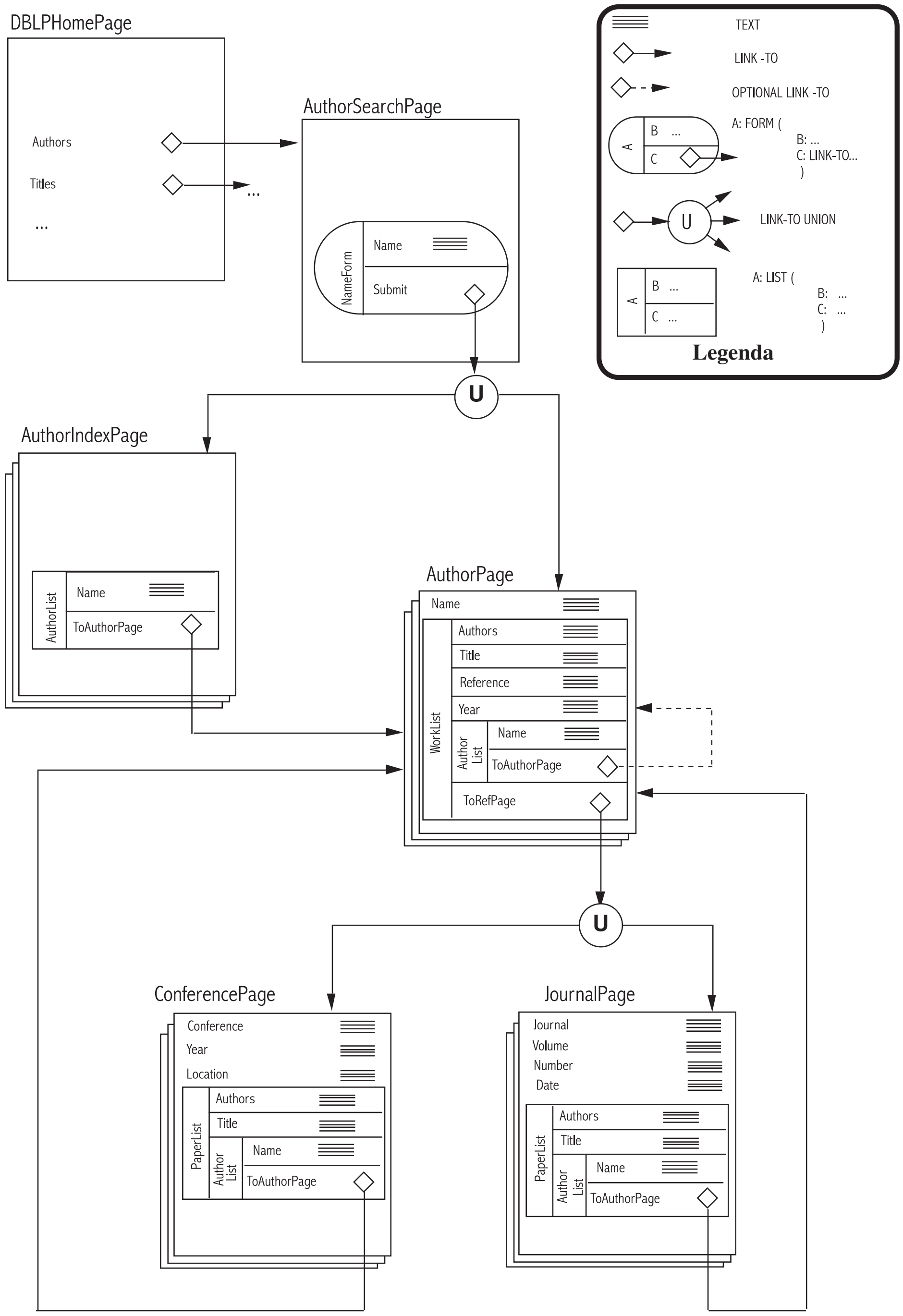

Figure 2: A portion of `DBLPScheme`, the ADM scheme for the the DB & LP Bibliography Server

documents in the Web mentioning Java" [23], using ULIXES. This shows that our approach is highly scalable: in fact, the model can be used also when there is little structure. However, in addition, it allows to model existing structures and regularities, when considered interesting. In fact, in those sites in which pages referring to similar concepts can be considered as having the same scheme, the model provides a concise and effective description of the site's content.[2]

