XGuide - A Practical Guide to XML-based Web Engineering

Clemens Kerer, Engin Kirda and Christopher Krügel

Technical University of Vienna, Distributed Systems Group
Argentinierstr. 8/184-1, A-1040 Vienna, Austria
Phone: +43 1 58801 18418, Fax: +43 1 58801 18491
{C.Kerer, E.Kirda, C.Kruegel}@infosys.tuwien.ac.at

Abstract. *Various approaches have been proposed in the field of Web engineering that attempt to exploit the advantages of XML/XSL technologies. Although a strict separation of presentation and content achieved through XML/XSL has many advantages, a considerable effort is involved in using these technologies to develop Web sites. The lack of experience in XML/XSL can be a major cause for the extra effort. In several XML/XSL-based Web projects, we felt the need for a methodology that systematically guides the developer in the field through the development process while taking into account the limitations and strengths of XML. In this paper, we present XGuide, a practical guide for XML-based Web Engineering that focuses on parallel development. XGuide is a methodology for XML/XSL-based Web development that is tool-independent and hence, can be used with a broad range of development tools. We are currently using the XGuide approach in several Web projects.*

Keywords: XML, Web Engineering, Methodology, Web Service Life Cycle, Parallel development

1 Introduction

For more than seven years the World-Wide Web has been growing rapidly. Organizations were quick to realize the Web's huge potential and it became a powerful and important means to stay in contact with customers, provide online services, express opinions and make money from e-commerce applications. As the functionality of Web sites increased, many new features were implemented that made Web applications more dynamic, interactive and complex. Thus, as the offered information grew and the management complexity of Web sites increased, maintenance often became difficult, too.

Many Web applications and services are developed in an ad-hoc manner today and the main reason is the lack of *practical*-oriented, easy-to-use methodologies, approaches and guidelines. Some authors have referred to the current situation on the Web as the *Web crisis* (e.g., [1]) and have likened it to the *software crisis* (e.g., [2]) in the 1960s when much of the produced software was not reliable and failed to reach basic levels of quality and user satisfaction.

In order to eliminate HTML's shortcomings and to define flexible, extensible standards that address current Web requirements, the World Wide Web Consortium (W3C) defined the eXtensible Markup Language (XML) and the eXtensible Stylesheet Language (XSL). XML is ubiquitous today and is widely used in areas such as data representation and data exchange, repositories and in domain-specific languages (e.g., SVG, MusicML, MathML or VoiceXML).

Although the majority of existing Web sites are not XML-based, there is general consensus that XML will be the Web language of choice in the future. The somewhat slow adoption of XML on the Web is probably because of the complexity of XML and its related technologies (e.g., XSL transformations, XSL formatting objects, XML schema, etc.) in comparison to HTML. Furthermore, tool support for XML-based Web development is still in its infancy compared to the large number of available HTML-based development tools.

Although a strict separation of presentation and content achieved through XML/ XSL has many advantages, a considerable effort is involved in using these technologies to develop Web sites. The lack of experience in XML/XSL can be a major cause for the extra effort. In several XML/XSL-based Web projects, we felt the need for a methodology that systematically guides the developer in the field through the development process and supports parallel development to decrease development time.

In this paper we present XGuide, a practical guide for XML-based Web Engineering. XGuide is a methodology for XML/XSL-based Web development that is tool-independent and that can be used with a broad range of development tools. We are currently using the XGuide approach in several Web projects.

The remainder of this paper is structured as follows: in Section 2 we briefly discuss our previous experiences engineering XML-based Web sites. Section 3 presents the XGuide methodology with the eight important steps we identified for XML-based Web engineering. Section 4 briefly evaluates the XGuide approach. Section 5 presents related Web engineering approaches and Section 6 concludes the paper and lays out future work.

