The ARANEUS Guide to Web-Site Development

G. Mecca[1], P. Merialdo[1,2], P. Atzeni[2], and V. Crescenzi[2]

[1] D.I.F.A. – Università della Basilicata
[2] D.I.A. – Università di Roma Tre
{mecca,merialdo,atzeni,crescenz}@dia.uniroma3.it

Abstract. Web sites are rapidly becoming a world-wide standard platform for information-system development. The paper reports on the recent advancements in our research on Web-site development conducted in the framework of the ARANEUS project. We present models, tools, methodologies, techniques for Web design and development, as well as ideas coming from a number of concrete experiences in developing data-intensive Web sites using the system. We also discuss research directions we are following to define a unified framework for data and application management on the Web.

1 Introduction

Web-site development has recently imposed itself as a new and challenging database problem. This has justified a number of research proposals coming from the database area (e.g., [5, 9, 4, 16, 15]) for data management in Web sites; other relevant works in the field have investigated the extension of design methodologies to these sites, and their interaction with development tools [6, 10]. Indeed, the notion of a Web site has recently evolved from a small, home-made collection of HTML pages into a number of different forms, including rather complex and sophisticated information system. Given the large number and diversity of Web sites, we find useful to classify them in categories, according to their complexity in terms of data and applications (i.e., services), as shown in Figure 1.

1. we call *Web-presence Sites* those with low complexity both in terms of data and applications; these sites usually contain a small number of pages (in the order of the dozens), and mainly serve for marketing purposes; we believe that a vast portion of Web sites do fall in this category; however, given the relatively small size, these are usually made by hand, possibly with the help of HTML editors or simple site-manager software;
2. the *service-oriented sites* are mainly dedicated to some specific service. There are several examples of these sites: one example are search engines; another typical example are free email services. In both cases, although the site may have a large back-end database, the structure of the data and of the hypertext is quite simple (typically one single class of objects), and the complexity is rather in the underlying applications that guarantee the service;

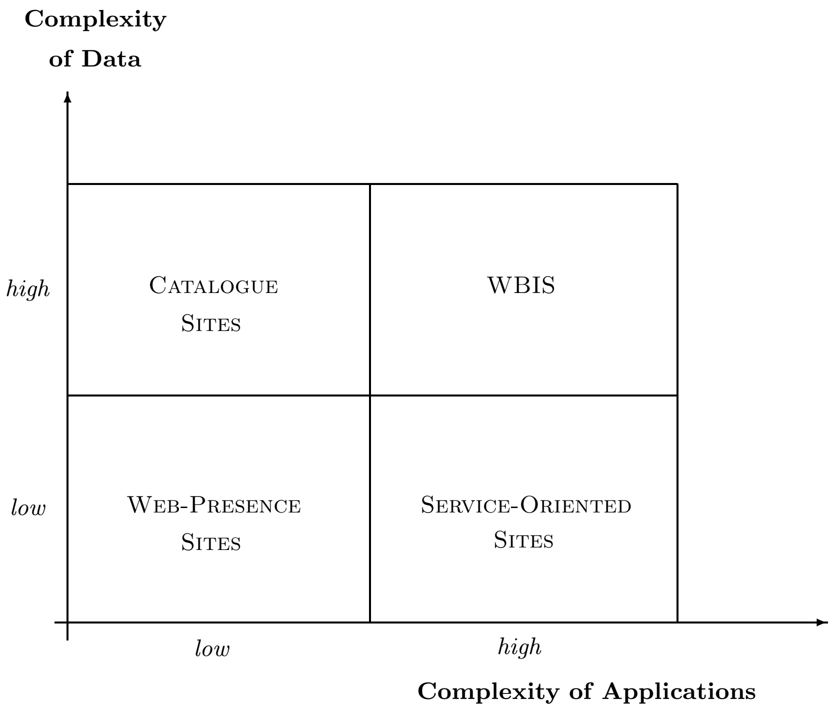

Fig. 1. Classification of Web Sites

3. *catalogue* (or *data-intensive*) *sites* publish lots of data, and typically have a complex hypertext structure, but offer little or no services. Academic sites – with data about people, courses, research – are one example. In all these sites, the focus is mainly on effectively organizing the data in hypertext form, and on the maintenance of both the underlying data and of the hypertext;
4. finally, the most general class of sites are what we call *Web-Based Information Systems*, i.e., real information systems on the Web that offer access to complex data and at the same time also provide sophisticated interactive services. Large electronic-commerce sites obviously fall in this category, as well as company information systems based on intranet platforms.