4 ULIXES: Defining Relational Views over the Web

In this section, we present ULIXES, a language for the definition of relational views over the Web. ULIXES has been designed to be a simple and flexible language for extracting data from the Web based on an ADM scheme. The data extraction process is based on the notion of *navigation* in the site. Navigations in ULIXES are expressed using *navigational expressions*, i.e., path expressions [17] denoting paths in the site graph. In our perspective, a site offers in essence a set of navigations through pages in the site; these navigations allow to follow links between different pages, but also to explore the hierarchical structure of a page: they represent a natural means to query the page. Consider for example the scheme in Figure 2. Suppose we are interested in reaching all author pages in the site. To do this, we can start from the author search page (`AuthorSearchPage`), whose URL is known, submit an empty form, and reach the search result page; this is an index of all authors in the site, that is an instance of page-scheme `AuthorIndexPage`; for each author, the corresponding page can be reached by following the associate link. These navigations in the site can be specified using the following *navigational expression*, in which the dot operator (.) denotes navigations inside pages, and the *link operator* (→) is used to follow links:

```
AuthorSearchPage.NameForm.Submit →
    AuthorIndexPage.AuthorList.ToAuthorPage →
        AuthorPage
```

The semantics of the expression can be interpreted as all possible paths in the site obtained by starting from the unique instance of page-scheme `AuthorSearchPage`, submitting an empty form, traversing the `AuthorIndexPage` and then reaching an `AuthorPage`. Each of these navigations can be represented as a tuple of values, one value for each monovalued attribute associated to nodes in the navigation; thus, each navigational expression can be represented as a relation, in the relational model sense. Based on these ideas, we can associate a relation, i.e., a set of tuples to each navigational expression. Given a navigational expression, N, we call $\text{SEM}(N)$ the corresponding relation. We assume that attributes are suitably renamed whenever needed.

Given the relational nature of navigations, the definition of relational views over ADM schemes can be directly based on navigational expressions. We have a `DEFINE TABLE` statement to be used for this purpose, with the form:

`DEFINE TABLE`	$R(B_1, B_2, \ldots, B_n)$
`AS`	N
`IN`	S
`USING`	$A_1, A_2, \ldots, A_n$
`WHERE`	$c_1, c_2, \ldots c_k$

where: *(i)* R is a relation name and $B_1, B_2, \ldots, B_n$ are attributes; *(ii)* S is an ADM scheme; *(iii)* N is a navigational expression over S; *(iv)* $A_1, A_2, \ldots, A_n$ are attributes of $\text{SEM}(N)$; and *(v)* $c_1, c_2, \ldots, c_k$ are a set of conditions over the attribute values.

As an example, suppose we are interested in the authors, titles and references for all papers by Leonardo da Vinci in VLDB conferences. We can generate such a relation using the following statement:

```
DEFINE TABLE VLDBPapers (Authors, Title, Reference)
AS           AuthorSearchPage.NameForm.Submit →
                 AuthorPage.WorkList
IN           DBLPScheme
USING        AuthorPage.WorkList.Authors,
             AuthorPage.WorkList.Title,
             AuthorPage.WorkList.Reference
WHERE        AuthorSearchPage.NameForm.Name =
                 'Leonardo Da Vinci',
             AuthorPage.WorkList.Reference
                LIKE '%VLDB%'
```

In this expression, we are essentially giving a name, `VLDBPapers`, to a relation corresponding to the navigational expression:

```
AuthorSearchPage.NameForm.Submit →
    AuthorPage.WorkList
```

