Building Web Information Systems using Web Services

Flavius Frasincar[1,2], Geert-Jan Houben[1,3], and Peter Barna[1]

[1]Eindhoven University of Technology
PO Box 513, NL-5600 MB Eindhoven, the Netherlands
{flaviusf, houben, pbarna}@win.tue.nl

[2]Erasmus University Rotterdam
PO Box 1738, NL-3000 DR Rotterdam, the Netherlands
frasincar@few.eur.nl

[3]Vrije Universiteit Brussel
Pleinlaan 2, B-1050 Brussels, Belgium
gjouben@vub.ac.be

Abstract. Hera is a model-driven methodology for designing Web Information Systems. In the past a CASE tool for the Hera methodology was implemented. This software had different components that together form one centralized application. In this paper we present a distributed Web service-oriented architecture for the Hera software in which components are mapped to Web services. The present CASE tool is based on two Web services: a Data Service which maps to the data component and a Presentation Service which maps to the presentation component. For these two Web services a detailed description of their interfaces in WSDL is given as well as examples of SOAP messages exchanged between these services and a client. The paper also proposes how to extend the present architecture in order to include other services like an Adaptation Service for performing presentation adaptation based on a user profile and a Profile Service as a shared memory service for user profiles.

1 Introduction

The Web has more than three billion pages and around half a billion users. Its success goes beyond national frontiers and imposes it as the first truly global information medium. While the nineteenth century was dominated by the "industrial revolution", the beginning of this new century is marked by an "information revolution" with the Web as its main "engine".

A Web Information System (WIS) [1] is an information system that uses the Web paradigm to present data to its users. In order to meet the complex requirements of a Web Information System several methodologies have been proposed for WIS design. In the plethora of proposed methodologies we distinguish the model-driven ones that use models to specify the different aspects involved in the WIS design. The advantages of such model-based approaches are countless, better understanding of the system by the different stakeholders, support for reuse of previously defined models, checking validity/consistency between different design artifacts, (semi-)automatic model-driven generation of the presentation, better maintainability etc.

Based on the principle of separation of concerns, model-driven methodologies like OOHDM [2], WebML [3], UWE [4], SiteLang [5], and Hera [6] distinguish: a conceptual model that describes the schema of the (multimedia) data to be presented, a navigation model that specifies how the user will be able to navigate through the data, and a presentation model that gives the layout and the low-level characteristics (e.g. font size, color etc.) of the hypermedia presentation to be generated. For the model representations, these methodologies make use of different technologies: OOHDM is based on the OO paradigm, WebML offers a set of visual units (serialized in XML) to be graphically composed, UWE is based on UML notation extended with a Web UML profile, SiteLang suggests a mathematical language (story algebra), and Hera uses Semantic Web technology to make explicit the semantics captured in a certain model and to subsequently exploit this semantics.

Today there is an increasing need for making WIS components interoperable. An isolated WIS is not able to provide all the information/computational power required by an organization. Web services (WSs) appear to be the most popular mechanism to support application interoperability. Their success is based on the fact that they are cross-platform and cross-language distributed applications. The fact that the provider and the requester of a WS are loosely coupled ensures application robustness. A disadvantage of such an approach is the XML messaging involved, which affects application speed. Nevertheless using appropriate optimization techniques and/or fast computers/networks this disadvantage can be overcome. We distinguish between two classes of WSs: WSs that provide data-on-demand and WSs that offer computation-on-demand. As an example for the first type of WSs we mention WSs offered by Amazon or Google and for the second type of WSs we refer to the services developed in the Grid project [7].

Different science communities saw the opportunity of using WSs to enable data sharing between their systems. For example, the geoscience community came up with a service-oriented solution for the interoperability of geographical information systems aiming at providing better forecasts. The proposed WS solution [8] is compared with a puzzle (of WSs) as depicted in Fig. 1. In this puzzle, shape represents interfaces and color stands for data models/encoding.

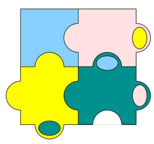

Fig. 1. Web services puzzle

Most model-driven methodologies for WIS design have modular implementations made from components driven by different models. Based on the Hera methodology, this paper proposes the use of WSs in realizing the different components and aggregating them in a WIS. In this way the software plug-and-play vision can be realized in the context of WISs.

