Web Service Engineering with DIWE

Engin Kirda[*], Clemens Kerer[*], Christopher Kruegel[‡] and Roman Kurmanowytsch[*]

[*]Technical University of Vienna, Distributed Systems Group
Argentinierstr. 8, 184/1 1040 Vienna, Austria
{E.Kirda,C.Kerer,R.Kurmanowytsch}@infosys.tuwien.ac.at

[‡]University of California, Santa Barbara
CA 93106, USA
chris@cs.ucsb.edu

Abstract

A Web service is frequently defined as browser-less access to content on a Web site. The industry's focus to date has been on providing easy-to-use low-level libraries, tools and technologies to enable the rapid construction of Web services. The problem of how to support the automatic integration of Web services into Web sites has received less attention. In this paper, we describe and show how SOAP-based Web services can be modeled, implemented, composed and automatically integrated into Web sites using the Device-Independent Web Engineering (DIWE) framework. The framework provides support for the separation of layout, content and application logic in Web sites and automatically generates Web service support for the browser-less access to the content and the functionality.

Keywords: Web Service Engineering, Web Services, SOAP, Modeling and Integrating Web Services into Web sites

1 Introduction

Until the late 90s, the focus of Web research has been the development of tools, technologies and methodologies for the design, implementation and maintenance of HTML-based Web sites. The common assumption was that a Web site would always be accessed by a browser found on a personal computer or a laptop. Concurrently with the miniaturization and widespread availability of processors have come advances in networking capabilities, enabling ever more devices to communicate with each other by connecting to existing network infrastructures such as the Internet, the telephone network, or private company or home intranets. The Web of the future will not only be accessed by users using their personal computers and laptops, but by users using a wide variety of *Web devices* (any hardware or software that can be used to access Web content[19]) such as telephones equipped with speech recognition software, watches, digital televisions and Personal Digital Assistants (PDAs). One of the next challenges faced by the research community and the World Wide Web Consortium (W3C) is the definition of standards, tools and technologies for the "browser-less Web".

The W3C's eXtensible Markup Language (XML) and eXtensible Stylesheet Language (XSL) standards have paved the way in creating the browser-less Web by providing a basic flexible infrastructure to independently define content and presentation (i.e., layout) information and exchange data among parties on the Web. The XML and XSL standards have been designed to eliminate HTML's shortcomings such as its poor support for flexibility and device-independence.

Although the term *Web Service* has been used since the mid 90s to describe the collection of static and dynamic information offered to users on a Web site (e.g., see [7, 13, 18, 21]), it is now often being used to denote browser-less access to content on a Web site (e.g., see [1, 8, 23]).

In order to keep a consistent meaning in the context of this paper, we define the following terms:

- **Web Service**: An XML-based content delivery mechanism which is primarily intended for machine-to-machine communication. A Web service provides browser-less access to Web content.

- **Web page**: Static or dynamic content on a Web site that is intended for browser-access and that is accessible through a unique URL.

- **Web site**: Collection of *Web pages* and *Web services* in a *single* domain (e.g., the Web site of a company).

The industry's focus to date has been on providing easy-to-use low-level libraries, tools and technologies to enable the modeling and the rapid construction of Web services. The problem of how to support the automatic integration of Web services into Web sites has received less attention.

The Simple Object Access Protocol[25] (SOAP) is one technology that has gained popularity and has raised interest. SOAP is an XML-based lightweight protocol for exchange of information in a decentralized, distributed environment. SOAP can potentially be used in combination with a variety of transport protocols (e.g., SMTP, FTP), but it is generally used in combination with HTTP. Companies such as Microsoft and IBM have been integrating SOAP support into their most recent products (e.g., the .NET platform). SOAP was submitted to the W3C for the formation of a working group in the area of XML-based protocols and to standardize the work (i.e., the W3C Working Group on XML Protocol[26]).

In this paper, we present an approach to model, implement, integrate and compose Web services into Web sites with the DIWE (Device-Independent Web Engineering) framework. The framework provides support for the separation of layout, content and application logic in Web sites and automatically generates Web service support for the browser-less access to the content and the functionality. We have used DIWE to engineer parts of the Web site of the annual Vienna International Festival.

This paper is structured as follows: Section 2 gives a brief overview of the DIWE framework. Section 3 discusses how Web services are used in DIWE and demonstrates how they can be implemented, integrated and composed to build Web sites. Section 4 presents related work, Section 5 briefly presents a case study and Section 6 concludes the paper.

