18th World IMACS / MODSIM Congress, Cairns, Australia 13-17 July 2009
http://mssanz.org.au/modsim09

Service-oriented Architecture for Environmental Modelling – The Case of a Distributed Dike Breach Information System

Theisselmann, F.[1, 2], D. Dransch[2] and S. Haubrock[1]

[1] Humboldt-Universität zu Berlin, Department of Computer Science, Systems Analysis
[2]Helmholtz Centre Potsdam GFZ German Research Centre for Geosciences
Email: theissel@informatik-hu-berlin.de

Abstract: An important aim of modelling complex environmental systems is to embed scientific outcomes into applications. With this characteristic, not only the modelled systems, but also the required technical realizations are complex. Managing interdisciplinary, interorganisational modelling tasks and dealing with complex heterogeneous technologies are challenges the modellers and respective institutions are faced with. In practice, issues arise due to technological heterogeneity across organisational and disciplinary boundaries, model reuse within different application contexts, and integrative system specification.

Service-oriented Architecture (SOA) with Web Services and Workflow Management provide a conceptual framework and a technological basis to deal with these problems. As part of a web-based disaster information platform established in Germany, we implemented the SOA-based prototypical distributed information system *NaDiNe-Dike*. It combines a numerical dike breach simulation model, spatial data handling, simulation management, and a web-based user interface. All functionalities of the system are provided by Web Services. The execution of these services is managed by a Workflow Management System. In this paper, we describe how various existing technologies, including heritage simulation and other off-the-shelf software (i.e. OGC-WS), have been integrated using SOA. This approach is particularly flexible by supporting the reuse and recombination of functionalities (e.g. simulation models) in different contexts and applications.

We discuss the main characteristics of the implemented system and conclude that the main advantages using SOA in such a way are the possibility to distribute functionality of a system spatially and functionally across institutions, programming languages and operating systems. Moreover, SOA provides enough flexibility to integrate simulation models with data management and presentation technologies that originate in different technological spaces and may be provided and maintained by different spatially distributed organizations. The possibility to reuse software components helps to reduce redundant implementation efforts and eases the transfer from science to the application domain. The main disadvantages of the approach followed here are the use of the network as the coupling media, the need to redesign heritage software, and the lack of specific simulation support. Moreover, the introduction of the component-based software implementation paradigm into the participating organizations is required, which must be accompanied by adequate system specification techniques. However, SOA provides the technological basis for a software infrastructure that reflects the distributed nature of today's modelling tasks, enhancing the division of responsibilities, thus, making modelling more efficient in the long run.

Keywords: *Service-oriented Architecture, Web Service, Workflow Management System, Dike breach model, Distributed information system*

1. INTRODUCTION

A challenge for environmental modellers is to provide simulation models that are not only useful in the scientific context, but which may also serve planning and communication purposes. Argent (2004) divides the lifecycle of simulation models into four levels. On level I, a model is initially developed by the modeller for a specific research purpose. The user community of this model is a small group of experts. The user group grows as the model is applied to other sites and refined by experts (level II). As a model is used in operational planning, the coupling of the model to other models is more important and the user group of models becomes more diverse (level III). Finally, a model may serve planning and policy analysis purposes on level IV, where the model may be used by users who are unaware of the internal working of the model.

A particular challenge is to technically support this lifecycle of a simulation model, since the technical and organizational requirements change throughout the lifecycle. Traditionally, simulation models are implemented using general purpose programming languages, such as C/C++ and FORTRAN. The integration of such legacy models with other models and technologies is often difficult, because they are often built using outdated practices (Argent, 2004). There is a variety of environmental modelling frameworks available. Although these frameworks provide valuable functionality, their specificity is high: often the use of a particular implementation language is required and the inclusion of tailored external logic, e.g. GIS functionality, user interfacing, and data management, is difficult. Thus, there is a high specificity resulting in high transition costs throughout the levels of a model's lifecycle.

There exist flexible technologies that are designed to minimize such efforts by allowing the integration of software across tool, language and platform boundaries to facilitate collaborative, spatially distributed modelling and computing, e.g. CORBA[1], J2EE[2], .NET[3], and HLA (Dahmann et al. 1997). A basic approach to structure such software systems is decomposition. Decomposition can be applied to system modelling (e.g. through modularization or object-orientation), as well as to implementation and execution (e.g. by means of distributed computing).