2 Hera Methodology

Hera [6, 9] is a model-driven methodology for WIS design. It proposes a sequence of steps to be followed by WIS designers. Each step produces a model that captures a certain aspect of the application. The conceptual model (CM) gives the schema of the data to be presented. The application model (AM) specifies the way the user will be able to navigate through the data. The layout-specific issues are described in the presentation model. As the one-size-fits-all paradigm doesn't suffice to the WIS user demands, the user wants the presentation to adapt to its preferences and to the capabilities of the devices of their choice, Hera devises the Adaptation Model. Such a model will use information coming from the User Profile (static information about the user, i.e. information that doesn't change after the user starts browsing the presentation) in order to adapt the AM.

Hera has several other features that will be briefly mentioned here. It has the capability to integrate data coming from heterogeneous sources. Also, based on a User Model (representing dynamic information about the user, i.e. information that changes while the user browses the presentation) it is able to produce dynamic hypermedia presentations.

In order to experiment with the Hera methodology a WIS was built using the Hera CASE tool. This WIS presents data made available by the Rijksmuseum in Amsterdam. In this paper we will present only two of the Hera models, the CM and the AM, that will enable us to depict a WS-oriented Hera architecture. All the Hera models and their instances are represented in RDF(S) [10, 11]. The Hera tool uses RDF/XML for the serialization of the different models and XSLT [12] for the transformation between different models and their instances. It is able to produce HTML, WML, and SMIL presentations. Figure 2 gives a snapshot of the generated HTML presentation. It presents a Self Portrait painting of Vincent van Gogh.

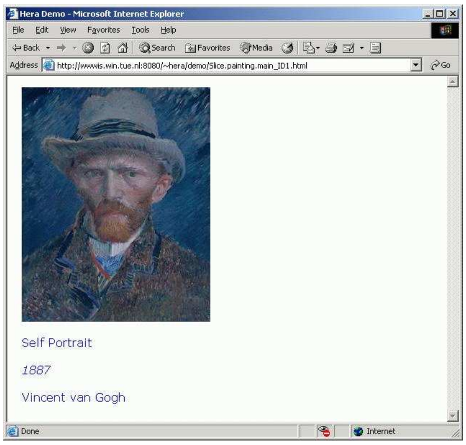

Fig. 2. Presentation in Internet Explorer

2.1 Conceptual Model

The conceptual model (CM) represents the schema of the data that needs to be presented. It is composed of concepts, concept relationships, and media types. There are two kinds of concept relationships: attributes which relate a concept to a particular media type and inter-concept relationships to express relations between concepts. A CM instance (CMI) gives the data that needs to be presented. This data might come from an integration/retrieval service that is out of the scope of this paper. Figure 3 illustrates an excerpt from a CMI. It describes the `Self Portrait` painting of Vincent van Gogh. Such a painting exemplifies the Impressionism painting technique identified by `Technique_ID2`. In the description one can find the year in which the painting was made, i.e. `1887`, and the URL of the Rijksmuseum image depicting it.

```
<Painting rdf:ID="Painting_ID6">
    <name>
        <type:String>
            <type:data>Self Portrait</type:data>
        </type:String>
     </name>
     <year>
        <type:Integer>
            <type:data>1887</type:data>
        </type:Integer>
     </year>
     <picture>
        <type:Image rdf:about="http://.../SK-A-3262.ORG.jpg"/>
     </picture>
     <exemplifies rdf:resource="#Technique_ID2"/>
     <painted_by rdf:resource="#Painter_ID2"/>
</Painting>
```