2 Brief overview of DIWE

The DIWE framework consists of the *MyXML* language, a compiler that can interpret the language and several run-time components that are configured and deployed on the Web server to provide access to the Web site. The framework supports the design, implementation, deployment and maintenance stages of Web sites[17]. Figure 1 illustrates the different stages DIWE supports. For each Web page that is modeled and built using DIWE, Web services that provide browser-less access to the contents of the page are automatically generated. Furthermore, the framework allows the integration and composition of external Web services in building new Web pages and sites.

During the design stage of a Web site, the content, layout and the application logic are designed and defined. The application logic is written using a technology of choice such as Java servlets.

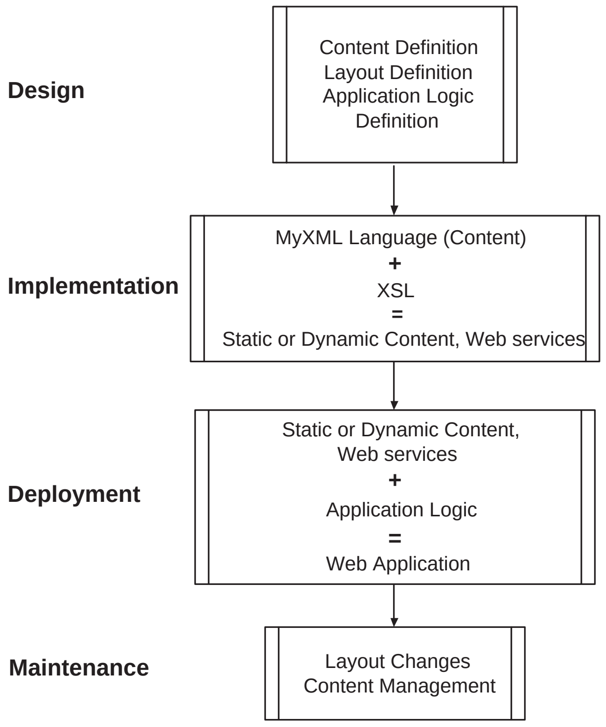

Figure 1. Web site design, implementation, deployment and maintenance with DIWE

A Web page in DIWE is modeled with the MyXML language. The layout is defined using XSL stylesheets. The MyXML language supports the complete Layout/Content/Logic (LCL) separation and is used by the Web developers to design and define the content, the interfaces to the application logic and the integration and composition rules for external Web services.

During the implementation stage, a MyXML language compiler integrates the layout and generates static content embedded in HTML or XML, or source code that provides interactive functionality. Web services are also automatically generated that provide browser-less access to the content.

The MyXML language used for Web page and service specification is a simple XML-based language that uses loops, variables and database access functions. One of main advantages of an XML-based Web specification language is that it allows the definition of functionality that is technology-independent. Any popular programming or scripting language can be generated from the XML and

XSL specifications with an appropriate MyXML language compiler. A complete description of the MyXML language is beyond the scope of this paper. The reader is referred to [15, 17] for more information.

In the deployment stage, the resulting dynamic or static content and the corresponding Web services are deployed on the Web server along with the application logic.

During the maintenance phase, layout changes are integrated by adapting the XSL layout definitions and the content in XML files is managed.

3 DIWE and SOAP-based Web Services

The Web service support in DIWE consists of three parts: *Service creation*, *service integration* and *service composition*. Service creation denotes the modeling and implementation of SOAP-based Web services. Service integration denotes the inclusion of an external Web service into a Web page or Web service that is being created. For example, accessing and retrieving content from an external weather reporting service and displaying it in a Web page is service integration. Service composition, on the other hand, denotes the creation of a new Web page or Web service by integrating two or more external services. For example, showing stock information that is retrieved from an external stocks service based on the current time from a time service is a typical example of Web service composition. Web service composition includes the interactions between independent, distributed Web services (e.g., retrieving content from one service and passing it to another as input).

The prototype implementation of the DIWE framework uses the Apache SOAP toolkit[3], JLex[6] and JCup[2] for the code generation, and Apache Xerces[4] and Xalan[5] as the XML and XSL processors. In the prototype, Java source code (i.e., Java class definitions) are generated.

In Section 3.1, 3.2 and 3.3, we give simple examples of flexible service creation, integration and composition in DIWE.

3.1 Web service creation

Imagine we are building a Web page with DIWE. The page has a form where one can enter and submit the (unique) last name of a person and receive her phone number as a result.

