A Logical Model for Metadata in Web Bases*

P. Atzeni[2], G. Mecca[1], P. Merialdo[2], G. Sindoni[3]

[1] D.I.F.A.–Università della Basilicata [2] D.I.A.–Università Roma Tre [3] D.C.I.–Rutherford Appleton Lab.

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

Abstract

In systems that provide integrated management of both structured, database–style, data and semi-structured, Web-style, data, metadata may allow for the explicit management of information about the structure and data content of pages. Unfortunately, the HTML standard doesn't offer any explicit framework for document metadata management. The eXtensible Markup Language vice versa allows for the explicit description of the structure of a document, but it lacks of database perspective. The aim of this paper is then to describe an approach to the generation of database–derived Web meta–information that is based on a logical model for Web pages and to show how this approach can be effectively used for both HTML– and XML–based systems.

1 Introduction

The Web scenario is increasingly being populated by sites where data play a major role. These sites are usually built over one or more underlying databases and they are mainly used to dynamically present a hypertext view to the user. However, the great opportunity offered by the Web–related technologies being a de facto standard, calls for an evolution of these sites towards a more sophisticated and powerful type of information systems, which overcome the vision of databases in the Web as purely data sinks or sources. Such systems should provide integrated management of both structured, database–style, data and semistructured, Web–style, data. In particular, the same declarative data querying functionality that is offered by DBMSs should be offered also for semistructured data; hypertext views over the database should be designed [8, 11, 12], described by means of a suitable model and language, and automatically generated; finally, it should be possible to create cooperative Web applications by the coordination of the above activities. We call systems with the above characteristics *Web Base Management Systems* [17].

In such a context, metadata have a fundamental importance with respect to all the above features. In fact, by allowing the system to explicitly manage meta–information about the structure and data content of pages, it would be possible to easily *(i)* enable site structured querying, *(ii)* embed into pages some knowledge about presented values, *(iii)* integrate and exchange data with other systems.

In this respect, the recent World Wide Web Consortium (W3C) recommendation for an eXtensible Mark-up Language (XML, [5]) deserves particular attention because its formal, concise design made it straightforwardly usable on the Internet and it thus gained immediate consensus among research communities and industries. XML allows for the description of the structure of a document and for the complete separation between document *structure* and *layout*. It seems then the ideal candidate as the reference document format for providing Web–based cooperative applications with the above classes of functions [9]. Unfortunately, the database perspective hasn't been taken into account during XML design process and the corresponding proposal lacks of the notion of *site scheme* and of specific features to describe *site–database mappings*, consequently, XML meta–information framework as is, cannot tell anything about the relationships between structured and semistructured data. It then needs to be effectively supported by suitable models and tools for designing mappings between databases and XML sites.

*__Contact Author__: Giuseppe Sindoni, D. I. A., via della Vasca Navale 79, 00146 – Roma, Italy. Tel. +39-6-55173229, fax: +39-6-5573030

The aim of this paper is to describe an approach to the generation of Web meta–information that is based on a logical model for Web pages and to show how this approach can be effectively used for the generation of both HTML and XML sites. The Araneus Data Model (adm) [6] will be presented as a logical model for Web site design. It has been defined in the framework of the Araneus project [1] and it is an ODMG–like model that allows to describe the structure of a Web hypertext. At the moment, metadata management in Araneus is performed by a language, which is called Penelope Definition Language [6], that allows to describe in a declarative fashion the mappings between site pages and database data, according to the adm model for the site. The adopted logical modeling approach, enables the Penelope interpreter to automatically generating the site pages (data) and to embed, into standard HTML comments, suitable meta–tags (metadata), which describe the page structure and its data content.

Finally, the relationships between adm and XML will be described and it will be shown how our approach can be properly extended to the automatic generation of XML sites. Generated sites will be so: *(i)* more easily suitable for declarative structured querying by navigational languages and advanced query systems (see for example [6, 16]); *(ii)* self-described, because the implicit knowledge contained in the logical scheme of the site can be made explicit into XML declarations and tags; and *(iii)* compliant with a recommended and widely accepted standard and then suitable for data exchange in cooperative applications.