Fig. 3. Excerpt from CMI serialization in RDF/XML

2.2 Application Model

The application model (AM) gives an abstract view of the hypermedia presentation that needs to be generated over the CM data. It is composed of slice and slice relationships. A slice is a meaningful presentation unit that groups media items coming from possibly different CM concepts. The AM is in this way built on top of the CM. A slice that corresponds to a page from the presentation is called a top-level slice. There are two kinds of slice relationships: aggregation which enables the embedding of a slice in another slice, and navigation which specifies how a user can navigate between slices. Navigation relationships will be implemented in a hypermedia presentation by hyperlinks. In a multimedia presentation, navigation relationships can be implemented for example as time sequences. An AM instance (AMI) populates an AM with data from a CMI. Figure 4 depicts an excerpt from an AMI. It presents the slice `Slice.painting.main_ID6`, a top-level slice associated to the painting presented in the above subsection. This slice refers using slice aggregation relationships to four other slices. Each of the four slices contains a media item related to different painting/painter attributes.

```
<Slice.painting.main rdf:ID="Slice.painting.main_ID6">
    <slice-ref rdf:resource="#Slice.painting.name_ID6"/>
    <slice-ref rdf:resource="#Slice.painting.year_ID6"/>
    <slice-ref rdf:resource="#Slice.painting.picture_ID6"/>
    <slice-ref rdf:resource="#Slice.painter.name_ID2"/>
</Slice.painting.main>
<Slice.painting.name rdf:ID="Slice.painting.name_ID6">
    <media>
        <type:String>
            <type:data>Self Portrait</type:data>
        </type:String>
    </media>
</Slice.painting.name>
...
<Slice.painter.name rdf:ID="Slice.painter.name_ID2">
    <media>
        <type:String>
            <type:data>Vincent van Gogh</type:data>
        </type:String>
    </media>
</Slice.painter.name>
```

Fig. 4. Excerpt from AMI serialization in RDF/XML

Slices can have attached conditions based on attribute values coming from a user profile. A user profile stores attribute-value pairs for user preferences (e.g. user's level of expertise) and/or device capabilities (e.g. image capability of the display). A slice that has the condition not fulfilled will be removed from the AM. The same is valid for slice relationships that refer to such a slice.

3 Hera Web Service-Oriented Architecture

Hera has an implementation made from different components that together form one centralized application [9]. In this paper we present a distributed architecture for the Hera tool in which components are mapped to WSs. The loosely coupled Hera WSs realize the plug-and-play software vision in the context of WISs. For example, Hera can generate a WIS by composing a WS which provides up-to-date data, a WS that knows how to present this data, and a WS that is able to perform adaptation of the presentation based on user preferences/device capabilities.

We chose for a WS solution for realizing the distributed Hera architecture because WSs have clear advantages compared to their predecessors CORBA, J2EE, and DCOM [13]. First of all WSs are based on the XML document paradigm, a human readable language that abstracts from the implementation details. WS interfaces are specified in a universally accepted Web Service Definition Language (WSDL) [14], an XML-based language. Last but not least, Web services use the popular HTTP protocol as the carrier of exchanged messages.

Figure 5 presents a Web service-oriented architecture (WSOA) for Hera based on two WSs: the Data Service and the Presentation Service. The Data Service is responsible for delivering up-to-date data for which the Presentation Service will make a hypermedia presentation. A Client placed at a Web Server location will orchestrate the communication with the two services. The proposed WSOA has a star topology, with the Client in the middle. The communication between Client and services is done at SOAP [15] level which resides on top of HTTP while the communication between the Web Browser and the Web Server is done in plain HTTP.