Every Web page that is created in DIWE is also automatically a Web service. For every page that is created with the MyXML compiler, a corresponding Apache SOAP deployment descriptor in XML is generated. The name of the generated service is always of the form <Generated Page Name>Soap (e.g., for the generated dynamic page *PhoneNumber* it would be *PhoneNumberSoap*).

The Apache deployment descriptor is used to *install* a Web service so that it is accessible over the Web via SOAP using the Apache SOAP Toolkit. The descriptor provides information to clients about this service such as the parameters it accepts and the method calls it exports. Figure 2 depicts the generated deployment descriptor for the phone number page.

Constructs of the MyXML language (e.g., database queries) are processed by the MyXML compiler and code is automatically generated that exports contents via SOAP. For example, Figure 3 shows the part of the generated Java code for the phone number example we gave that is used for SOAP invocations. Note that the code wraps the results of the database query into an XML stream of the form *<soap>* *<phoneList_phoneNumber/>* *</soap>*.

The name of the tags are generated based on the default values of the query and database field names. However, the MyXML language provides means to process the XML stream that is generated by using an XSL stylesheet to select and adapt it. Hence, the developer can adapt the content generated by the SOAP interface according to the requirements. It is possible to have less or more information offered over the SOAP interface than the generated Web page. This setting would be used, for example, if the developer would like to display first names and last names in the Web page, but only last names in the SOAP interface and vice-versa.

3.2 Web service integration

Now that we have created a Web page and Web service in the previous section, suppose we would like to integrate this service into another page with a different look-and-feel (i.e., layout).

To create this new page, we first model the page with the MyXML language. Then we define the layout information in an XSL file and invoke the MyXML compiler. Figure 4 depicts the integration of an external Web service content into the new Web page. The *<myxml:soap>* element from the MyXML language namespace is used to choose the external service, indicate the URL it can be accessed by and pass it parameters it requires. In this case, we are accessing the **PhoneNumberSoap** service that we just created and which is located on the server *kirda.kerer.at*. We pass to this service the CGI parameter *theName* that we accept in our page. You can see the mapping between the *theName* CGI parameter that we accept and the *nameParameter* parameter that the service we are accessing accepts.

When the MyXML compiler is invoked, it sees that we would like to integrate a SOAP service into our page and when the Java code is created, the functionality necessary for accessing this SOAP service as a client is generated by the compiler.

```
<isd:service xmlns:isd="http://xml.apache.org/xml-soap/deployment"
      id="urn:PhoneNumberSoap">
<isd:provider type="java" scope="Request" methods="printSoapData">
<isd:java class="PhoneNumber" static="false"/>
</isd:provider>
<isd:faultListener>
      org.apache.soap.server.DOMDefaultListener
</isd:faultListener>
<isd:mappings>
</isd:mappings>
</isd:service>
```

Figure 2. Generated deployment descriptor for the Apache SOAP toolkit

```
<…>
if (soapMode> out.println("<phoneList_phoneNumber>");
            out.println(phoneList.getString("phoneNumber"));
if (soapMode) out.println(""</phoneList_phoneNumber);
<…>

public String printSoapData(Hashtable hash) {
      this.hash=hash;
      soapMode=true;
      ByteArrayOutputStream bout = new ByteArrayOutputStream();
      PrintWriter out = new PrintWriter(bout);
      out.println("<? xml version=\"1.0\" ?>\n<soap>");
      printContentsImpl(out);
      out.println("</soap>");
      out.flush();
      return bout.toString();
}
```

Figure 3. Part of the code generated for SOAP

The Java code accesses the SOAP service at run-time and retrieves XML content from it. This is where *post-stylesheets* are used. To be able to add a layout to this XML content, the developer provides another XSL stylesheet. This post-stylesheet is used at run-time to format the XML content received from the external Web service and to integrate it into the page.

The use of post-styles and external Web services requires knowledge about the XML structure of the result the service returns. Before integrating a service, the developer needs to check and find out the structure and the semantics of the information the service is offering (e.g., by use of Document Type Definitions (DTDs), XML Schemas, Web Service Description Language, or accompanying human-readable textual descriptions – in the prototype implementation, we do not support automatic DTD, WSDL or XML Schema generation yet).

Note that the new page we just created also has a generated Web service interface itself. In the example we gave, the new service takes the parameters that it receives and contacts the **PhoneNumberSoap** service that actually provides the functionality. It passes the results it receives from **PhoneNumberSoap** directly back to the client. Hence, it is possible to build services that depend on other services.