2 Our Previous XML-based Web Engineering Experiences

Our group started working in the area of Web engineering in 1995 when we first implemented the Web presence of the Vienna International Festival (VIF) [3]. The VIF is an annual cultural festival in Vienna that attracts a national and international audience from Europe, Japan and the US.

Since 1995, we have been trying to improve the development processes and the deployed tools to effectively meet the customer requirements and to be able to finish the development work in the short time frame of 5-6 weeks. This is especially important since both the appearance and the functionality of the VIF site evolves every year and we often cannot reuse functionality and components from previous years.

Based on our previous experiences [4] and with the aim of improving our old tools, we built a new tool called MyXML [5, 6]. MyXML fully relies on XML and related technologies to support the development and maintenance of flexible, extensible Web applications. Some of the key requirements for this tool were:

- the strict separation of the content, the layout and the application logic (similar to the Model-View-Controller Pattern [7]),
- a reuse oriented resource management,
- the easy creation of static and dynamic pages,
- built-in support for common concepts on the Web such as database access or CGI parameters
- independence from a particular implementation model or programming language.

We deployed MyXML for the VIF Web presence in the 2001 season and, at the same time, started deploying it in the implementation of the Web site of the Austrian Academy of Science (AAS) [8]. This project had completely different characteristics (many more pages, longer project duration, emphasis on maintenance and evolution, etc.) and helped us to evaluate the versatility of the tool.

Although the deployment of MyXML and XML/XSL technology helped us in general, we also observed several problems when doing XML-based Web site development. We reported our positive and negative XML-based Web engineering experiences in [9].

We observed that the majority of the problems stems from the lack of experience in how to do XML-based Web development. Many developers did not know what challenges (as opposed to HTML-based development) were to be expected. Separating the presentation from the content, for example, was not always easy for the developers to understand and many questions came up that were not always easy to answer: Should a Document Type Definition (DTD) be used? What should be put in the DTD? How do you provide multi-language support using XSL? What should the graphical designers deliver? What is expected from the content managers?

After experiencing similar problems and questions in both the VIF and the AAS project, we realized that a set of guidelines for doing XML-based Web engineering are needed. Such guidelines should not exclusively cover the development process but include the analysis and design phases. Tasks for different roles (such as developers, graphical designers and content mangers) need to be taken into account and support is needed to enable parallel development to decrease development time.

In the next section we present XGuide, a methodology for XML-based Web engineering that covers the basic decisions and requirements that need to be considered in an XML-based Web project.

3 The XGuide Methodology

Web development involves many different tasks and requires different skills to solve them [10, 11]. We define four roles to represent the different kinds of activities involved in the process: the project manager, the content manager, the graphical designer and the programmer.

Clearly, a single person can play several of the above roles in small projects whereas in large ones, a single role can be represented by many people. In the following, we only distinguish between roles, abstracting from who or how many people a role represents.

3.1 Define the Goal and the Characteristics of the XML-Web Site

Before starting the design or implementation, it is important to make sure that all involved roles understand what the key aspects of the Web site are. With the characteristics of a Web site, we refer to functionality-oriented aspects such as the support for multiple languages and the ability to seamlessly switch between them. Another requirement might be that different output devices and/or formats must be supported (e.g., HTML browsers, simple HTML for hand-held devices, WML for mobile phones and PDF for a paper version).

In the following enumeration, we present the aspects we found to be important in our XML projects:

The Lifetime of the Web site is critical to determine how much effort should go into the various tasks. In the case of the VIF, for example, the lifetime of the Web site usually is half a year, in the case of the AAS it is several years.

Multi-lingual support per se is relatively easy to accomplish; this task becomes trickier if the user should be able to switch from a page in one language to the same page in the other language - especially if the pages are created dynamically (e.g., search results, shopping carts, etc.). The complexity further increases if more than two languages have to be supported.

Multi-device support affects the content manager, the graphical designer and the programmer. The content manager has to decide *what* content should be presented on different devices, the graphical designer has to plan the corresponding rendering of the content and the programmer has to take the different output formats and device capabilities into account.