In this work we focus on three aspects of environmental modelling and simulation as the base for decomposition: modelling, data management, and information presentation. Modelling is the description of a simulation model in a way that it can be executed. Data management is the documentation and archiving of data related to the model and model runs (e.g. input and output datasets). Information presentation is the presentation of the model itself and the datasets and information within them. One characteristic of these aspects is that their importance may vary throughout the lifecycle of a model. Hence, the requirements for adequate implementations may change. We propose a functional decomposition based on these aspects, which aims at efficiently supporting level I to level IV modelling with different technologies that reflect changing requirements. This decomposition also reflects the fact that there is existing software related to these aspects that can be reused in an information system.

We implemented the prototypical level IV distributed information system *NaDiNe-Dike* that includes a dike breach simulation model. NaDiNe-Dike is based on Service-oriented Architecture (SOA), which provides an infrastructure for which there exist off-the-shelf implementations of required functionality, according to the proposed functional decomposition (see chapter 2). NaDiNe-Dike combines a numerical dike breach simulation model, GIS functionality to store spatial data and generate cartographic output, data management, and user interaction, in order to supply useful information to research and disaster management. In chapter 2, we present some important aspects of SOA. In chapter 3, the prototypical information system NaDiNe-Dike demonstrates how the simulation model can be used throughout the different levels.

2. SERVICE-ORIENTED ARCHITECTURE AND BUSINESS PROCESSES

Service-oriented architecture (see Weerawarana et al. 2005) describes a software infrastructure in which the main functionalities of an application are organized as services. The services can be arbitrarily distributed and they can be dynamically combined to form business processes. A business process is a set of ordered activities that are performed in order to achieve a specific goal. Activities within a business process can be performed automatically (i.e. by Web Services) or by persons. The automation of a business process is a workflow (see Aalst, 2002).

---

[1] Common Object Request Broker Architecture. URL: http://www.corba.org/

[2] Java Platform Enterprise Edition. URL: http://java.sun.com/javaee/

[3] .NET. URL: http://www.microsoft.com/net/

A Web Service is software that implements functionality, which can be accessed by other software applications via a computing network. The way a Web Service can be accessed is well defined, ensuring interoperability across platforms and frameworks (W3C, 2002). The functionality of a Web Service may encompass data sinks (i.e. data storage), user interaction, data sources (i.e. models), and spatial data handling (i.e. map creation).

A workflow may be described using designated languages, which can be executed by Workflow Management Systems. Workflow Management Systems enable the formulation and execution of workflows (Aalst, 2002). At runtime, the execution of a workflow is realized by sending messages between the workflow management system and participating Web Service instances.

SOA can be implemented using various technologies. In this work, we used a well established SOA technology stack that contains a language to describe Web Services (WSDL), a protocol to exchange messages between Web Services (SOAP), and a language to define automated workflows (BPEL, see Weerawarana et al. 2005).

For automated processing, Web Services are described using the Web Services Description Language (WSDL), which is an XML format for describing services (Christensen et al. 2001). With WSDL it is possible to describe the messages that can be processed by a service and how and where a service can be accessed. WSDL can be combined with the SOAP protocol to structure the message exchange between Web Services. Each SOAP message is an XML document that contains an envelope with a header, and a body. The payload of the message (the data to be transmitted) is inside the body. Information inside the header can be used for message routing (Weerawarana et al., 2005, Gudgin et al., 2003).

In our implementation, workflows are defined using the Business Process Execution Language (BPEL). BPEL is a workflow based language that composes services by choreographing interactions of services (Weerawarana et al., 2005). BPEL makes use of WSDL descriptions of services and messages. It provides the means to process SOAP messages by using the query language XPath[4]. This can be used to control the workflow and to transform messages. Moreover, BPEL provides error handling in case unforeseen errors occur, for example when a Web Service fails (OASIS, 2007).

Domain specific standards for Web Services have been developed by the Open Geospatial Consortium[5] (OGC). The OGC web service (OGC-WS) standards describe Web Services related to geographic information and spatial data processing. OGC-WS provide functionalities on spatial data, such as storage and retrieval (Web Feature Service, Web Coverage Service), or the generation of cartographic representations (Web Map Service).