The classification is of course a bit crude: many sites may fall in between some of the categories above; yet it serves the goal of the discussion. Of these four classes, Web-presence sites can be created and maintained by hand, and usually don't need the development of ad-hoc techniques. Service-oriented sites are to be considered too application-specific to allow for a general treatment. We therefore will concentrate mainly on catalogue sites and WBIS. If developing and administering a catalogue site can in some way be considered as a typical database problem, on the other end it is still unclear what should be the foundation for developing real information systems on the Web, since a full-fledged proposal, encompassing all aspects of Web-based information-system development – namely, data–management techniques, application development and the associated methodologies – is still missing.

In this paper, we report on the recent advancements of our research on designing and developing Web applications carried on in the framework of the Araneus project [1]. In particular, we concentrate on two issues we consider critical in this context:

- the impact of *a model-based approach* on the site design and development process (in Section 2);
- the need to *evolve from static catalogue sites to full-fledged dynamic WBIS* (in Section 3).

Most of the ideas incorporated in the system stem from an intense development activity we have conducted on large real-life Web sites in different domains, ranging from University sites, to civil engineering, to non-profit organizations, for a total of several thousands Web pages. Such an experience contributed to refine our approach, on the one side by highlighting the real challenges that a system has to face in this field, and on the other side by somehow forcing us to pursue the rapid development of user needs and market technologies.

The Web-site development software discussed in this paper is part of a larger system, the Araneus *Web-Base Management System* [13], which in addition incorporates tools to query and integrate both structured and semistructured data; we will not report on this here, but simply want to emphasize that coupling data extraction with hypertext generation allows to develop a number of interesting applications, in which pieces of information are extracted at some sources, re-organized to create new sites, and these sites are not only browsed, but also possibly queried back by other applications. We refer the reader to [1] for references on other aspects of the system.

A distribution package containing part of the software described in this paper is available for download on the Araneus Project Web site [1].

2 Model-Based Site Development in Araneus

Although the market is nowadays flooded by tools for Web-site development, ranging from simple HTML editors to very complex DBMS enhancements, these are mostly rather low-level tools, either oriented to pure-HTML development or to procedural and SQL programming. On the contrary, our approach heavily relies on the adoption of high-level models, both at the conceptual and logical level, for designing and developing sites. Other proposals that share with Araneus this kind of model-based approach are AutoWeb [10], in which conceptual models for Web-site development are inherited from earlier methodologies for hypermedia design [11], and, to some extent, Oracle Designer [3].

In our approach, all aspects of the Web-site design process, namely managing data, hypertext and presentation, are based on the adoption of suitable models. More specifically:

– we adopt relational technology as a back-end for the site; as a consequence, data to be published in the site is described at the conceptual level using the well-established Entity-Relationship Model, and at the logical level using relational tables. The adoption of relational database — although giving less flexibility than, for example, using a full-fledged object-oriented database — has the important advantage of leveraging a wide-spread technology and drastically cutting the site development costs (in this way, an organization willing to develop a Web site doesn't usually need to buy a new DBMS, and can use its own as a back-end);
– we use a formal data model, called ADM, for hypertext description; ADM is essentially an object-relational data model, with untyped links and union types; each page is seen in the model as a URL-identified object with attributes, i.e., an instance of a type (the *page-scheme*);
– finally, based on ADM, we also have a formal model for describing the graphical layout of data items in a page; this is based on the notion of TELEMACHUS *styles* for attributes, pages and sites.

In the following sections we first describe in more detail the use of ADM and TELEMACHUS styles for modeling hypertext and presentation, and then, based on them, we introduce HOMER, a case tool conceived to help users in the site design and implementation phase.

For now we note that, although developed independently, both ADM and TELEMACHUS have somehow been validated by the recent ascent of XML. In fact, an interesting feature of ADM is that it represents a nice abstraction of XML-DTDs modeling primitives, thus providing a natural basis for describing XML data sources. Similarly, TELEMACHUS styles nicely fit into the XSL paradigm. As an outcome, our system allows for large flexibility in choosing the actual implementation for the site; it can easily generate both plain HTML or XML sites with XSL style-sheets.