Integration of legacy data/applications mainly concerns the programmer and content manager. The migration of the data or application might be an option if it is exclusively used for Web purposes; this might save time and money in the long run.

Browser-less access to the Web site is increasingly becoming important with the wide-spread use of Web services (i.e., XML-based machine-to-machine communication). Support for browser-less access can also be seen as another output device which must be supported.

External data sources such as remote databases or third-party Web services add additional complexity in terms of performance and/or maintenance. Especially if the data sources do not provide XML data, conversion might be costly.

Evolution of the Web site is another important factor. If new or extended functional requirements are likely to be added frequently, the development process needs to put even more emphasis on flexible and easy-to-change data structures and application logic. If high availability is required, appropriate update mechanisms have to be in place to support the evolution of the site with a minimum interference.

3.2 Decide on the Technology and Infrastructure

Based on the preliminary understanding of the Web site's functionality developed in step 1, the next question that needs to be answered is whether the use of XML technology for the realization of the project makes sense.

According to our experience, this step is sometimes skipped because people implicitly assume that a new technology such as XML is better suited for a task than an existing one. This is not necessarily the case. On the one hand, all parties involved in XML-based Web engineering have to be (or become) knowledgeable in XML and its related technologies; in the latter case this can take considerable educational effort. On the other hand, not all kinds of Web projects are suitable for an XML implementation – especially if the project deals with a small Web site, a Web site with a short time-to-live, or a creative Web site with many different pages that lack any commonalities. Thus, commitment to XML technology might sometimes be counterproductive.

3.3 Develop a Means of Communication

One of the key problems in the Web projects we implemented was the lack of communication among the different roles. For example, content managers did not communicate their requirements or intentions to the graphical designers. Likewise, the graphical designers did not make explicit their constraints. As a result, the content managers did not know what kind of content the graphical designers expected and vice versa. Eventually, either one or both of these roles had to adapt their work to conform to the other's requirements.

We use a simple graphical notation to represent pages, graphics and content and relations between them (rectangles and arrows or anything else the participants feel comfortable with). Figure 1 shows an example diagram illustrating the structure of a class of pages that contain navigational elements, a header and a footer and a content area. The content area contains a list consisting of images and corresponding descriptive texts. On this foundation, it is much easier for the roles to communicate on the same level. To record the outcome of the discussion and make it more explicit and binding, annotations can be added to the diagrams. The annotation number 3 in the diagram (usually a post-it in practice), for example, could be a constraint such as: "Every item in the list

consists of an optional image and a text. There must be at least one item in the list and at most five". Similar constraints can also hold for the other annotations on the diagram.

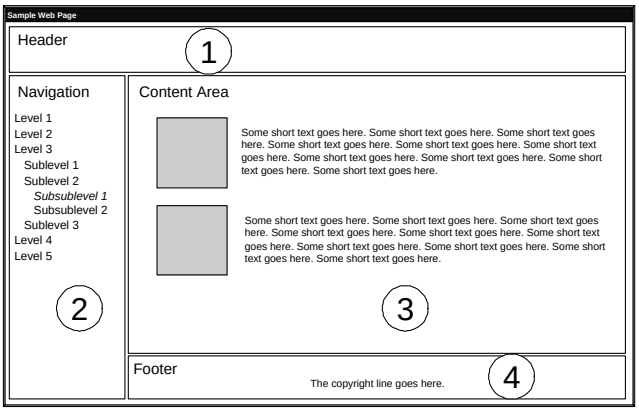

Fig. 1. Example of a graphical notation used in discussions showing the structure of a class of pages