3. NADINE-DIKE

NaDiNe-Dike has been implemented within the Natural Disasters Networking Platform (NaDiNe, Haubrock et al., 2006). The goal of NaDiNe-Dike is to support managers and scientist with information about possible dike breaches within a modelled area. The dike breach information is generated by a simulation model (for more information on the model see Vorogushyn et al., 2006).

3.1. The NaDiNe-Dike Workflow

Figure 1 shows the workflow of NaDiNe-Dike including the tasks to be performed. The system provides different scenarios to experts, who access the system via a web-based interface (*Gather scenario*). The expert chooses a scenario, which is related to a specific parameter set for the dike breach model. The scenario is calculated by the system, if it has not been calculated before. The numerical result is stored persistently and instantaneously made accessible via the Internet. After that, a map showing the locations of the predicted dike breaches is generated and an HTML[6] document containing the map is generated and made accessible by a web server (*Generate user info*). Finally, an email with all access information for the visual and numerical results is sent to the user. In addition, the metadata about the model run is stored (*Update metadata*). If an error occurs during the execution of the workflow, an email is sent to the user explaining the error and the workflow is stopped.

[4] XPath: http://www.w3.org/TR/xpath20/

[5] Open Geospatial Consortium: http://www.opengeospatial.org/

[6] Hypertext Markup Language

NaDiNe-Dike is functionally decomposed according to simulation modelling, data management, and presentation (see chapter 1). The different functionalities are provided by different Web Services (see chapter 3.2).

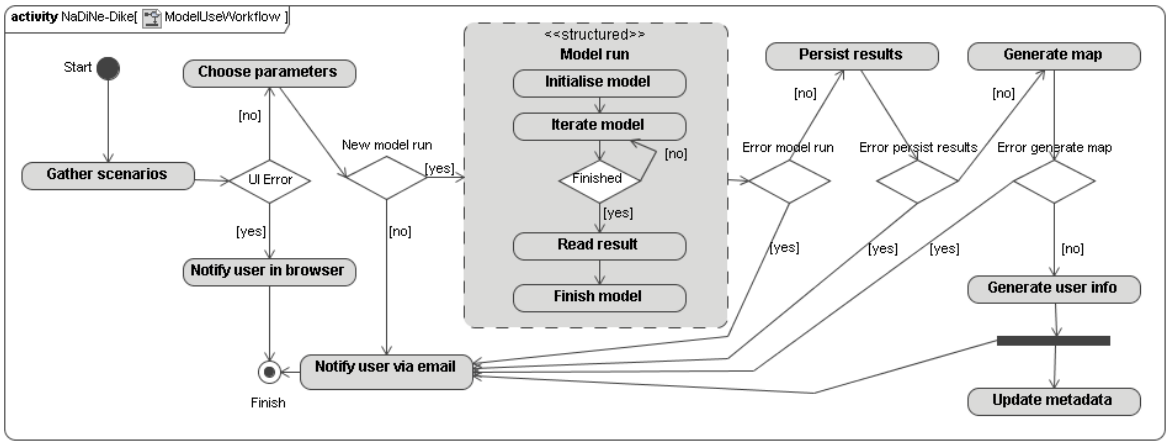

Figure 1. The NaDiNe-Dike workflow (UML).

3.2. The NaDiNe-Dike Components

As described above, the system functionalities are provided by Web Services (see figure 2) that execute the tasks of the workflow (figure 1). The choice of Web Services reflects the existence of available software (i.e. model software and OGC Web Services) that is aligned with the proposed functional decomposition. The simulation functionality is a numerical hydraulic and dike breach simulation (*modelService*). Data management consists of the persistent storage and web-based retrieval of model parameters (*inputService*), as well as numerical and map output (*resultService*, *WebFeatureService*). Presentation encompasses a user interface, map generation, and email communication (*userInterface*, *WebMapService*, *mailService*).