2.1 ADM: A Logical Model for Hypertexts

We use the ARANEUS Data Model (ADM) [5] to give an intensional description of a Web site, abstracting the logical features of Web pages. In ADM each page is seen as a complex object, with an identifier, the URL, and a set of attributes. Pages sharing the same structure are grouped in *page-schemes*; a set of page-schemes corresponds to a site scheme. Attributes have a type, which can be either simple, i.e. mono-valued, or multi-valued. Simple types are `TEXT`, `IMAGE`, and `LINK`. Complex attributes are based on a limited number of primitives, as follows: (*i*) *structures*, i.e. typed tuples; (*ii*) *union types*, i.e., disjunctions of attributes; (*iv*) *lists*, i.e., ordered collections of tuples (possibly nested); (*v*) *forms*; forms are seen in the model as "virtual" lists of tuples, with a number of attributes (the form fields) of different types (text-areas, selections, radios, checkboxes etc.), and an associated action, i.e., a link to some result page; the list is virtual in the sense that values for the form attributes are not physically stored in the page, but rather have to be specified by users before submitting the form.

Filling-out form fields and executing the form action is therefore conceptually similar to selecting one tuple of values in a list of links.

There are two points we want to emphasize here. First, the use of the ADM data model plays a cardinal role in our approach; in fact, it allows to give a compact, intensional description of a site structure at an abstract level, and provides a basis for reasoning about the effectiveness of the chosen hypertext organization; also, as it will be clear in the next Sections, the site scheme is essential in all phases of the site design and implementation (all tools are based on that).

Second, it is worth noting that ADM is somehow at the crossroads of traditional database models and XML. In fact, the fundamental modeling primitives of the model have a natural counterpart in the ones that are typically offered by object-database systems, the main differences being the absence of hierarchies and inheritance, and the presence of union types. Thus, ADM modeling primitives might somehow be considered as a subset of ODMG and SQL3 data models, enriched with union types. At the same time, ADM can be considered as a logical abstraction of XML. If, in fact, XML should rather be considered as a data format than as a data model, yet its modeling primitives do correspond to the ones present in ADM: structures, possibly nested lists, disjunction, links. In this respect, an XML DTD can be seen as a type declaration for a class of documents, which has a natural counterpart in ADM page-schemes.

2.2 Modeling Presentation with TELEMACHUS

One of the most difficult and underestimated tasks in developing a Web site consists in handling the graphical layout of pages. Nevertheless, people willing to create their sites are often more worried about having an appealing presentation than about data management issues; this is not surprising, since Web sites are becoming a prominent commercial vehicle, and therefore need to attract customers. This makes design and implementation of presentation a large part of the site life-cycle.

Experience tells that there are at least three fundamental requirements in this field: (*i*) first, it is very useful to have rapid prototyping tools, to be used to produce some approximate layout for all pages in a site; this allows to concentrate on the other aspects of site design with little initial effort on the layout; (*ii*) then, at a subsequent step, one should have flexible tools to refine presentation details and obtain an appealing final result. Finally, (*iii*) presentation is hardly developed by coding; it is much more convenient to work on *example HTML pages*, that can be displayed using a standard browser to get an immediate feedback, and then let the system derive the necessary code from examples. To meet these requirements, also presentation, like data and hypertext, needs to be handled at a higher level of abstraction with respect to pure HTML code.

These ideas have inspired the development of a framework for handling presentation in Web sites, made of two components: a logical model for describing the graphical layout of pages, based on the notion of *styles*, and an associated tool, called TELEMACHUS, for presentation design and development, described in the following.