3.4 Use a DTD as Contract

In the VIF project, the only way to meet the deadline was to extensively develop in parallel. Thus, our desire was to enable the content managers, layout designers and programmers to work in parallel and independently of each other.

```
<!--
  <in>
    <param name="studentname" type="String" />
    <param name="grades" type="String[]" />
  </in>
-->
<!ELEMENT webpage (text*, grades, text*)>
<!ELEMENT text (#PCDATA | studentname)*>
<!ELEMENT grades (grade+)>
<!ELEMENT grade (#PCDATA)>
<!ELEMENT studentname (#PCDATA)>
```

Fig. 2. The DTD contract for a page in the Web site

Our approach to achieve this is to define a *contract* that specifies the high-level constraints and interfaces and is binding for all roles. We use a document type definition (DTD) for this purpose. In terms of content, the DTD specifies the structure and the type of the content to be expected; for the graphical designer, this structure is the only important information needed to create corresponding stylesheets to render the content; finally, the programmer must be able to derive from such a contract how the interfaces to the layout/content templates described in Section 2 look like. Figure 2 depicts a contract for a page in the Web site that lists the grades of a student. In addition to the structure and content model of the elements, the interface to the layout/content is specified in the contract. In the example, the student's name (a string) and a list of grades (a string array) are used as input parameters. Similarly, output parameters can be specified to describe the results a Web form may return. Once this contract is fixed, all roles can work independently, i.e., the application logic can be developed without the actual content and layout at hand, the content can be created without any knowledge of the layout and the layout can be designed based on the content's structure as opposed to the content itself.

Since tool support for WYSIWYG creation of XSL stylesheets is only rudimentary compared to HTML tools, in practice an additional step is required to transform HTML layout templates into XSL/CSS stylesheets - again strictly conforming to the content's structure as defined in the contract (i.e., DTD). We obtained good results using this technique. For instance, the application logic of the VIF 2001 Web site could be finished *before* the first layout drafts or content information was available.

3.5 Develop in Parallel

Based on the contract defined in the previous step, the content managers, layout designers and programmers can start to do their job independently of each other. In this section, we briefly discuss each role's task and the main XML-related issues they need to consider.

The Content Management Content management is about storing and maintaining content. In the context of XML-based Web development, this means that the content has to be stored permanently and can be delivered in XML according to the structure defined in the contract (i.e., DTD). A content manager has several options for storing content: in XML files in the file system, in a relational database, in an XML repository or any combination of these. Furthermore, existing content sometimes has to be integrated, too. In any case, a way to generate the XML representation of the content from the actual repository has to be provided. If the content is natively stored in XML format, this is relatively easy. If a relational database or any other non-XML repository is in use, appropriate transformation mechanisms have to be installed.

In the VIF project, we provided prefabricated word processor templates for content managers that were then automatically processed to gather the content.

In the AAS project, we developed WebCUS [12], a Web-based content management tool for relational databases, and used MyXML's ability to query relational databases and present the result as XML. This approach facilitated the reuse of existing relational databases know-how in the AAS, supported a flexible way to edit content via the Web, and provided transparent creation of XML documents using MyXML.

The Layout Definition The aim of layout definition is to provide a set of layout templates that can be applied to the XML content. As mentioned before, direct generation of XSL stylesheets is currently difficult due to missing satisfactory tool support. As a result, graphic companies frequently deliver HTML templates as design mockups that have to be manually transcoded into stylesheets. This process is guided by the structure of the content as defined in the contract. Depending on the graphical appearance of the site, commonalities of these stylesheets can be further extracted, thus facilitating reuse of layout fragments. Using stylesheets has the advantages of separating the layout from the content, supporting reuse of layout definitions and ensuring layout consistency across all pages using the same stylesheet. Despite these desirable properties, the use of stylesheets also bears some restrictions in that exceptions for single pages (e.g., to change the background color for a single page, to add an image only on one page, etc.) are difficult to implement.