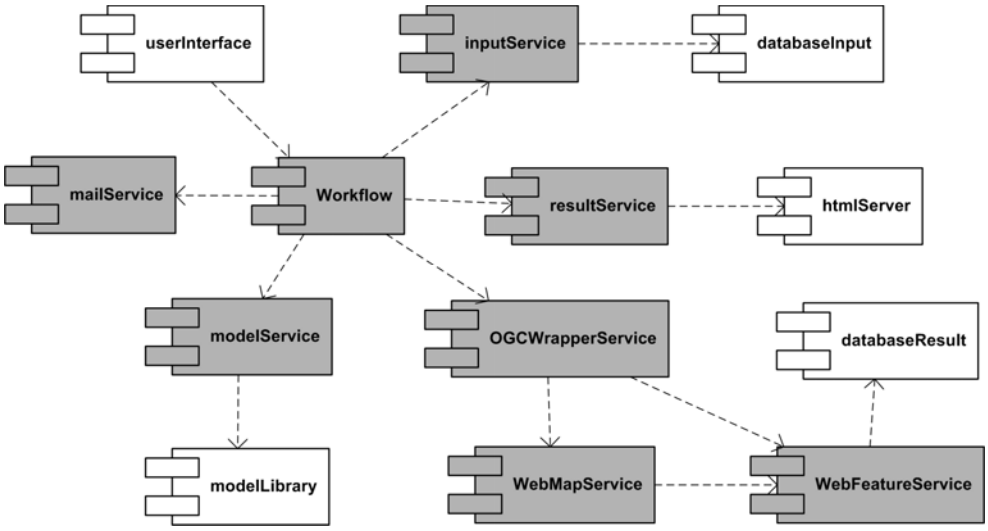

Figure 2. The architecture of NaDiNe-Dike (UML).

Figure 2 shows the Web Services (grey) and auxiliary components (white) of the system: The *userInterface* provides access to the system by facilitating the communication between the user and the workflow. It can be accessed via a standard web browser. The *inputService* administrates the model parameters and time series data that represent different scenarios. The time series data is the input for the dike breach model. The *inputService* accesses a relational database, in which the input data resides (*databaseInput*). The *resultService* generates HTML results and makes them accessible through a web server.

The *WebFeatureService* is an OGC compliant Web Service that provides functionality to store and retrieve the geographic result data using a spatial database (*databaseResults*). The *WebMapService* generates map output based on data, which it gathers directly from the *WebFeatureService*. The *WebMapService* and the *WebFeatureService* are based on an implementation of deegree[7] WMS and deegree WFS. The *OGC-WrapperService* is a proxy Web Service component that transforms the numerical output of the model into an OGC web feature service compliant format and provides SOAP access to the OGC compliant Web Services, thus, it acts as a "wrapper" for them.

The dike breach simulation model is provided by the *modelService*. The model service makes use of a shared library that provides the model functionality (*modelLibary,* see chapter 3.3). The *mailService* sends emails containing the result or error messages to the user. It uses an SMTP mail server. The *workflow* is a Web Service that executes the NaDiNe-Dike workflow. It is implemented as a BPEL workflow that is executed by the activeBPEL[8] workflow engine.

3.3. The Dike Breach Model Service

With the dike breach model, legacy model software has been integrated into NaDiNe-Dike. For integration, the original model software had to be redesigned, because it was a monolithic standalone application, written in FORTRAN. For this, the process behaviour has been extracted from the code and an interface has been defined that enables the execution. Finally, the model is realized as a shared library that provides an implementation of the interface.

A model run is divided into three phases. In the first phase (*initialisation*), the input parameters are read and the initialization status of the model is calculated. In the second phase (*iteration*), the model iteratively calculates the model status for each time step up to the final time and the model state changes. In order to finish a model run, the memory has to be cleared and the model must be set to a status where a new initialization may occur ("*stand-by*"-phase).