3.3 Web service composition

The ability to integrate an external Web service into a Web page is useful in many situations. For example, a weather information Web site would probably like to enable external Web sites to integrate the weather news into their sites and to enable various Web devices to access and

```
<phoneNumberIntegrator
      xmlns:myxml="http://www.infosys.tuwien.ac.at/ns/myxml">
<soapCall>
      <myxml:soap>
            <myxml:urn> PhoneNumberSoap </myxml:urn>
            <myxml:url>
            http://kirda.kerer.at:8080/soap/servlet/rpcrouter
            </myxml:url>
            <myxml:parameter>
            nameParameter,<myxml:cgi>theName</myxml:cgi>
            </myxml:parameter>
      </myxml:soap>
</soapCall>
</phoneNumberIntegrator>
```

Figure 4. Integrating a Web service into a page

```
<phoneNumberComposer
      xmlns:myxml="http://www.infosys.tuwien.ac.at/ns/myxml">
<getPhoneNumber>
      <myxml:soap name="PhoneNumber" visible="false">
            <myxml:urn> PhoneNumberSoap </myxml:urn>
            <myxml:url>
            http://kirda.kerer.at:8080/soap/servlet/rpcrouter
            </myxml:url>
            <myxml:parameter>
            nameParameter,<myxml:cgi>theName</myxml:cgi>
            </myxml:parameter>
      </myxml:soap>
</getPhoneNumber>

<getAddress>
      <myxml:soap visible="true">
            <myxml:urn> AddressSoap </myxml:urn>
            <myxml:url>
            http://kirda.kerer.at:8080/soap/servlet/rpcrouter
            </myxml:url>
            <myxml:parameter>
            address,<myxml:reference>PhoneNumber</myxml:reference>
            </myxml:parameter>
      </myxml:soap>
</getAddress>
</phoneNumberComposer>
```

Figure 5. Composing a Web service from two other services

process the content in XML.

Taking the integration example we gave in the previous section a little further, sometimes it would also be useful to *combine* Web services and to allow them to interact with each other.

Imagine we would like to find a person's address after we have found his/her phone number using the **PhoneNumber-Soap** service. Suppose there is already an existing service that does this. This service accepts a phone number as parameter and returns the address as result. Such services are sometimes offered by phone companies.

Figure 5 shows how services can be composed using the MyXML language. Note that composition is similar to integration. The first <*myxml:soap*> definition has a *name* and *visible* attribute. The first attribute gives a name to the <*myxml:soap*> block so that it can be referenced later. The second attribute, *visible*, indicates that the content delivered by this SOAP Web service *does not* need to be displayed on the Web page. The second <*myxml:soap*> block does not have a *visible* attribute. Hence, the default reaction of the processor is to integrate the content into the page after it has been processed with a post-stylesheet. Note that the <*myxml:parameter*> in the second block takes the result of the first SOAP block and sends it to the *Address Soap* service (using <*myxml:reference*>). Both services are located on the server *kirda.kerer.at*.

This composition mechanism is a simple, but effective way of constructing Web services and creating Web pages that use external content. The example given in Figure 5 is again a Web service itself that depends on two other Web services.

4 Related Work

For the last couple of years there has been a growing interest in Web services. Much has been written about technologies, solutions and frameworks for building Web services and large companies such as Sun Microsystems, Microsoft and IBM have been active in the area.

There have recently been proposals for XML-based Web service composition languages such as Microsoft's XLANG, IBM's Web Service Flow Language (WSFL), and their joint Business Process Execution Language [12]. In [24], a high-level, UML-based approach is presented for modeling and composing Web services. These approaches, however, solely focus on the integration and composition of Web services whereas the approach we present in this paper focuses on the integration and composition of Web services *into* Web sites. We, therefore, view a Web service as functionality that complements the typical functionality provided by a Web site and focus on the Web engineering problem [11] of using Web services.

Sun has been advocating the Open Network Environment (ONE) which, according to Sun's definition[23], is a platform "to provide a flexible and cost-effective environment where data, applications, reports and transactions can be developed, deployed, discovered and utilized". In [23], Sun advocates that Web services are not anything new and are "self describing software components that can automatically discover and engage other Web components to complete tasks over the Internet".