The Application Logic The focus of this task is on the functionality (usually) taking place on the server. The programmer is not interested in how the pages look or what other content is available on the site. His only interest is how dynamic content can be given out in the appropriate layout. In other words, he needs access to the 'executable form' of the layout and the static content. Furthermore, dynamically generated content must be integrated at runtime. One way of achieving this is by using generated user interface classes as in MyXML (e.g., [6]). Another possibility is to merge the static with the dynamically generated content first and apply the stylesheet at runtime (e.g., [13–15]).

3.6 Navigational Structures

Navigation bars and hierarchical menus can be found on almost all Web sites today. Especially in large Web projects, the consistent implementation and maintenance of navigational structures for static and dynamic pages can be a difficult and tedious task.

A good solution for this problem is to separate the navigational information from the content. Instead of integrating slightly different navigational constructs in all pages, the whole navigational information is specified only once and stored externally. From this single source of information all navigational structures are generated depending on the actual target page (contextual links).

As we reported in [9], we usually use a separate XML structure to hold the navigational information including the hierarchy or list of all hyperlinks, their representation as text or images and the destination URLs.

3.7 Integration

When the parallel tasks are concluded, they can be integrated to form the final Web site. In the first step, the content documents are processed with stylesheets containing the layout information. In the next step, the navigational information is added. This leaves us with the task to integrate the application logic by calling the appropriate layout classes as discussed in Section 3.5. The above steps are what worked best in our projects, but alternative integration paths can easily be imagined: if CSS stylesheets are used, the browser performs the first step; if the navigational structures need further decoration with layout information, the content should first be merged with the navigational information before applying the style, and so on.

Although the final integration can only take place when all other tasks are concluded, we found the early integration of subsets as soon as they are finished useful. This gives a first indication of whether the specified contracts hold and supports early detection of necessary changes, thus, reducing the overall cost of such changes. Furthermore, it helps the content manager to review parts of the content in the final layout as well as the layout designer to test the layout with real content.

3.8 Implementation, Evolution and Maintenance

Just as in software engineering, in Web projects the requirements keep changing and evolving all the time. To maximize the benefit of parallel development as discussed above, it is crucial to distinguish between implementation and evolution and not to continuously extend the functionality and/or requirements for the Web project. Instead, only when the integration of a given set of features is finished, another round of parallel development dealing with extensions should be started.

The distinction between the evolution and the maintenance of a Web site is often difficult to make; e.g., is the adaptation of a stylesheet part of the site's evolution or maintenance? We usually define activities as maintenance if they affect only a single role but have no consequences for other roles or the contract (e.g., changing the value of a content item or formatting properties of a stylesheet). Due to the independence of the roles with respect to the contract, these activities can be performed easily. If a change of the contract is necessary to implement a new requirement, we talk about evolution of the site that automatically involves all roles and usually requires major updates. While maintenance activities can be performed continuously, evolution cycles are scheduled less frequently and result in a new release or version of the whole site.

4 A Web Site Example

In this section, we discuss a Web site that we implemented according to the steps we define in this paper and illustrate the presented ideas. The Web site

offers information about an introductory course on XML and related technologies (see [16]) and is the primary source of information for students. The functional requirements for the site were quickly defined: a news section should inform about recent updates and accompany more general information about the lecture such as the dates or the grading scheme. Furthermore, the description of the lab examples is offered, a download section provides the lecture slides and required tools for the lab, a grading service reports the points a student earned so far, and a feedback form allows students to contact the lecturer.

The characteristics of this Web site, as discussed in Section 3.1, are presented in the following list:

- the site's lifetime is several years (as long as the course is offered),
- the content is only available in English,
- different output formats shall be supported (i.e., a full-fledged HTML version for Web browsers, a light-weight HTML version for hand-held devices and a printable version in PDF),
- no legacy applications have to be integrated,
- no browser-less access is envisioned,
- only a (relational) database with student information serves as additional source of information, and
- the evolution of the site happens only for the next semester's lecture, i.e., in 6 months steps.