2 The Araneus Approach to Metadata Generation

In order to describe site structure and mappings between database and sites, a specific model and a language for hypertext views are used. Their basic features are described by the following example concerning the scheme of a simple University department database, from which the department Web site is derived.

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

The reference data model for hypertext views is a subset of the Araneus Data Model (adm), and it will be described in the following, with the help of the scheme of Fig. 1, which gives a graphical intuition of how the model can be used to describe the logical structure of a site.

adm is a page–oriented model, because page is the main concept. Each hypertext page is seen as an object having an identifier (its url) and a number of attributes. The structure of a page is abstracted by its *page scheme* and each page is an instance of a page scheme. The notion of page scheme may be assimilated to the one of relation scheme, in the relational data model, or object class, in object oriented databases. In Fig. 1 for example, home pages of professors are described by the `PROF_GENERAL_PAGE` page scheme. Many examples of adm schemes can be found at the Araneus project Web site [1].

Each `PROF_GENERAL_PAGE` instance has six simple attributes (`Name`, `Area`, `Photo`, `E-mail`, `Phone` and `RoomNum`). Pages can also have complex attributes: **lists**, possibly nested at an arbitrary level, and **links** to other pages. The example shows the `RESEARCH_GROUP_PAGE` page scheme, having two list attributes (`Topic-List` and `MemberList`). In particular the elements of `MemberList` are couples, formed by a link, to either an instance of `PROF_ GENERAL_PAGE` or of `STUDENT_PAGE`, and the corresponding anchor (`Member`). Finally, the `PUBLICATION_LIST_PAGE` page scheme, has a list attribute with a second level nested list (`AuthorList`).

The structure of a page scheme can be alternatively defined by a simple definition language [6]. For example, the adm declaration for the `RESEARCH_GROUP_PAGE` page scheme is the following.