The resulting relation includes only a subset of attributes, namely those listed in the `USING` clause: the authors, title, and reference for each of the selected papers. There are several things to note with respect to this example; first, during the navigation, data are filtered using conditions in the `WHERE` clause, so that only paths ending with a paper in a VLDB conference are considered. Moreover, the `WHERE` clause is also used to fill-in the form in a completely transparent way by specifying that the name attribute of the form must be equal to 'Leonardo da Vinci'. Note also that in the navigational expression we require that the `Submit` link returns a page over scheme `AuthorPage`, that is, we are selecting one of the possibilities in the union type. The language has specific mechanisms for dealing with union types, so that, if the page returned is of the correct type, it navigates it; otherwise, it simply returns an empty result.

[2] Pages in a site can be often reconducted to a relatively small number of different types; in fact, in order to reduce design and maintenance costs, large sites tend to reduce the amount of heterogeneity among their pages, trying to gain structure [4].

5 Penelope: Generating Derived Hypertexts

The approach discussed in the previous section is interesting but could be considered as extraneous to the Web framework, where users access information by navigating hypertexts. We thus would like to extend the view paradigm in such a way that, once data have been retrieved, they are presented to the user as a hypertext. Here, we show how relational views can be transformed back into Web hypertexts, whose pages have a structure that does not appear in the existing site(s).

In order to reach this goal, we introduce Penelope, which allows the definition of new page-schemes according to which data will be organized. Let us illustrate the process by means of an example, again on [22]. Suppose we have used Ulixes to navigate the site and build the table `DaVinciPapers` containing a tuple of the form *(Authors, Title, Reference, Year, ToRefPage)* for each paper by Leonardo Da Vinci. We want to re-organize papers, dividing them on the basis of the year of publication. To do this, for each year in the table, we generate a page containing the list of papers published by the author in that year; moreover, a unique page containing the list of all years is created to provide access to year pages. We need a *restructured scheme* as described in Figure 3; the two page-schemes are called `DaVinciYearsPage` and `YearPage`. The structure of the pages can be defined using the following `DEFINE PAGE` statements, where attributes of the source table `DaVinciPapers` are enclosed in angle brackets <>.

```
DEFINE PAGE YearPage
AS   URL      URL(<Year>);
     Year:     TEXT <Year>;
     WorkList: LIST OF
                 (Authors:  TEXT   <Authors>;
                  Title:    TEXT   <Title>;
                  Reference: TEXT  <Reference>;
                  ToRefPage: LINK TO ConferencePage
                             UNION JournalPage
                               <ToRefPage>);
FROM DaVinciPapers

DEFINE PAGE DaVinciYearsPage UNIQUE
AS   URL 'result.html';
     YearList: LIST OF
                 (Year:     TEXT <Year>;
                  ToYearPage:LINK TO
                             YearPage (URL(<Year>)));
FROM DaVinciPapers
```

These statements generate the HTML code for the new pages. The first statement defines `YearPage` as a page-scheme with a monovalued attribute, the year, and a multivalued attribute corresponding to the list of papers by Leonardo published in that year. A page will be created for each different year; 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(<Year>)` specifies that the system has to generate an URL for each page over scheme `YearPage`, and that the URL must be uniquely associated with the year value.[3] The `DEFINE PAGE` statement also describes how pages must be filled-in starting from attributes in the relation. For example, the definition of attribute `Year` of type `TEXT` in page-scheme `YearPage`, specifies that its values come from attribute *Year* of relation `DaVinciPapers`.