The target environment for our Web site consisted of a Redhat Linux system with Apache as a Web server and Jakarta Tomcat as servlet container. For XML processing, we used Xerces and Xalan in combination with our MyXML development tool.

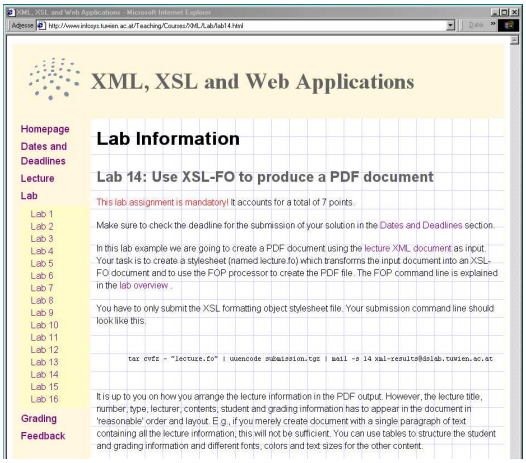

Fig. 3. A full-fledged Web page for viewing in an HTML browser.

In the VIF and AAS projects, we had several different kinds of pages and thus needed several DTDs to describe them. In the course Web site, we planned to have only a single type of page with a header, footer, navigation and content area as shown in Figure 3. A flexible set of content definitions and structures helped to map the content requirements of different pages to the same contract.

We started and finished the application logic before any content or layout was defined. In a second step, we defined the layout based on some test content; the content was added only as the last step. The development of the layout, content and the application logic was thus independent of each other.

The navigation of the site is managed by a hierarchical menu that provides access to the main sections of the Web site and shows shortcuts to the subsections of the currently selected section. As proposed in this paper, we defined the whole navigational structure external to the content and layout and automatically added the appropriate representation to all pages.

The integration of the content with the layout and the application logic worked smoothly. Some changes were made to the layout definition after the actual content had been added; this was mainly done for stylistic reasons and to better render the content on different browsers. Furthermore, the generation of different versions of the site for the different output formats also worked without major problems. A slight adaptation of the content structure was necessary to incorporate information that was only to be present in the printable PDF version of the site. Figure 4 shows the same sample lab exercise depicted in Figure 3 in the alternative output formats of light-weight HTML and PDF.

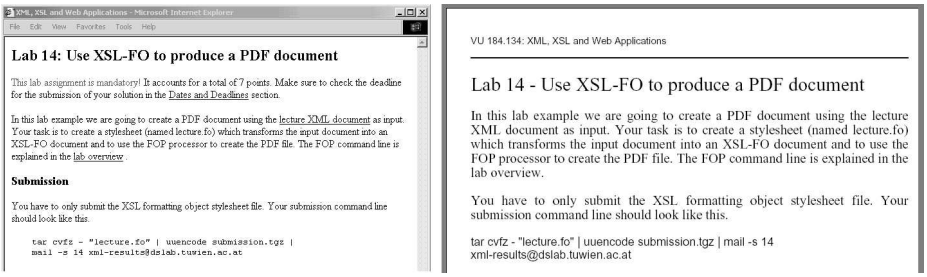

Fig. 4. Additional output formats (a) light-weight HTML for hand-held devices and (b) printable PDF for the same content.

5 Related Work

The Relational Management Methodology (RMM) and the Object Oriented Hypermedia Design Methodology (OOHDM) are two well-known Web site development methodologies. RMM [17] is based on the Entity Relationship (ER)

model [18] and focuses on database-backed Web development. A severe restriction of the original RMM is that the mapping of contents from several entities onto a single Web page was not possible. The extended RMM [19] explicitly introduces the separation of content, layout and application logic (storage level, presentation level and logical level) and focuses on the logical level of a Web application to provide formalized mappings of the content through the logical level onto the presentation level. OOHDM [20] is an object-oriented hypermedia design methodology based on the HDM data model [21] that focuses on the database application domain. In OOHDM, separate phases for the conceptual design, the navigational design and the user interface design are introduced. The conceptual design models the application domain using the notation of Rumbaugh; the navigational design distinguishes nodes, links between nodes and access structures that are the equivalent to our navigational structures.