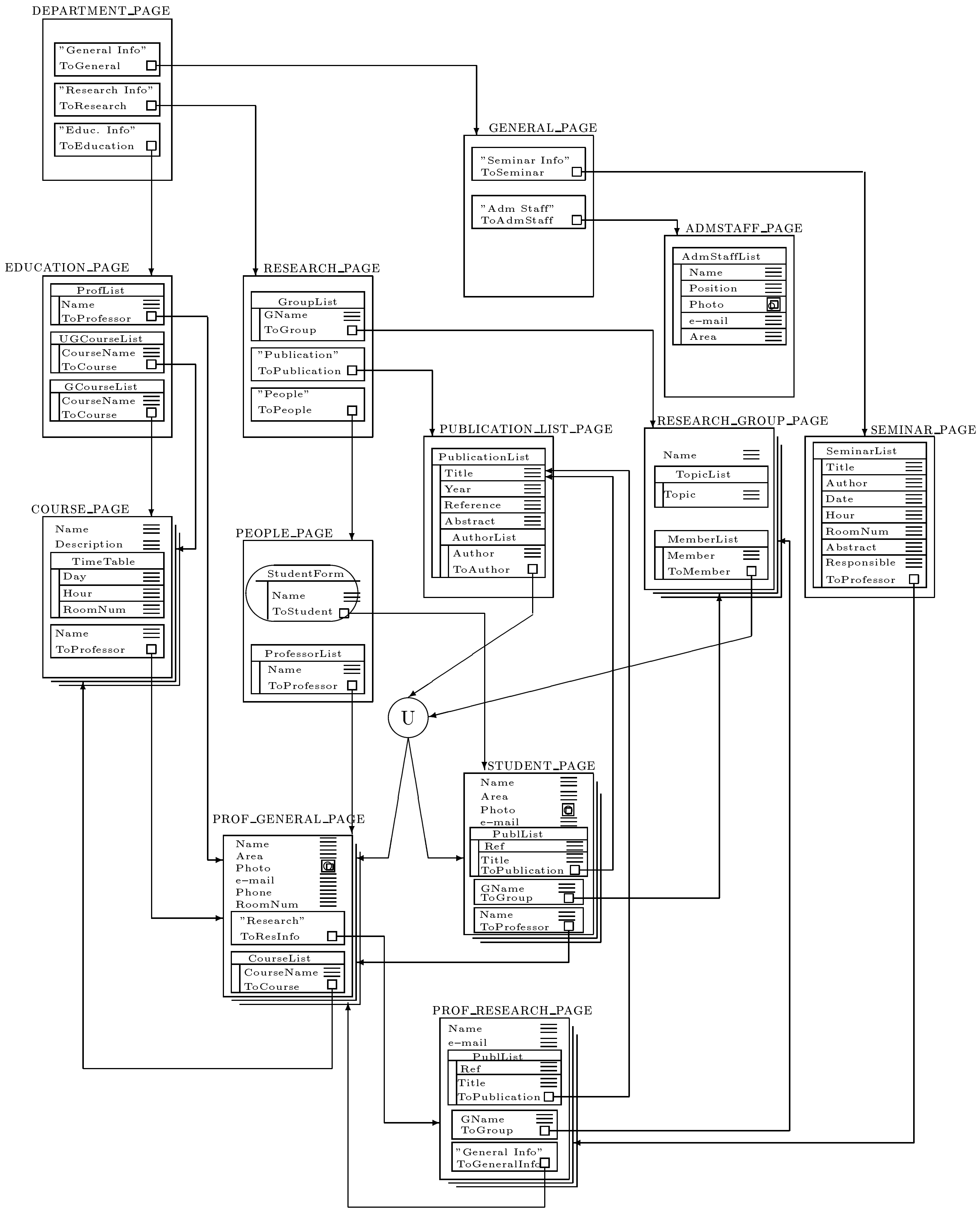

Figure 1: The Department ADM scheme

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

Note the use of a `heterogeneous UNION` type, to describe the the fact that the target of the `ToMember` link attribute can be either a professor page or a student page. The reader interested in the details of the full model, may see [6].

Hypertext views over a relational database can be defined in a declarative fashion using the PENELOPE language for Web site definition [6].

The language supports both materialized and virtual solutions for page generation: pages can be either generated from the database content and materialized on a server, or they can be dynamically delivered to the browser after a user's explicit request. The advantages of materializing database–derived sites have been discussed in previous works [17, 19]. Here it is worth to put into evidence that the ability of automatically embedding meta–information in HTML code allows pages to be queried using suitable tools, as it will be clarified in the follow.

In PENELOPE, page structure is described by `DEFINE PAGE` statements: they essentially specify how to generate pages based on database table attributes. There are some main PENELOPE features that are worth to put into evidence: *(i)* a suitable URL invention mechanism is used, that allows to correlate page instances while guaranteeing consistence to the whole hypertext; *(ii)* the automatic embedding of meta–information into the derived pages allows to keep track of page structure; *(iii)* hypertext pages may be mapped both on database relations and views.

For example, the following PENELOPE `DEFINE PAGE` statement generates the HTML code for the page instances corresponding to the `RESEARCH_GROUP_PAGE` page scheme of Fig. 1:

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

Where `RESEARCH_GROUP_PAGE_VIEW` is the associated database view, which has attributes `GroupName`, `Topic` and `MemberName`. The name of the starting database (`DEPTDB`) is specified in the `IN` clause, while the `FROM` clause indicates the database table(s) or view(s) that contains relevant data. The `AS` clause specifies the filling–out of data in pages. For each page, a different URL is created by the use of function terms. Filling–out of pages is specified by attribute definitions. For example, the values of the `Name TEXT` attribute come from attribute `GroupName` of the database view. Finally, coherently with the ADM page scheme structure, the definition of the `MemberList` attribute specifies how, for each member, a link to the corresponding page must be established. This is accomplished by using as an anchor the member name and by using as a value for the the link reference the function term `URL(<MemberName>)`.