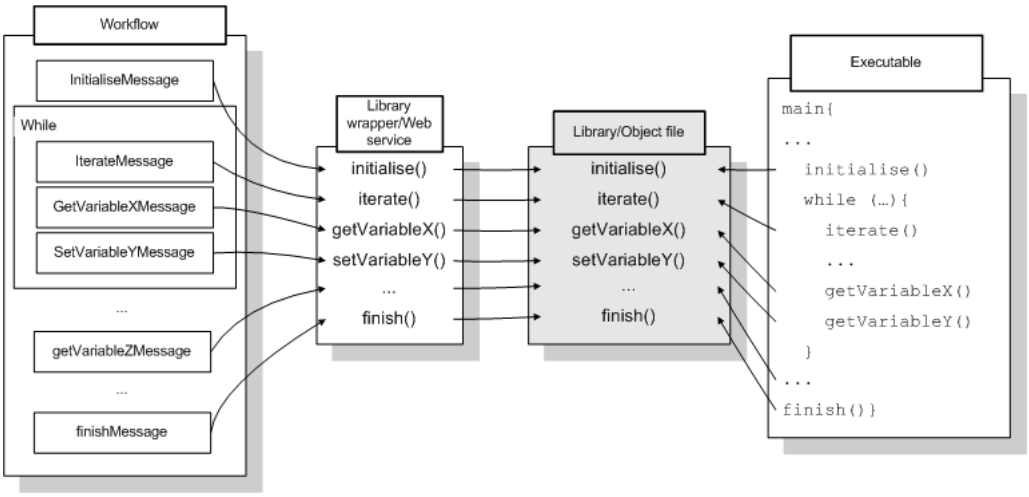

Figure 3. The implementation of interface functions enables the use of the same model (grey box) during early development (right) and within NaDiNe-Dike (right).

In accordance to these phases, the interface of the *modelLibrary* has three operations, which enable the execution of a model run: *initialise(), iterate(),* and *finish().* In addition, *get()* and *set()* methods have been defined in order to retrieve model output and to be able to couple other models in the future (e.g. an inundation model). Get() methods (e.g. *getVariableX()*) read data from the model and set() methods write data to the model. The process behaviour information must be specified outside the library, e.g. in an executable or a workflow. Figure 3 shows the pattern for this interface and its application in NaDiNe-Dike and preceding development.

[7] Deegree. URL: http://www.deegree.org/

[8] activeBPEL: http://www.activevos.com/community-open-source.php

This simple interface facilitates the usage of the same model at all levels of the model lifecycle. During the development of the model, a C executable was used to run and test the model locally. At this level, data management was simply realized by filesystem-I/O with the means of built in I/O operations. Data presentation and dataset management had to be realized manually, i.e. with the import of data into a Geographic Information System (GIS). The interface functions of the library are directly referenced by name from within the main()-function of the simulation program (see figure 3, right).

The same model code was used to provide the model functionality in the level IV application NaDiNe-Dike. For this, the model library has been wrapped into a Web Service, implemented with the Java programming language, using the Java Native Interface[9] (JNI) for integrating the native model component with the Java Web Service. The Web Service serves two purposes: 1) sending and receiving SOAP messages and 2) calling the model functions of the library. The model Web Service is used by sending SOAP messages to it. Figure 3 (left) illustrates the sequence of messages, which are sent by the workflow engine. First, there is a message for initialization (*InitialiseMessage*). After that, messages for iteration (*IterateMessage*) and messages for data exchange (i.e. *GetVariableXMessage*) are repeated until a *finishMessage()* ends the model run.

```
<?xml version="1.0" encoding="utf-8"?>
   <soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <soapenv:Body>
         <isInitialisedResponse xmlns="http://soapinterop.org">
            <isInitialisedReturn>
               <message>Model is not initialised</message>
               <errCode>1</errCode>
            </isInitialisedReturn>
         </isInitialisedResponse>
      </soapenv:Body>
   </soapenv:Envelope>
```

Figure 4. A SOAP message with an envelope and a body with its payload (bold).

In NaDiNe-Dike, each message is answered by the Web Services and processed by the workflow engine in order to control the workflow. Figure 4 shows a SOAP message that was sent from the *modelService* to the workflow, after an initialization attempt that was not successful. A high structural content (XML tags) compared to the actual payload (bold) can be observed, which is caused by the use of SOAP.

4. DISCUSSION

SOA provides accepted standards and off-the-shelf technology, which can be applied to scientific modelling. With OGC web services (WFS, WMS), for example, it was possible to integrate standardised, off-the-shelf GIS functionality efficiently. However, with the usage of technologies like SOA, there is a need to involve and integrate technical experts in the development process. Software engineering provides techniques to deal with complex development processes, but they require specific skills (i.e. UML modelling, WSDL), which, at least partly, have to be learnt by domain experts in order to make use of the enabling methods and tools.

We integrated legacy software across implementation language boundaries. We proposed an approach to develop and use simulation model software, which is similar to the widely used modelling practice using general purpose programming languages like FORTRAN or C/C++. Using a simple interface allows the usage of the same model code throughout the lifecycle of a model. However, although model implementation languages may vary, this approach may require a shift from implementing models as monolithic tools to implementing reusable components with an interface and adequate documentation. This change may require time, labour, and training.