Microsoft's ASP.NET framework has extensive support for the creation of Web pages and Web services. The Visual Studio graphical development environment enables Web developers to rapidly create Web pages, Web sites and Web services. For example, in Microsoft's C# (and other .NET languages) it is easy to export C# methods as Web services. Furthermore, C# code that is written can easily be installed as a dynamic Web application in the platform. However, the way .NET deals with Web applications and Web services is still quite low-level. Web service integration and composition are only supported in the code-level where the developer has to write the code herself. The framework lacks a higher-level, language-independent model for dealing with Web pages, sites and services.

IBM Alphawork's Web Services Invocation Framework (WSIF)[1] provides an API to invoke services, no matter how or where the service is. It uses WSDL descriptions to retrieve information about services and allows the programmer to work with representations of services instead of working directly with SOAP APIs. The programmer can work with the same programming model regardless of how the service is implemented and accessed. Although WSIF provides a higher-level abstraction in dealing with Web services, it is still a programming library and does not provide any support for Web page creation or any higher-level support for Web service integration or composition.

Much has also been written about methodologies, tools and technologies to develop Web applications and Web sites. Well-known Web application development methodologies such as the Object-Oriented Hypermedia Development Methodology[22] (OOHDM) and the Relationship Management Methodology[14] (RMM) are hypertext-based and do not consider issues involved in the construction of Web services.

Web development tools such as Strudel[9] and WebML[7] deal with Web requirements such as layout flexibility, multi-device access to Web content and content integration from heterogeneous data sources. However, they do not tackle new Web site problems such as the ability to create and maintain Web pages and sites from distributed, external Web services.

An extensive survey of methodologies, technologies and tools for Web site development can be found in [10]. We note, however, that none of the tools or approaches discussed in the survey directly deal with or support Web services as we do.

To our knowledge, the most similar Web development framework to DIWE is the Apache Cocoon project[20]. Cocoon supports the separation of Layout, Content and Logic and has a flexible architecture. While Cocoon has a rich set of tools for publishing Web documents, the eXtensible Server Pages (XSP) technology that it uses still mixes layout and logic to a certain degree. Cocoon has become highly flexible in the new Cocoon 2 framework and has recently started supporting the integration of SOAP Web services into Web pages using a technique that is similar to that in DIWE. Unlike DIWE, however, Cocoon does not support the automatic generation of Web services.

5 Case study

Since 1995, our group has been managing and implementing the Web presence of the Vienna International Festival (VIF). We have been using the VIF Web site as a test-bed for the techniques and tools we have developed. The VIF is a major cultural event in Vienna that lasts five weeks. The festival attracts tourists and visitors from around the globe and during the festival, events such as operas, musicals and art exhibitions are organized in different parts of the city. The festival Web site offers detailed information about the events that are taking place and an e-commerce component enables visitors to buy tickets online.

We have built a version of this e-commerce component with DIWE that also provides browser-less access using the automatically generated Web services. The component can be accessed via the traditional HTML-based Web interface or the automatically generated SOAP Web service interface. Parts of the VIF site were also reconstructed by integrating and composing Web services.

Although run-time XSL transformations with post-stylesheets do not cause significant performance problems in the case-study we implemented, we note that there is a need to investigate performance issues in such systems that depend heavily on XSL transformations. XSL transformations are especially critical if a Web service is returning large volumes of XML content.

Building Web sites based on Web services can be more complex than traditional XML/XSL-based Web site engineering as a higher number of XSL stylesheets and dependencies are involved. As reported in [16], the planning overhead of XML/XSL-based Web sites is higher than HTML-based Web sites. This complexity may increase with the integration of Web services. Nevertheless, the flexibility achieved through the deployment of such concepts usually pays-off during the maintenance phase because the functionality and content can often be re-used. For example, instead of writing new code to extract information from a database, we can contact an existing service and select the information using XSL stylesheets. This also enables us to put different user interfaces on top of the same information, thus providing support for different Web devices.

6 Conclusion

The importance of browser-less access to content on the Web will continue to increase in the near future. Web sites will need to support a wide variety of Web devices, XML dialects and interfaces.

Most existing Web service tools and technologies are code-dependent and do not address the Web engineering problem of integrating and composing Web pages and sites from Web services. The industry has mainly been focused on providing easy-to-use low-level libraries, tools and technologies to enable the rapid construction of Web services. Although these tools and technologies can be used to implement the examples we presented in this paper, multiple versions of Web pages, services and applications often need to be provided in parallel (e.g., the construction of an HTML-based application and a separate SOAP-based application for browser-less access) and the effort necessary is thus higher.