The PENELOPE source code must of course be enriched by proper *tagging directives*, in order to achieve the desired page layout.

It is very useful to keep track of page structures in the generated HTML files in such a way to make those meta–information available to Web–based applications. This is accomplished by embedding *meta–tags* into HTML comments, thus making them completely transparent to ordinary Web browsers; they can however be used by more sophisticated applications in order to extract relevant pieces of information from the page.

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

Figure 2: A sample HTML source generated by PENELOPE with embedded meta–tags

Consider for example the HTML source shown in Fig. 2. Some hidden tags have been added, beside actual data to be displayed. The first of these tags, in the page header, `<!-- PAGE-SCHEME Research_Group_Page ... -->`, describes the structure of the page-scheme according to which the page is organized. Then, for each attribute in the page, the corresponding value is marked by suitable meta–tags in order to easily recognize and extract the value.

Once the page has been organized in this way, applications can be developed that make use of page data. For example it may be possible to *query* the site, i.e., to automatically navigate the site to extract information based on high–level queries; this is often desirable when accessing large amounts of data, in order to avoid the usual disorientation associated with browsing; ULIXES [6, 7], is a tool we have explicitly designed to this end; queries over a site can be expressed based on the corresponding ADM scheme using simple path-expressions, in order to access pages and store data in a local database; as an alternative, straightforward extensions of *W3QS* [15] or *WebSQL* [18] could be used, or even a combination of a HTTP robot and grammar parser. In this way, the resulting site is not only a bunch of HTML files, but a highly structured repository that can be either browsed or queried, thus making data access more effective. More sophisticated applications may be built on top of such query tools, in order to customize data use according to specific requirements.

The embedding mechanism for meta–tags presents however some major drawbacks. In particular:

- it does not allow for explicit separation between comments (which are supposed to be pieces of information about the HTML code itself and not about data structure) and metadata (which are pieces of information about the structure and content of the document to be used by applications);
- applications must be tailored for a specific embedding syntax;
- it essentially doubles page sizes, making site management heavy;
- in order to separate document layout and structure descriptions, specific application modules must be implemented.

For example, in order to overcome the latter, we have developed a specific tool that allows to separately manage the definition of a page scheme logical structure, which is stored in *templates*, from the definition of its presentation, which is described in *stylesheets*.

The ability of generating XML files for describing page storage and structure may solve those problems.

3 The eXtensible Markup Language

The aim of this section is to give some basic notions about the main features of XML, in order to keep the reader able to understand the concepts illustrated in the following sections. For more details, see [10, 13, 20, 5].

XML is a "meta–markup" language and it allows to define tagging systems to explicitly represent the structure of a document as a tree of nested objects. A piece of XML document may in fact look like this.

```
...
<Research_Group>
        <Name>Database Group</Name>
        <TopicList>
                <Topic>Database Theory</Topic>
                <Topic>Databases and the Web</Topic>
                ...
        </TopicList>
        ...
</Research_Group>
...
```

XML documents are composed by *markup* and *content*. In the above example, the couple of tags `<Name>` and `</Name>` are the markup for the string `Database Group`, which is content.

Elements. There are different kinds of markup that can appear in an XML document. In particular, the example presents *element* markups. Elements identify the nature of the content they surround. Each element begins with a start–tag and ends with an end–tag.

Attributes. Another important kind of markup are *attributes*. Attributes are name–value pairs that occur inside tags after the element name. For example, `<Topic identifier="DBT">` is the `Topic` element with the attribute `identifier` having the value `DBT`.

Document Type Declarations. Document type declarations syntactically define the tagged structure of a document, allowing at the same time to present meta–information about its content. Declarations are used by application parsers to *validate* documents. In fact, they define the allowed sequences and nesting of tags, attribute values and their type and defaults, the names of external files and the formats of some external (non–XML) data that may be included. The most important kinds of XML type declarations are *element* declaration and *attribute* declaration.