Distributed Web services may be developed, maintained, and updated with the domain experts. This provides the possibility of rapid integration of model software updates and latest research findings, so that all users benefit from it. However, the distribution of functionalities in a network implies network transport of data. Networks have limited bandwidth and reliability, so the network capacity has to be taken into account when designing applications and services. Heavy data exchange may significantly affect the performance of an application, compared with non SOA implementations. Particularly the use of SOAP may cause network traffic overhead.

[9] Java Native Interface: URL: http://java.sun.com/javase/6/docs/technotes/guides/jni/index.html

As a generic technology, SOA does not provide specific simulation support, such as synchronization or model templates. However, in NaDiNe-Dike, this was not an issue, since the process behaviour of this time-driven simulation model was easy to emulate by means of workflow modelling.

5. CONCLUSION AND OUTLOOK

We conclude that using SOA with Web Services and workflow management is reasonable, when there is a need to combine software and expertise from different domains and institutions in an information system, if the network provides enough bandwidth. In order to realize more complex scenarios with coupled synchronized models, simulation functionality must be implemented using workflow modelling. Most exchanged data within the system passes through the workflow engine and is processed by it. This has to be taken into account in scenarios with heavy data exchange, since it affects performance.

In order to minimize transition costs throughout the lifecycle of models, a functional decomposition should take place from the very beginning of the model implementation, since redesign is costly. The division of functionality in modelling, data management, and presentation functionality is aligned with existing technical spaces and allows the use of existing off the shelf technologies that are appropriate for different applications (i.e. OGC-WS, databases).

By encapsulating domain knowledge in separate Web Services, a division of labour can be realized within the development process, not only between different sciences, but also between IT experts and scientists. Reusing model software via Web Services is a straightforward way to simplify the transfer from science to management, although simply combining interfaces does not ensure the sensible use of models and must be accompanied with an appropriate development and communication process.

With the approach specified in this paper, a technological framework has been applied that allows the integration of scientific results in information or decision support systems. In particular, the user-friendly access to scientific simulation models supports the efficient transfer of reliable information to a broader audience.

ACKNOWLEDGEMENT

The results presented in this paper have been developed at the German Research Centre for Geosciences by the working group Geoinformation Management and Visualisation in co-operation with Section 5.4 Engineering Hydrology in the framework of the Helmholtz project Natural Disasters Networking Platform (NaDiNe). Sincere thanks are given to the anonymous reviewers, whose useful comments helped to improve this paper.

REFERENCES

Aalst, W. v. d., (2002), Workow Management. MIT Press Cambridge.

Argent, R.M., (2004), An overview of model integration for environmental applications - components, frameworks and semantics. *Environmental Modelling & Software 19*, 219-234.

Christensen, E., F. Curbera, G. Meredith, S. Weerawarana, (2001), Web Services Description Language (WSDL) 1.1 W3C note. Published on the World Wide Web.

Dahmann, J. S., R.M. Fujimoto, R.M. Weatherly, (1997), The Department of Defense High Level Architecture, *Proceedings of the 29th conference on Winter simulation, IEEE Computer Society*, 142-149

Gudgin, M., M. Hadley, N. Mendelsohn, J.-J. Moreau, F. Nielsen, A. Karmarkar, Y. Lafon, (2003), SOAP version 1.2 part 1: Messaging framework W3C recommendation. Published on the World Wide Web.

Haubrock, S., D. Dransch, H. Kreibich, B. Merz, S. Plattner, S. Voigt, H. Mehl, (2006), Natural Disasters Networking Platform (NaDiNe). In: Amman, W., J. Haig, C. M. S. Huovinen, (Eds.), *Proceedings of the International Disaster Reduction Conference. Vol. 1*. Swiss Federal Research Institute WSL, 74.

OASIS, (2007), Web Services Business Process Execution Language Version 2.0, OASIS Standard, 11 April 2007, Published on the World Wide Web, URL: http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf

Vorogushyn, S., H. Apel, K.-E. Lindenschmidt, B. Merz, (2006), Coupling 1D hydrodynamic, dike breach and inundation models for large-scale flood risk assessment along the Elbe River, *Geophysical Research Abstracts*, 8, 10565, EGU General Assembly, Wien, April 2006.

W3C, (2002), Web Services Activity Statement. Published on the World WideWeb. URL http://www.w3.org/2002/ws/Activity

Weerawarana, S., F. Curbera, F. Leymann, T. Storey, D. F. Ferguson, (2005), Web Services Platform Architecture. Prentice Hall.