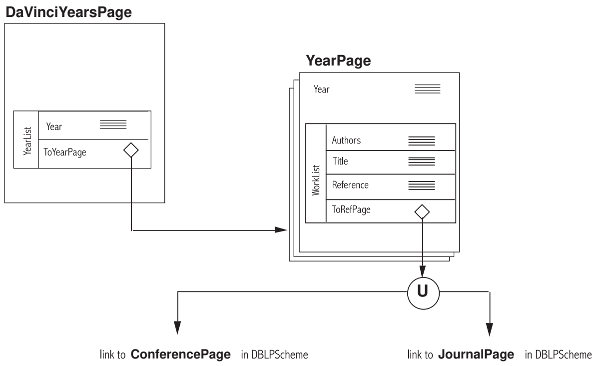

Figure 3: New page-schemes to organize papers by year

The second statement defines `DaVinciYearsPage` as a unique page-scheme with a multivalued attribute `YearList`, corresponding to the list of years; note how, in this case, a local, constant URL, `result.html`, is assigned to the corresponding instance. Since we declare the page-scheme as unique and indicate a single URL, we are assuming that a unique page will be generated by the statement. In the page, each item in the year list must be linked to the corresponding year page; to do this, we use as a value for the link the same function term used to generate URLs of years pages, i.e., `URL(<Year>)`.

Note that, for each paper in `YearPage`, we also want to access the corresponding conference or journal page. However, we do not re-create these pages in the derived hypertext, but instead we access those in the original site. To link these pages, we use URLs of existing pages, i.e., values of attribute `ToRefPage` in the table, which have been extracted using Ulixes. In essence, by using the two languages together, we generate a hypertext made of some new pages, offering a different perspective over data, plus an existing portion of the original site.

`DEFINE PAGE` statements are based on these ideas. *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(<Year>)` denotes a local URL built from values of attribute `Year`. Thus, a `DEFINE PAGE` statement has the form:

DEFINE PAGE P [UNIQUE]
AS S
FROM R

[3] This technique is somehow similar to the use of *Skolem functors* to invent new OID's in object-oriented databases [15].

where: *(i)* P is a new page-scheme name; *(ii)* R is a relation; and *(iii)* S describes the page structure, by specifying the page attributes, their type, and their correspondence with attributes of R. The `UNIQUE` keyword is optional; it is used to specify that the defined page-scheme is unique.

References

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

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

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

[4] S. Abiteboul. Querying semi-structured data. In *Sixth International Conference on Data Base Theory, (ICDT'97), Delphi (Greece), Lecture Notes in Computer Science*, 1997.

[5] 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. `http://www-db.stanford.edu`.

[6] S. Abiteboul, S. Cluet, and T. Milo. Querying and updating the file. In *International Conf. on Very Large Data Bases (VLDB'93), Dublin*, pages 73–84, 1993.

[7] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. Wiener. The Lorel query language for semistructured data. `http://www-db.stanford.edu`, 1996.

[8] P. M. G. Apers. Identifying internet-related database research. In *Second International East-West Database Workshop, Klagenfurt, Workshops in Computing*, pages 183–193. Springer-Verlag, 1994.

[9] 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`.

[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] T. Berners-Lee, R. Cailliau, A. Lautonen, H. F. Nielsen, and A. Secret. The World Wide Web. *Communications of the ACM*, 37(8):76–82, August 1994.

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

[14] 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.

[15] 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.

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

[17] M. Kifer, W. Kim, and Y. Sagiv. Querying object-oriented databases. In *ACM SIGMOD International Conf. on Management of Data*, pages 393–402, 1992.

[18] W. Kim, editor. *Modern Database Systems: the Object Model, Interoperability, and Beyond.* ACM Press and Addison Wesley, 1995.

[19] 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.

[20] 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.

[21] 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.

[22] M. Ley. Database systems and logic programming bibliography server. `http://www.informatik.uni-trier.de/~ley/db/index.html`.

[23] A. Mendelzon, G. Mihaila, and T. Milo. Querying the World Wide Web. In *First Int. Conf. on Parallel and Distributed Information Systems (PDIS'96)*, 1996. `ftp://db.toronto.edu/pub/papers/websql.ps`.

[24] 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.