Element Declarations. They identify the name of elements and the nature of their content. For example, the following declaration

```
<!ELEMENT Research_Group (Name, TopicList, MemberList)>
```

identifies the element named `Research_Group`. This element must contain exactly one occurrence of the element `Name`, followed by exactly one occurrence of the element `TopicList`, followed by exactly one occurrence of the element `MemberList`. Declarations for `Name`, `TopicList` and `MemberList` must also be present for an XML processing application to check the validity of a document. For example, the definition for `TopicList` might be the following.

```
<!ELEMENT TopicList (Topic+)>
<!ELEMENT Topic (#PCDATA | TopicPage)>
```

`TopicList` must contain at least one, but maybe more occurrences of `Topic`. Topic occurrences may be either `#PCDATA`, which is a keyword that means “Parseable Character DATA” and is reserved to indicate character data, or a child element named `TopicPage`.

Attribute Declarations. They identify element attributes. For example, we may have the following attribute declaration.

```
<!ATTLIST Topic
        identifier      ID              #required
        active          (yes | no)      ‘yes’>
```

Here, the element Topic has two attributes: the first is called `identifier`, it is of type ID (a specific type that means essentially an identifying name) and the presence of the corresponding value is compulsory; the second is called `active` (to indicate for example if research in that topic is currently active or not), its value must be either ‘yes’ or not and it defaults to ‘yes’.

Validating and non–Validating Applications. The content of an XML document might be in principle processed without a type declaration. The start–end tagging mechanism is in fact enough for a parser to recognize the object tree and the nesting structure. However, there are contexts where a rationale exists for requiring a declaration. For example, most authoring environments need to read and process document type declarations in order to understand and enforce the content models of the document. With respect to our specific application environment, even if the structure of the generated document is guaranteed by ADM and PENELOPE, if we want XML documents to be parsed and queried by specific tools, we need to make the document definitions available together with the documents themselves.

The document type declaration must be the first thing in the document and identifies the root element of the document. Additional declarations may come from an external definition file [1], may be included directly in the document or both. A validating application may need to read one or both the declarations, in order to perform its validating tasks.

An XML document is valid if *(i)* it obeys the syntax of XML and *(ii)* it contains a proper document type declaration and it obeys the constraints of that declaration.

Linking. The XML linking specification (XML Linking Language, XLink [3]) is currently under development, consequently, in the follow only a survey of its basic and more consolidated features, which are however enough for the aims of this paper, will be presented.

A link describes a relationship between any locations that are addressed in it. The nature of this relationship depends on both the processing application and the semantic information supplied by the document authoring application.

XML does not have a fixed set of elements, hence the element name cannot be used by a parser to locate links. A specific attribute, named `XML-LINK` is then used to identify links. Other attributes can be used to provide additional information to the processor. There are essentially four kinds of links in XLink, but the most important for our aims are *Simple Links.*

Simple Links. They are quite similar to HTML links:

```
<MemberLink XML-LINK="SIMPLE" HREF="/ProfessorPage/johndoe.xml">
        <Member>John Doe</Member>
</MemberLink>
```

They define a link between two resources: the content of the linking element itself (the `Member` element) and another resource, which may be for example an URL or a query.

XML and Presentation. Since XML documents have no fixed tag set, the hard–coded approach of HTML browsers will not work for presentation. XML document presentation must then be based on stylesheets. There is also an ongoing effort by the W3C in this regard, the proposal is called *eXtensible Style Language* (XSL [4]) and it is likely to be focused on the definition of a standard stylesheet language. Other stlylesheet languages, like Cascading Style Sheets (CSS [2]) are likely to be supported as well.

[1]The external definition is called Document Type Definition, DTD, in the spirit of SGML [14]

XML and Web Site Design. It may be argued that XML could be directly used as a logical model for Web site design. There are two main drawbacks in that respect.