A fundamental notion in our approach is the one of *attribute style*, which specifies how values of a given attribute must be formatted in a page. To be able to produce sophisticated formatting, an attribute style is made of two arbitrary pieces of HTML code, called *prefix format string* and *suffix format string*, between which the attribute values will be enclosed when generating pages. To give an example, consider the name of a professor in her/his personal page. To have a simple, boldface style and color red for names in pages, we may specify the following attribute style:

```
NAME: [<FONT COLOR="red"><B>] [</B></FONT>]
```

In this way, for each name – say "*John Smith*" – a piece of HTML code of the form: `<FONT COLOR="red"><B>John Smith </B></FONT>` will be produced in the page. If, however, we want a more elaborate formatting, in which the name is written in red, "Arial" font, bold face, and preceded by an small image, we may use the following style (the HTML table is needed to correctly align image and text):

```
NAME: [<TABLE BORDER="0" ROWS="1" COLS="2">
       <TR><TD WIDTH="30">
       <IMG SRC="image.gif" WIDTH="30"></TD>
       <TD><FONT FACE="Arial" COLOR="red"><B>]
      [</B></FONT></TD></TR></TABLE>]
```

It can be seen how such a simple mechanism is in fact very flexible. A *page-style* specifies all format directives for a given page-scheme; it contains a set of attribute styles, one for each attribute in the ADM scheme of the page, plus a *header section* and a *footer section*. Header and footer specify graphical features to be associated with the page itself, rather than with a specific attribute, such as, for example, page background and banners. In the same way as attribute styles, also header and footer consist of arbitrary pieces of HTML code.

The styling mechanism provided by TELEMACHUS offers a further feature. In order to guarantee a good compromise between rapid prototyping and accuracy in the final product, beside attribute styles and page styles, we also have a notion of *site-styles*. Site-styles are used at the beginning of the presentation design phase, in order to quickly produce a first version of page-styles based on formatting choices that will be common to the whole site. In fact, usually pages in a site are organized according to some common lines – i.e., background color, font face, font color, link format etc. A site-style is essentially a *generic* page-style, in the sense that it specifies one header and one footer common to all page-schemes in the site, plus a number of formats, each common to all attributes of a given ADM type – text, image, link, list etc. – in the site. TELEMACHUS uses site styles as starting directives in order to automatically generate a first version of page-styles for the different page-schemes. Then, page-styles can be further customized, by changing header, footer, and attribute styles, in order to vary the layout from one page to the other.

We are now ready to discuss how TELEMACHUS works. In essence, the presentation design phase goes from rather general and undistinguished formatting (as specified by site-styles) to very particular formatting (obtained by customizing attribute-styles in page-styles). However, as already discussed above, in this process it would be inconvenient to require the designer to write style code as the one shown in examples above, but rather work with sample HTML pages, to be able to check the chosen layout without the need of generating the actual site.

TELEMACHUS has been conceived to support this process. It makes styles completely transparent to the designer: it allows to write sample HTML pages, called *templates*, from which page styles are automatically produced. A page-template is a prototypical HTML page; it does not contain actual data, but place-holders, one for each attribute in the page-scheme. For example, a template for professors' personal pages above will not contain actual names, but rather strings corresponding to attribute names, of the form `$NAME`. Beside this detail, a template is a fully standard HTML page, which can be edited using the designer's preferred editor (the only limitation being that place-holders cannot be changed) to refine the presentation, and browsed using any HTML browser to have a preview of what the corresponding pages in the site will look like. This makes templates a much more convenient work tool than a style.

Based on these ideas, the presentation design process is sketched in Figure 2. In essence, designers only work on templates, and TELEMACHUS takes care of generating the corresponding styles. If the target mark-up language is XML, TELEMACHUS also generates an XSL style-sheet for each page-scheme, on the basis of the page-style: for each attribute (i.e., XML element), an XSL rule is generated, that specifies how to embed the value of the element within the pair of formatting strings in the style.

2.3 HOMER: A Case Tool for Web Sites

It can be seen from the previous sections that designing and implementing a site is a rather complex task, that involves several aspects and requires to deal with data under different perspectives, mapping the one onto the others. For large and complex Web sites, the complexity of the design and maintenance process can be reduced only through 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.

We have developed a thorough methodological framework for designing data-intensive Web sites, the ARANEUS methodology [6]. In essence, by adopting the methodology, designers start from a conceptual description of the site domain (an Entity-Relationship scheme) and through a set of precise steps progressively moves to database logical design (this produces the database relational scheme), then hypertext design (this produces the site ADM scheme), and finally presentation design (producing page-styles).