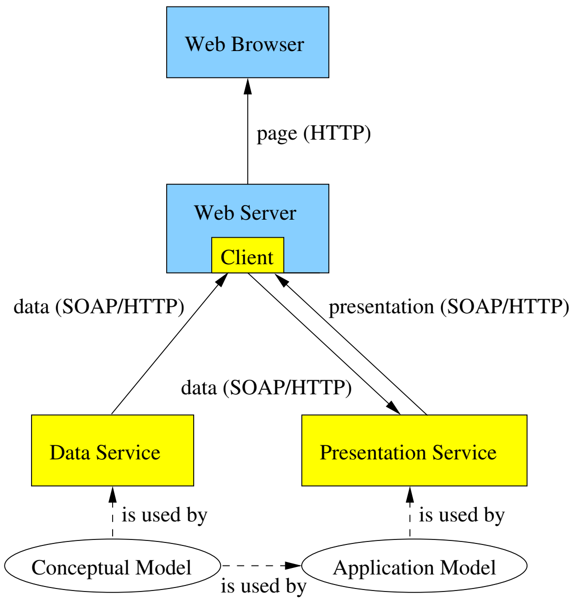

Fig. 5. Web service-oriented architecture

First the Client asks the Data Service to provide the data. Once the data is received it is passed to the Presentation Service. After receiving the data, the Presentation Service constructs a hypermedia presentation which is passed back to the Client. The Web Server that hosts the Client uses this presentation in providing pages to the Web Browser. The underlying assumption here is that the Data Service and the Application Service share the same CM.

Note that a service-based solution provides a lot of flexibility for such a system. Different Presentation Services (with different AMs) can be plugged into the system to produce different presentations for the same data. Moreover one Presentation Service may make an HTML presentation, while another one can provide a WML presentation, ensuring thus the ubiquity of the built WIS. Also different Data Services can be used in the same manner (assuming the fact that they agree with the Presentation Service on the CM). In this way data that comes from two different sources but sharing the same domain can benefit from presentation capabilities from the same Presentation Service.

Figure 6 describes the data transformations performed in the two services. The Data Service holds only the CMI and this is provided as output of the service. The Presentation Service gets as input a CMI and performs two pipelined XSLT transformations [9] to produce a presentation (in this particular service an HTML presentation, but other services can produce other types of presentations like WML, SMIL, cHTML etc.). The first transformation (`CMI2AMI`) populates an AM with data coming from the input CMI. This transformation is based on the fact that the AM is built on top of the CM as we pointed out in the previous subsection. The second transformation (`AMI2HTML`) computes the HTML pages that correspond to top-level slices. These HTML pages represent also the output of the Presentation Service.

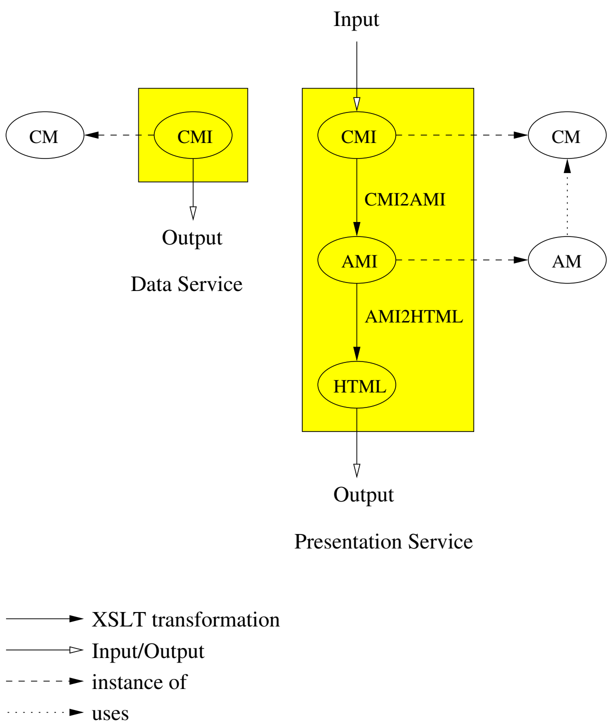

Fig. 6. Data Service/Presentation Service

3.1 Web Services Descriptions

The interface of a WS is given in a Web Service Description Language (WSDL) specification. In addition to the service interface, such specifications give also the data types used by the service messages and the location of the service. In this subsection we will describe only the interface as we only use existing XML Schema Datatypes [16, 17] (we did not need to define our own types) and the service location can be defined anywhere on the Web.