1. XML lacks of the explicit notion of site scheme, which is particularly important in the Web site design process [8].
2. In XML it is not possible to explicitly represent some class of constraints. For example, in the current version it is not possible to bind the target object type of a link object to a given document type. The definition of link semantics is somehow supported by the attribute declaration mechanism, but, in the end, it is left to the authoring and processing applications.

To clarify the latter, let us have the following XML type declaration.

```
<?XML version="1.0" rmd="internal"?>
<!DOCTYPE ProfGeneralPage [
        <!ELEMENT Name (#PCDATA)>
        ...]>
```

The first line is the XML markup declaration. The markup `rmd="internal"` states that only the internal declarations needs to be processed in order to validate the document. The second line is the internal declaration of the root element of a document. But, as the scope of the name `ProfGeneralPage` is limited to the document itself, the corresponding type declaration is unknown to other documents. They cannot bind the target of link attributes in the document type declarations to the `ProfGeneralPage` type. A way out may be to spot an external DTD file by means of a specific link attribute and to instruct the processing application to parse the file in order to bind the link target to the declared type. In the next section this will be clarified with an example.

As a consequence of what has been illustrated so far, we think that the use of a more structured model like ADM for site scheme description and of a meta–markup language like XML for page and metadata storing, together with the PENELOPE language for describing database–site mapping, will enable the design, realization and maintenance of Web Base Management Systems that rely on standard and widespread technologies and that are suitable to be easily integrated into cooperative applications.

In the follow, the relationships between ADM and XML declarations will be illustrated, in order to show how it is possible to generate XML sites with the ARANEUS tools.

4 Extending the ARANEUS Approach to the Management of XML Sites

The ARANEUS Web Base Management System allows to generate HTML views over relational databases, as illustrated in the previous sections, and to query existing sites to produce *relational views over an HTML hypertext* [6]. However, as has been pointed out in Sec. 2, there are some drawbacks connected with the use of HTML as the markup syntax for the generated pages. In particular, it is not possible to separate page structure from its layout directives and the only way to export metadata about document contents is by embedding them into comments. By using XML as the syntax for document storage structure definition, a more efficient and standardized management of metadata can be achieved. Hence, in order to extend both ARANEUS site generation and querying to XML documents it is first necessary to analyze the relationships between ADM and the XML document type declaration syntax.

Document type declarations in XML can be internal or external. However, as it has been illustrated in Sec. 3, internal declaration scope spans just the document in which they are declared. So, in order to correctly bind link attributes to their target type in such a way to represent link type consistency, we must take into account external declarations (i.e. DTD files) only. In particular, we must devise a standard framework to express the fact that a given link expresses a relationship between classes of documents. While XML syntax allow to define explicitly only relationships between document instances. Moreover, having a single external declaration for each page scheme may allow applications to access only one file for all the document instances

```
<!ELEMENT ResearchGroupPage     (Name, TopicList, MemberList)>
<!ELEMENT Name                  (#PCDATA)>
<!ELEMENT TopicList             (Topic)+>
<!ELEMENT Topic                 (#PCDATA)>
<!ELEMENT MemberList            (MemberLink)+>
<!ELEMENT MemberLink            (ToProf | ToStud)>
<!ELEMENT ToProf                (Member)>
<!ELEMENT ToStud                (Member)>
<!ELEMENT Member                (#PCDATA)>

<!ATTLIST ToProf
                XML-LINK        CDATA   #FIXED          "SIMPLE"
                TRG-TYPE        CDATA   #FIXED          "/DTD/ProfPage.dtd"
                HREF            CDATA   #REQUIRED       >
<!ATTLIST ToStud
                XML-LINK        CDATA   #FIXED          "SIMPLE"
                TRG-TYPE        CDATA   #FIXED          "/DTD/StudPage.dtd"
                HREF            CDATA   #REQUIRED       >
```

Figure 3: The DTD file corresponding to the RESEARCH_GROUP_PAGE page scheme

of a given page scheme. Take for example the RESEARCH_GROUP_PAGE page scheme definition of Sec. 2: it may be mapped on the XML DTD of Fig. 3.