To simplify this design process, as well as to automate the implementation phase based on the site design artifacts, we have developed HOMER, a case tool

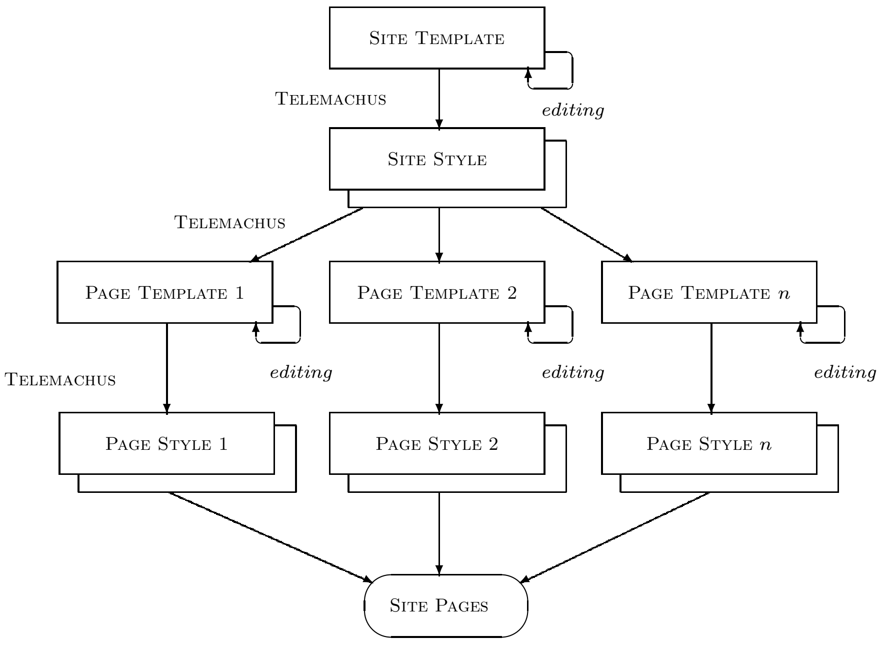

Fig. 2. Presentation Design using TELEMACHUS

conceived to support the designer through the successive design steps. This is a natural complement of our approach, in which the site design evolves through different levels and different descriptions, each based on a formal model.

Briefly, HOMER works as follows.

- Through the use of HOMER's graphical interface, the user specifies a starting conceptual scheme (Entity-Relationship scheme) of the domain of interest, to serve as a basis for both designing the database and the site hypertext.
- The ER scheme is automatically translated by HOMER into a logical (relational) database scheme.
- Then, based on the ER scheme, the designer progressively specifies how to shape the hypertext structure. This is done by graphically mapping (with the help of HOMER's user interface) entities and relationships into page-schemes and links.
- Once the ADM description for the site has been generated, HOMER first helps in choosing the appropriate styles for attributes, and then, based on the specified transformations, it automatically generates the code needed during the page-generation phase to map the resulting hypertext onto the underlying database.

Since all data models involved in the process (ER, relational, ADM), can be described as subsets of a nested-relational model (with object-identifiers and union types), at the hearth of HOMER stands a module that collects transformations as graphically specified by the site designer, and composes them into nested-relational queries (with URL inventions) over the original database. These queries will be run to construct actual pages in the site.

We have developed a tool, called PENELOPE [5], which is essentially an implementation of a nested-relational algebra with URL inventions. In the current version, HOMER produces as output a bunch of queries which are run by PENELOPE against the database to extract data items, merge them with TELEMACHUS styles, and produce pages (either HTML or XML). However, the output produced by HOMER in terms of mapping to the database is essentially independent from PENELOPE; it might as well produce HTML skeletons with SQL calls for any of the major DBMS page-generation tools, as well as for other HTML-oriented database gateways, like Java Server Pages [2].

3 Towards Web-Based Information Systems

It can be seen how the tools described in sections above represent a flexible platform for developing data-intensive sites. Still, they provide little support for adding services and application to the site. Our goal is therefore to extend the framework to handle application as a further level in the design and implementation phase. More specifically, we would like on the one side to extend the framework described above with a further level, the one of *applications*, and have high-level models and tools for this new level as well; on the other side, we would like this new level to be as integrated with the previous ones as possible.