Figure 7 depicts an excerpt from the Data Service WSDL specification. First it specifies which are the messages, their embeddings, and the type of data that messages will carry. The `getDataRequest` message is an empty message. This message is used just to trigger the response from the service. The `getDataResponse` message has a `<wsdl:part>` containing the requested data string. The `<wsdl:portType>` associates the request and response messages with the `getData` operation of the Data Service. It also specifies the type of the message as input or as output for the operation. The data string returned is the CMI that the service holds.

```
<wsdl:message name="getDataRequest">
</wsdl:message>
<wsdl:message name="getDataResponse">
     <wsdl:part name="getDataReturn"
                type="xsd:string"/>
</wsdl:message>
<wsdl:portType name="DataService">
     <wsdl:operation name="getData">
          <wsdl:input message="impl:getDataRequest"
                      name="getDataRequest"/>
          <wsdl:output message="impl:getDataResponse"
                       name="getDataResponse"/>
     </wsdl:operation>
</wsdl:portType>
```

Fig. 7. Excerpt from Data Service WSDL

Figure 8 depicts an excerpt from the Presentation Service WSDL specification. The request message `getPresentationRequest` has a `<wsd:part>` named `in0` (this is the default naming convention used by our SOAP server for the operation arguments) containing one string. The response message `getPresentationResponse` will contain also a string. The `<wsdl:portType>` associates the request and response messages with the `getPresentation` operation of the Presentation Service. As for the Data Service, it also specifies the type of the message as input or as output for the operation. The input data string is the CMI and the data string returned from the operation is the encoded (in one string) presentation.

```
<wsdl:message name="getPresentationRequest">
    <wsdl:part name="in0"
               type="xsd:string"/>
</wsdl:message>
<wsdl:message name="getPresentationResponse">
    <wsdl:part name="getPresentationReturn"
               type="xsd:string"/>
</wsdl:message>
<wsdl:portType name="PresentationService">
    <wsdl:operation name="getPresentation"
                    parameterOrder="in0">
        <wsdl:input message="impl:getPresentationRequest"
                    name="getPresentationRequest"/>
        <wsdl:output message="impl:getPresentationResponse"
                     name="getPresentationResponse"/>
    </wsdl:operation>
</wsdl:portType>
```

Fig. 8. Excerpt from Presentation Service WSDL

3.2 SOAP messages

After we have defined the service interface we can have now a closer look at the actual representation of the service messages. Despite its name the Simple Object Access Protocol (SOAP) is not a classic (communication) protocol. It is rather a one-way message exchange paradigm (or some others prefer to say a lightweight protocol to exchange information in a distributed system). A SOAP message is an XML message containing a SOAP envelope. A SOAP envelope has an optional SOAP header and a required SOAP body. It is the SOAP body that will actually contain the data carried in a message. The current implementation uses SOAP RPC which means that all message communication is done synchronously.

Figure 9 presents a snapshot of Client-services communication, namely the SOAP messages exchanged between the Client and the Data Service. The `SOAP Request` window displays the `getDataRequest message`, an empty message as we already saw from the interface description. The `SOAP Response` window shows the `getDataResponse` message containing in its `getDataReturn` part an actual CMI. Note that <, > are escaped as we encoded the data as a string.

3.3 Tools

In order to experiment with the proposed architecture a Java-based Hera tool was developed. Tomcat 4.1 [18] was used as the Web server that supports servlets. On this Web server we installed Axis 1.1 (Apache eXtensible Interaction System) [19], a SOAP 1.1 engine. By SOAP engine we mean a tool that supports both a SOAP server and SOAP clients. We did deploy on the SOAP server two services DataService and PresentationService. For their deployment we used appropriate Axis Web Service Deployment