A graphical design technique, W3DT, is introduced in [22] and extended into eW3DT in [23]. eW3DT represents pages and classes of pages in a hierarchical diagram and has an explicit notion to model database interactions. This methodology is solely targeted at HTML generation, though.

In [24], the authors discuss the analysis and design of Web-based information systems. A *sequential* methodology based on ER analysis, scenario analysis, architecture design, attribute definition and construction is proposed. The architecture is represented in an extended version of RMM's diagram notation. Attributes can be viewed as meta data for later maintenance use. The different stages in the life cycle of a Web application are presented and traditional methodologies such as RMM are used in the design phase.

Another Web application design methodology for mainly data-centric Web sites is WebML[25]. WebML defines a structural model for the content, a hypertext model for content composition and navigation, a presentation model and a personalization model. The proposed design process covers similar aspects as our methodology, but does not support parallel development and does not have much support for logic development.

The Extensible Web Modeling Framework (XWMF) [26] is an RDF [27] based approach towards Web engineering. It focuses on the specification of a reusable hierarchy of fragments that eventually form a Web page. These specifications are extensible and can be enriched with other meta data. Although the implementation of such a model can be generated automatically, no strict separation of layout, content and logic is possible.

6 Conclusion and Future Work

We are currently deploying XGuide for the design, development and maintenance of this year's VIF Web presence. Currently, we are using MyXML for the logic, but also plan to test XGuide with other development tools such as Cocoon [14].

We are also extending the contract model to better describe the properties of a Web page and add extensibility with respect to new aspects such as security or workflow. Furthermore, we extend our ideas to support components (e.g.,

content components, layout components, logic components, etc.) and their reuse in the Web development process.

Easy support for different output devices is another area of our active research. The goal is to define an architecture to support arbitrary output formats with minimal changes to existing components. We are working on a runtime environment where support for new devices is transparent to the programmer and the content manager. Only the person responsible for the layout needs to take the capabilities of the output devices into account.

We presented a Web engineering methodology that focuses on XML-based Web development. We exploit the strengths of XML and its related technologies to achieve flexibility and design for change by strictly separating the content, the layout and the application logic information. Furthermore, parallel development supports a reduced time-to-market – a critical factor in most Web projects.

Many Web applications and services are developed in an ad-hoc manner today and the main reason is the lack of *practical*-oriented, easy-to-use methodologies, approaches and guidelines. Unlike many other development methodologies, XGuide is based on standards such as XML and XSL, thus enabling a rich set of tools to be used. In this context, XGuide attempts to combine the abstract concepts of a Web engineering methodology with the actual technology used for the implementation.

Acknowledgments

The authors would like to thank the Vienna International Festival and the Austrian Academy of Science for their financial support and cooperation.

References

1. Ginige, A., Murugesan, S.: Web Engineering: An Introduction. IEEE Multimedia, Special Issue on Web Engineering **8** (March 2001) pp. 14–18.
2. Sheppard, D.: An Introduction to Formal Specification with Z and VDM. The McGraw-Hill International Series in Software Engineering (1995)
3. Vienna International Festival: VIF homepage, http://www.festwochen.at/ (2001)
4. Kirda, E., Jazayeri, M., Kerer, C., Schranz, M.: Experiences in Engineering Flexible Web Services. IEEE Multimedia **8** (January - March 2001) pp. 58–65.
5. Kirda, E., Kerer, C.: MyXML: An XML based template engine for the generation of flexible Web content. In: Proceedings of Webnet 2000 Conference, San Antonio, Texas. (Nov 2000)
6. Kerer, C., Kirda, E.: Logic, Layout, and Content Separation in Web Engineering. In: Proceedings of the 9th World Wide Web Conference, 3rd Web Engineering Workshop, Amsterdam, The Netherlands. (May 2000)
7. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading Mass. and London (1995)
8. Austrian Academy of Science: AAS homepage, http://www.oeaw.ac.at/ (2001)