Here, the mechanism is worth to be put into evidence to map the ToMember link attribute. There are two problems to overcome: first, the fact that the link target cannot be explicitly bound to a type declaration, because type names can either be embedded into documents or into DTD files; second, the fact that the ToMember target may be either a ProfPage page scheme or a StudentPage page scheme (it is in fact an ADM heterogeneous union type [6]). The problems are tackled by mapping the ADM link attribute into an XML element (MemberLink), having one element child, which is a choice between either ToProf or ToStud, which are the link elements. The TRG-TYPE attribute of the link elements indicates the DTD file containing the type declarations for ProfPage and StudPage respectively. A parsing application will then access and parse the DTD in order to extract the structure of the target document.

The general mapping rules between ADM types and XML declarations follow. The rules can be used by the Penelope interpreter to generate XML declarations corresponding to ADM page schemes, in such a way to produce a consistent set of type declarations for the generated XML documents.

An ADM monovalued TEXT attribute

```
attName :  TEXT;
```

is mapped to the XML element

```
<!ELEMENT attName   (#PCDATA)>.
```

An ADM monovalued IMAGE attribute

```
attName :  IMAGE;
```

is mapped to the XML element

```
<!ELEMENT attName   (#PCDATA)>
<!ATTLIST attName
                XML-LINK        CDATA   #FIXED          "SIMPLE"
                TYPE            CDATA   #FIXED          "IMAGE"
                HREF            CDATA   #REQUIRED       >.
```

Again, it is here worth to put into evidence the use of the `TYPE` attribute as a stratagem to indicate that the element is an `IMAGE` type.

The ADM monovalued `LINK TO` attribute

```
attName :  LINK TO pageScheme;
```

is mapped to the XML element

```
<!ELEMENT attName   (#PCDATA)>
<!ATTLIST attName
                XML-LINK        CDATA   #FIXED          "SIMPLE"
                TRG-TYPE        CDATA   #FIXED          "pageScheme.dtd"
                HREF            CDATA   #REQUIRED       >.
```

where `pageScheme.dtd` is the file containing the declarations for `pageScheme` page scheme. A processing application must then be specifically instructed that, in order to validate the target documents, the DTD file must be parsed.

The ADM monovalued `LINK TO` attribute

attName : LINK TO P_1 UNION P_2 UNION $\ldots P_n$;

where each P_i is a page scheme name, is mapped to the set of XML elements

```
<!ELEMENT attName               (P1 | P2 | ... | Pn)>
<!ELEMENT P1                    anchor>
<!ELEMENT P2                    anchor>
...
<!ELEMENT Pn                    anchor>

<!ATTLIST P1
                XML-LINK        CDATA   #FIXED          "SIMPLE"
                TRG-TYPE        CDATA   #FIXED          "P1.dtd"
                HREF            CDATA   #REQUIRED       >
<!ATTLIST P2
                XML-LINK        CDATA   #FIXED          "SIMPLE"
                TRG-TYPE        CDATA   #FIXED          "P2.dtd"
                HREF            CDATA   #REQUIRED       >
...
<!ATTLIST Pn
                XML-LINK        CDATA   #FIXED          "SIMPLE"
                TRG-TYPE        CDATA   #FIXED          "Pn.dtd"
                HREF            CDATA   #REQUIRED       >.
```

Where `anchor` is the link element content.

The ADM multivalued `LIST OF` attribute

attName : LIST OF ($A_1 : T_1$, $A_2 : T_2$, $\ldots A_n : T_n$);

where each A_i is an attribute name and each T_i is an attribute type, is mapped to the XML element

```
<!ELEMENT attName (A1, A2, ... An)+>
```

and the mapping of each A_i is recursively defined.