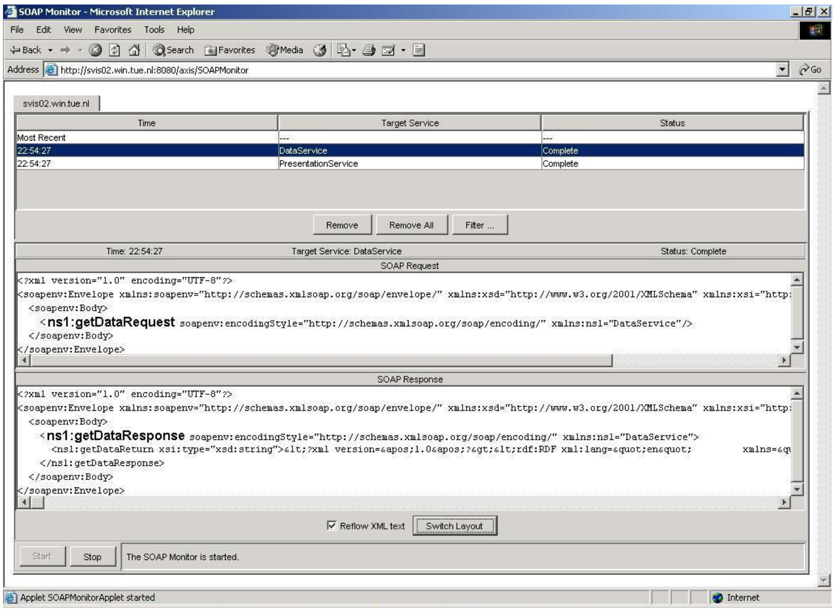

Fig. 9. SOAP messages

Descriptors. The SOAP Client that communicates with these services was installed on the Web server, outside the SOAP server. The WSDL specifications were generated by the Java2WSDL emitter. Both Java2WSDL and the SOAP Monitor are part of the Axis distribution kit. Tomcat and Axis are Java-based and freely available from the Apache Software Foundation. The services and the client were written in Java. The Saxon [20] implementation was used as the XSLT processor. All software is running on the Java 1.4 platform.

It is important to notice that when developing WSs with Axis, the programmer doesn't need to bother about making WSDL interfaces or the actual encoding of the SOAP messages. All these will be automatically done by the system. Making all WS details transparent to the programmer enables him to focus only on the application logic implementation in Java and makes thus the system less error-prone.

3.4 Adaptation in Hera Web Service-Oriented Architecture

In previous work [9] we had extended the Hera methodology to consider presentation adaptation (at AM level) based on a user profile. Figure 10 presents a WSOA based on four services: Data Service, Presentation Service, Profile Service, and Adaptation Service. In order to denote the order in which the messages will be passed we added a label to the continuous arrows. This label should be read in the increasing number order or alphabetical order. Both sequences (1, 2, 3) and (a, b) can be done in parallel. Step 4 is performed after completion of steps 3 and b.

The steps 1, 2, and 5 were already discussed in the beginning of this section. In step a the nProfile ('n' stands for new) provided by the Client is sent to the Profile Service. The Profile Service can be viewed as a shared memory service for user profiles. By a shared memory service we simulate shared memory between services using one service. The profile attributes not captured in the nProfile will be taken from the oProfile ('o' stands for old), the old profile of the user, and result in a merged Profile. The data management (e.g. updates on the oProfile) of the Profile Service is not presented here. The Profile is sent to the Adaptation Service. Also, the Adaptation Service receives the AM from the Presentation Service. After receiving these two messages the Adaptation Service computes the aAM ('a' stands for adapted) and sends it to the Presentation Service. In this WSOA the Presentation Service will use the aAM, instead of the AM, to compute the presentation.

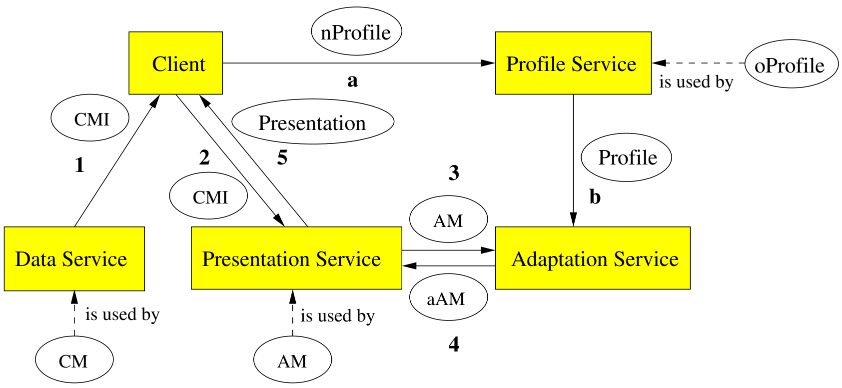

Fig. 10. Extended Web service-oriented architecture

4 Conclusion

In this paper we have described how we have implemented a distributed WIS architecture based on the Hera methodology. We chose for a WS-oriented solution due to the popularity and easy-to-implement features of WSs. In this way WISs can be seamlessly built by composing appropriate WSs. The Axis distribution kit proved to be a very flexible set of tools to support WS development, deployment, and monitoring. We have also shown examples of WSDL specifications used to describe WSs and of SOAP messages exchanged with WSs.