In this respect, *workflows* [12] represent a promising direction. Born to automate *business processes* – i.e., coordinated procedures and activities aimed at realizing some business objective – workflow management system are a natural solution to deliver services on the Web [14]. Moreover, in adopting workflows we can leverage on a rather consolidated platform in terms of design and modeling [8, 7]. Our approach is to extend the framework developed in the previous sections with a workflow conceptual model and a workflow management system, called NEPTUNE, conceived to cooperate with other system tools. The model we have adopted is the one of [8, 7]. In this framework, the development of complex information systems is based on the following simple ideas.

A site is made of several intermixed portions: (*i*) a catalogue portion, of *data-access* pages, used to access and browse the site underlying database; (*ii*) one or more *workflow-execution* portions, giving access to one or more services through the execution of a workflow. To give an example, consider a conference site; most probably the site will have a part publishing data about accepted papers, program, organization, and a (private) part to handle the review process; the latter is naturally implemented as a workflow. A similar argument also holds for most electronic commerce sites. The two different portions are seamlessly combined in the site, in the sense that users may want to browse some data while running the workflow, or starting a workflow after browsing the site.

All the logics of the workflow is handled by NEPTUNE, which generates Java code to coordinate the various activities, assign tasks to actors, and authenticate accesses to the workflow, if necessary; the site is used as an *interface* to the workflow, i.e., all the interaction between actors and workflow management system happens through pages in the site; these pages are generated via a client-server interaction between NEPTUNE on the client side and PENELOPE plus TELEMACHUS on the server side, and contain suitable forms to collect user-inputs and execute tasks. Communication between the site and the workflow management system is based on the database, which is used to store both the workflow state and user inputs.

We want to stress the fact that, as well as the data-access part, also the workflow-execution part of the site needs an hypertext and presentation design phase, and is fully integrated with the data-access part, to which it can be linked. The introduction of the workflow therefore changes the overall design process. Although the implementation of NEPTUNE is still under development, our first experiences with the prototype of NEPTUNE have shown the benefits of this approach. The site is in fact a natural platform for implementing the workflow interface, whereas the design and development of a workflow nicely fits inside the design and implementation framework presented above.

References

1. The ARANEUS Project Home Page. http://www.dia.uniroma3.it/Araneus http://www.difa.unibas.it/Araneus.
2. Java Server Pages (JSP) home page. http://www.java.sun.com/products/jsp/.
3. Oracle Home Page. http://www.oracle.com.
4. 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.
5. 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://www.dia.uniroma3.it/Araneus/.
6. 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.
7. F. Casati, S. Ceri, B. Pernici, and G. Pozzi. Conceptual modeling of workflows. In *14th International Conference on Object-Oriented and Entity-Relationship Modelling, (OOER'95) Gold Coast, Australia, December 12-15, 1995. Lecture Notes in Computer Science, Vol. 1021, Springer-Verlag*, pages 341–354, 1995.
8. Workflow Management Coalition. The workflow reference model. WfMC Document n.TC00-1003, http:/www.wfmc.org, 1995.
9. M. Fernandez, D. Florescu, J. Kang, A. Levy, and D. Suciu. Catching the boat with Strudel: Experiences with a web-site management system. In *ACM SIGMOD International Conf. on Management of Data (SIGMOD'98), Seattle, Washington*, pages 414–425, 1998.

10. 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.
11. F. Garzotto, P. Paolini, and D. Schwabe. HDM – a model-based approach to hypertext application design. *ACM Transactions on Information Systems*, 11(1):1–26, January 1993.
12. D. Georgakopoulos, M. Hornick, and A. Sheth. An overview of Workflow Management: From process modeling to infrastructure for automation. *Journal on Distributed and Parallel Database Systems*, 3(2), 1995.
13. 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*, pages 544–546, 1998. Exhibition Program. `http://www.dia.uniroma3.it/Araneus/`.
14. J. A. Miller, D. Palaniswami, A. P. Sheth, K. Kochut, and H. Singh. WebWork: METEOR_2's web-based workflow management system. *Journal of Intelligent Information Systems*, 10(2):185–215, 1998.
15. 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://www.dia.uniroma3.it/webdb98`, 1998.
16. 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://www.dia.uniroma3.it/webdb98`, 1998.