9. Kerer, C., Kirda, E., Jazayeri, M., Kurmanowytsch, R.: Building XML/XSL-Powered Web Sites: An Experience Report. In: Proceedings of the 25th International Computer Software and Applications Conference (COMPSAC), Chicago, IL, USA, IEEE Computer Society Press (October 2001)
10. Rosenfeld, L., Morville, P.: Information Architecture for the World Wide Web. O'Reilly & Associates (Feb. 1998)
11. Streitz, N.A.: Designing Hypermedia: A Collaborative Activity. Communications of the ACM **38** (August 1995)
12. Kerer, C., Kirda, E., Kurmanowytsch, R.: WebCUS: A generic Web-based Database Management Tool powered by XML. IEEE Internet Computing (to appear) (2002)
13. Barta, R., Schranz, M.W.: JESSICA – An Object-Oriented Hypermedia Publishing Processor. Computer Networks and ISDN Systems **30** (Apr. 1998) p. 281.
14. Mazzocchi, S.: The Cocoon Project Home Page, http://xml.apache.org/cocoon/ (1999-2001)
15. Webmacro: Webmacro Home Page, http://www.webmacro.org (2001)
16. Kerer, C.: XML, XSL and Web Applications Homepage, http://www.infosys.tuwien.ac.at/xml/ (2001)
17. Isakowitz, T., Stohr, E.A., Balasubramanian, P.: RMM : A Methodology for Structured Hypermedia Design. Communications of the ACM **38** (August 1995) pp. 34–44.
18. Teorey, T., Yang, D., Fry, J.: A logical Design Methodology for Relational Databases Using the Extended Entity-relationship Model. ACM Computing Surveys **18** (1986) pp. 197–222.
19. Isakowitz, T., Kamis, A., Koufaris, M.: The Extended RMM Methodology for Web Publishing, Working Paper IS98 -18, Center for Research on Information Systems (1998)
20. Schwabe, D., Rossi, G.: The Object-Oriented Hypermedia Design Model. Communications of the ACM **38** (August 1995) pp. 45–46.
21. Garzotto, F., Paolini, P., Schwabe, D.: HDM - A Model-based Approach to Hypermedia Application Design. ACM Transactions on Information Systems **11** (1993) pp. 1–26.
22. Bichler, M., Nusser, S.: Modular Design of Complex Web-Applications with W3DT. In: Proceedings of the 5th Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE '96), IEEE Comput. Soc. Press., Los Alamitos, CA, USA (1996) pp. 328–333.
23. Scharl, A.: Reference Modeling of Commercial Web Information Systems Using the Extended World Wide Web Design Technique (eW3DT). In: Proceedings of the 31st Hawaii International Conference on System Sciences (HICSS-31), Hawaii, USA, IEEE Computer Society Press (1998)
24. Takahashi, K., Liang, E.: Analysis and Design of web-based Information Systems. In: Proceedings of the 6th International World Wide Web Conference, Santa Clara, CA, USA. (1997)
25. Ceri, S., Fraternali, P., Bongio, A.: Web Modeling Language (WebML): a modeling language for designing Web sites. In: Proceedings of the 9th World Wide Web Conference, Amsterdam, Netherlands. Volume 33 of Computer Networks., Elsevier Science B.V (2000) pp. 137–157.
26. Klapsing, R., Neumann, G.: Applying the Resource Description Framwork to Web Engineering (2000)
27. Lassila, O., Swick, R.R.: Resource Description Framework (RDF) Model and Syntax Specification. Technical report, World Wide Web Consortium (1999)