As future work we would like to expand the Hera Web service-oriented architecture/tool with new services like a data query service, a data integration service, or a service able to generate dynamic hypermedia presentations.

References

1. Isakowitz, T., Bieber, M., Vitali, F.: Web information systems. Communications of the ACM **41** (1998) 78–80
2. Schwabe, D., Rossi, G.: An object oriented approach to web-based application design. Theory and Practice of Object Systems **4** (1998) 207–225
3. Ceri, S., Fraternali, P., Bongio, A., Brambilla, M., Comai, S., Matera, M.: Designing Data-Intensive Web Applications. Morgan Kaufmann (2003)
4. Koch, N., Kraus, A., Hennicker, R.: The authoring process of the uml-based web engineering approach. In: First International Workshop on Web-Oriented Software Technology. (2001)
5. Schewe, K.D., Thalheim, B.: Reasoning about web information systems using story algebras. In: Advances in Databases and Information Systems (ADBIS 2004). Volume 3255 of Lecture Notes in Computer Science., Springer (2004) 54–66
6. Frasincar, F., Houben, G.J., Vdovjak, R.: Specification framework for engineering adaptive web applications. In: The Eleventh International World Wide Web Conference, Web Engineering Track. (2002) `http://www2002.org/CDROM/alternate/682/`.
7. Foster, I., Kesselman, C., Nick, J.M., Tueke, S.: The physiology of the grid: An open grid services architecture for distributed systems integration. Global Grid Forum (2002)
8. Nativi, S.: Service-oriented technology to support geosciences. In: Expanding Horizons: Using Environmental Data for Education, Research, and Decision Making Workshop. (2003) `http://my.unidata.ucar.edu/content/Presentations/2003_expanding_horizons/nativioverview.pdf`.
9. Vdovjak, R., Frasincar, F., Houben, G.J., Barna, P.: Engineering semantic web information systems in hera. Journal of Web Engineering **2** (2003) 3–26
10. Klyne, G., Carroll, J.J.: Resource description framework (rdf): Concepts and abstract syntax. W3C Recommendation 10 February 2004 (2004) `http://www.w3.org/TR/rdf-concepts/`.
11. Brickley, D., Guha, R.: Rdf vocabulary description language 1.0: Rdf schema. W3C Recommendation 10 February 2004 (2004) `http://www.w3.org/TR/rdf-schema/`.
12. Kay, M.: Xsl transformations (xslt) version 2.0. W3C Candidate Recommendation 3 November 2005 (2005) `http://www.w3.org/TR/xslt20/`.
13. O'Toole, A.: Web service-oriented architecture: The best solution to business integration. Cape Clear Software (2005) `http://www.capeclear.com/technology/clearthinking/websoa.shtml`.
14. Christensen, E., Curbera, F., Meredith, G., Weerawarana, S.: Web services description language (wsdl) 1.1. (W3C Note 15 March 2001)
15. Box, D., Ehnebuske, D., Kakivaya, G., Layman, A., Mendelsohn, N., Nielsen, H.F., Thatte, S., Winer, D.: Simple object access protocol (soap) 1.1. (W3C Note 08 May 2000)
16. Thompson, H.S., Beech, D., Maloney, M., Mendelsohn, N.: Xml schema part 1: Structures. (W3C Recommendation 02 May 2001)
17. Biron, P.V., Malhotra, A.: Xml schema part 2: Datatypes. (W3C Recommendation 02 May 2001)
18. Apache Software Foundation: Apache tomcat (2006) `http://jakarta.apache.org/tomcat/`.
19. Apache Software Foundation: Webservices - axis (2006) `http://ws.apache.org/axis/java/user-guide.html`.
20. Kay, M.: Saxon (the xslt and xquery processor) (2006) `http://saxon.sourceforge.net`.