The above mapping rules will allow the PENELOPE interpreter to produce XML DTDs for the derived XML site documents. The XML document for the `Database group` page should then look like the one in Fig. 4.

```
<?XML version="1.0" rmd="all"?>
<!DOCTYPE ResearchGroupPage SYSTEM "research_group_p.dtd">

<Name>Database Group</Name>
<TopicList>
        <Topic>Database Theory</Topic>
        <Topic>Databases and the Web</Topic>
      ...
</TopicList>
<MemberList>
        <MemberLink>
                <ToProf  XML-LINK="SIMPLE" HREF="/ProfessorPage/johndoe.xml">
                        <Member>John Doe</Member>
                </ToProf>
        </MemberLink>
            ...
        <MemberLink>
                <ToStud  XML-LINK="SIMPLE" HREF="/StudentPage/franksmith.xml">
                        <Member>Frank Smith</Member>
                </ToStud>
        </MemberLink>
        ...
</MemberList>
```

Figure 4: A sample XML source, whose DTD is depicted in Fig. 3

The value `"all"` for the `rmd` directive means that both the internal (if present) and external declarations must be read in order to validate the document. The `SYSTEM "research_group_p.dtd"` directive indicates to applications the external DTD file to be read. In our case it is the only document type declaration available for each class of pages.

References

[1] Araneus Home Page. http://poincare.dia.uniroma3.it:8080/Araneus.

[2] Cascading Style Sheets, level 2 CSS2 Specification. http://www.w3.org/TR/REC-CSS2/.

[3] The XML Linking Language (XLink). http://www.w3.org/TR/WD-xlink.

[4] A Proposal for XSL. http://www.w3.org/TR/NOTE-XSL.html.

[5] Extensible Markup Language (XML). http://www.w3.org/TR/PR-xml.html.

[6] P. Atzeni, G. Mecca and P. Merialdo. To Weave the Web. In *International Conf. on Very Large Data Bases (VLDB'97), Athens, Greece*, pages 206–215, 1997.

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

[8] P. Atzeni, G. Mecca, P. Merialdo. Design and Maintenance of Data–Intensive Web Sites. In *International Conf. on Extending Datbase Technology (EDBT'98), Valencia, Spain*, 1998.

[9] J. Bosak. XML, Java, and the Future of the Web. *World Wide Web Journal*, 2(4):219–227, Fall 1997 1997.

[10] T.Bray Annotated XML 1.0. Available at http://xml.com/axml/axml.html.

[11] 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.

[12] P. Fraternali, P. Paolini. A Conceptual Model and a Tool Environment for Developing More Scalable, Dynamic, and Customizable Web Applications. In *International Conf. on Extending Datbase Technology (EDBT'98), Valencia, Spain*, 1998.

[13] L.M.Garshol Introduction to XML. Available at http://www.stud.ifi.uio.no/~ larsga/download/xml/-xml_eng.html.

[14] E. van Herwijnen. Practical SGML. Kluwer Academic Publishers, 1990.

[15] 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.

[16] O. Liechti, M.J. Sifer, T. Ichikawa. Structured graph format: XML metadata for describing Web site structure. In *Proceedings of the 7th International World Wide Web Conference (WWW7), Brisbane, Australia*, 1998.

[17] G. Mecca, P. Atzeni, A. Masci, P. Merialdo, and G. Sindoni. The Araneus Web–Base Management System. In *International Conf. on Extending Database Technology (EDBT'98), Valencia, Spain*, 1998. Exhibits Program.

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

[19] G. Sindoni. Incremental Maintenance of Hypertext Views. In *International Workshop on the Web and Databases (WebDB'98 In conjunction with EDBT 1998), Valencia, Spain*, Available at http://-poincare.dia.uniroma3.it:8080/webdb98/papers/.

[20] N. Walsh. A Guide to XML. *World Wide Web Journal*, 2(4):97–107, Fall 1997 1997.