In this paper, we focused on the problem of using Web services in constructing Web sites and described and showed how SOAP-based Web services can be modeled, implemented and composed into Web sites using the Device-Independent Web Engineering (DIWE) framework. The framework provides support for the separation of layout, content and application logic in Web sites and automatically generates Web service support for the browser-less access to the content and the functionality.

Acknowledgments

The authors thank the Vienna International Festival for their financial support and cooperation during the development of the DIWE framework.

References

[1] Alphaworks. Web Services - http://www.alphaworks.ibm.com/webservices.

[2] S. Anian. JCup: CUP Parser Generator for Java - http://www.cs.princeton.edu/ appel/modern/java/CUP/ , 2001.

[3] Apache. SOAP Toolkit Homepage - http://xml.apache.org/soap, 2001.

[4] Apache. Xerces XML Parser - http://xml.apache.org/xerces-j , 2001.

[5] Apache. Xalan XSL Processor - http://xml.apache.org/xalan-j , 2001.

[6] E. Berk. JLex: A Lexical Analyser Generator for Java-http://www.cs.princeton.edu/ appel/modern/java/JLex/, 2001.

[7] S. Ceri, P. Fraternali, and A. Bongio. 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*, page 137 157. Elsevier Science B.V, May 2000.

[8] Developmentor. *Essential .NET :Component Development with C#*, 2001.

[9] M. Fernandez, D. Florescu, J. Kang, and A. Levy. Catching the Boat with Strudel: Experiences with a Web-Site Management System. In *Proceedings of Sigmod '98, Seattle, Washington, USA*, page 414 425, June 1998.

[10] P. Fraternali. Tools and approaches for developing data-intensive applications: A survey. *ACM Computing Surveys*, 31(3):227 263, 1999.

[11] A. Ginige and S. Murugesan. Web Engineering: An Introduction. *IEEE Multimedia, Special Issue on Web Engineering*, 8(1):14–18, March 2001.

[12] IBM.
http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/, 2003.

[13] D. B. Ingham, S. J. Caughey, and M. Little. Supporting highly manageable Web services. In *Proceedings of the 6th International World Wide Web Conference, Santa Clara, California,* number 29 in Computer Networks and ISDN Systems, page 1405 1416. Elsevier Science, 1997.

[14] T. Isakowitz, E. A. Stohr, and P. Balasubramanian. RMM: A Methodology for Structured Hypermedia Design. *Communications of the ACM*, 38(8):34–43, August 1995.

[15] C. Kerer and E. Kirda. Layout, Content and Logic Separation in Web Engineering. In *Proceedings of the 9th International World Wide Web Conference, 3rd Web Engineering Workshop, Amsterdam, Netherlands, May 2000*, number 2016 in Lecture Notes in Computer Science, page 135 147. Springer Verlag, 2001.

[16] C. Kerer, E. Kirda, M. Jazayeri, and R. Kurmanowytsch. 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.

[17] E. Kirda. *Engineering Device-Independent Web Services*. PhD thesis, Technical University of Vienna, 2002.

[18] E. Kirda, M. Jazayeri, C. Kerer, and M. Schranz. Experiences in Engineering Flexible Web Services. *IEEE Multimedia*, 8(1):58–65, April-June January - March 2001.

[19] H. W. Lie and J. Saarela. Multipurpose Web Publishing: Using HTML, XML, and CSS. *Communications of the ACM*, 42(10), October 1999.

[20] S. Mazzocchi. The Cocoon Project Home Page, http://xml.apache.org/cocoon/, 1999-2000.

[21] M. W. Schranz. Management process of WWW services: An Experience Report. In *Proceedings of the 9^{th} International Conference on Software Engineering and Knowledge Engineering (SEKE '97),Madrid, Spain*, pages 16–23. Knowledge Systems Institute, June 1997.

[22] D. Schwabe and G. Rossi. The Object-Oriented Hypermedia Design Model. *Communications of the ACM*, 38(8):45–6, August 1995.

[23] Sun. Implementing Services on Demand with the SUN Open Net Environment – Sun ONE. Technical report, Sun Microsystems.

[24] S. Thoene, R. Depke, and G. Engels. Process-Oriented Flexible Composition of Web Services with UML. In *Proceedings of the Joint Workshop on Conceptual Modeling Approaches for e-Business, (eCOMO 2002, at ER 2002 conference), Temper, Finland*, October 2002.

[25] W3C.
Simple Object Access Protocol V1.1 - http://www.w3.org/TR/SOAP. Technical report, 2000.

[26] W3C.
XML Protocol Activity Home Page - http://www.w3.org/2000/xp, 2001.