Proceedings of the TMCE 2004, April 13–17, 2004, Lausanne, Switzerland, Horváth & Xirouchakis (eds.)

WEB-BASED SIMULATION: APPLICATION SCENARIOS AND REALIZATION ALTERNATIVES

Sven Meyer zu Eissen
Department of Computer Science
Paderborn University
smze@upb.de

Benno Stein
Department of Computer Science
Paderborn University
stein@upb.de

ABSTRACT

Web-based simulation is an ambiguous term that is used for various applications and with different meanings. Ernest Page identifies the following five research and development areas: simulation as hypermedia, simulation research methodology, Web-based access to simulation programs, distributed modeling and simulation, and simulation of the WWW (Page, 1998). In this paper, the term Web-based simulation relates to the first three areas.

The paper is organized as follows. The first section outlines the potential of Web-based services that can be built upon a remote simulation engine. Our main contribution, however, relates to software engineering: In Section 2 we compare different realization concepts for a Web-based simulation service and discuss the impacts with respect to the outlined application scenarios. In Section 3 we then introduce a prototype of a simulation Web service that realizes the analysis and execution of models defined in the well-known Modelica modeling language.

KEYWORDS

Web-based Simulation, Web Service, SOAP, Modelica, non-distributed Simulation

1. APPLICATION SCENARIOS AND RATIONALE OF WEB-BASED SIMULATION SERVICES

Web-based simulation is often associated with distributed simulation and with multi-user simulation (Kilgore, 2002). Moreover, the simulation applications and examples that can be found in the respective conferences relate to discrete event system simulation (DESS) in the first place (Yücesan et al., 2002).

In this paper we focus on non-distributed, single-user simulation tasks. I. e., at the client side, a user can formulate a model in a high-level modeling language such as VHDL-AMS or Modelica (Elmqvist et al., 1999). In particular, model formulation and experiment definition shall enable the description of multi-disciplinary systems and allow object-oriented model composition, non-causal modeling, mixed discrete-event/continuous-time relations, and the reuse of existing model libraries. At the server side, which is connected to the client via the Internet, there is a set of tools for model syntax analysis, experiment execution (i. e., model simulation under the desired user constraints), textual and graphical result preparation, model hosting, sharing, and distribution etc.

Note that a number of apparent as well as future scenarios become possible if the aforementioned tools are operationalized in the form of Web-based services (see also (Fishwick, 1997)). The following list gives interesting examples.

- Web-based simulation and development tools for fast model building and quick and easy experimentation will be available at each Internet access point and without cumbersome installation. Of course, such a service may allow the easy integration of a client's model libraries.
- Instead of porting or reimplementing approved simulation technology, the provision of an existing simulation environment in the form of Web-based services will directly address legacy aspects such as cross-platform usability.
- New license models for simulation software become possible. This is useful for individuals

and small companies where simulation capabilities are only rarely needed.

- Particular high-level simulation services can be set up, which focus on a special domain or task and which provide domain-specific engineering know-how for model optimization or for failure effects and mode analysis (FEMA).
- Simulation services can be realized that play the role of third party analyses and referee evaluations.
- High-level simulation services open new possibilities for education and training. This relates to the availability and distribution through the World Wide Web as well as to hypermedia concepts, since simulation capabilities can be integrated seamlessly in course material and combined with text, audio, and video.

Observe that the mentioned scenarios share the same service structure: A single user works in a well-defined client-server environment. Nevertheless, a standardized simulation Web service can open a new quality of *document enrichment*: Documents like CAD drawings, system descriptions, research papers—to mention only a few—can be equipped with the underlying simulation models and be published via the World Wide Web. The recipient of such an enriched document can simulate the embedded models by the press of a button, comparable to the PDF-document standards which allow for the embedding of several kinds of multimedia data.

2. REALIZATION APPROACHES FOR WEB-BASED SIMULATION SERVICES: PROS AND CONS

A Web-based simulation service can be realized as a *Web service*—where the distinction between the two terms "Web-based service" and "Web service" is more than a distinction without difference. The former is a collective term for all kinds of services that can be accessed via the Internet, while the latter describes in fact a distributed software architecture of service components which come along with salient properties (cf. (IBM Web Services Architecture Team, 2000)).

Sleeper defines Web services as loosely coupled, reusable software components that semantically encapsulate discrete functionality and that are distributed and programmatically accessible over standard Internet protocols (Sleeper, 2001). While this informal definition is in accordance with the Web Services Architecture Team at IBM, other authors define Web services meticulously by the following equation (cf. (Page, 1998; Kilgore, 2002)):

"Web Service = HTTP + XML + SOAP
(+ WDSL + UDDI + WSFL)"

Though the quoted protocols and description languages, which have partly been proposed and adopted by the W3C consortium and big software vendors, form a powerful and tailored framework for Web service implementation, Web services can be realized in different ways, so long as they fit into the architectural framework. Among others, they have to play one or more of the fundamental roles, such as service provider, service requester, or service broker. Moreover, they must come along with the following properties: self-containedness, service description, dynamical composition, platform independence, and interoperability. (IBM Web Services Architecture Team, 2000).

In the following we describe the components, the architecture, and the deployment of several realization approaches for a Web-based simulation service and outline problems to be solved.

2.1. Classical Interfaces: RPC, DCOM, RMI, CORBA

RPC, DCOM, RMI and CORBA are technologies that enable a client to remotely invoke functions on a server. If an RPC (DCOM, RMI, CORBA) server is set up, an appropriate RPC (DCOM, RMI, CORBA) client has to be used that can interpret the binary request/response format. This restricts a potential client to a specific platform, vendor, or programming language: RPC is mainly implemented in Unix, DCOM is Microsoft-specific, and RMI is Java-specific. Moreover, RPC doesn't offer access to object-oriented programs, and Microsoft discourages the use of DCOM and pushes the use of SOAP (Box, 2000). CORBA is available for many programming languages and platforms but suffers from a noticeable programming overhead. Aside from its complicated architecture, CORBA implementations from different vendors may not be fully compatible (Hoffman, 1999). As all of the aforementioned interface types are pairwise incompatible and for the most part platform or language-dependent, it is impossible for

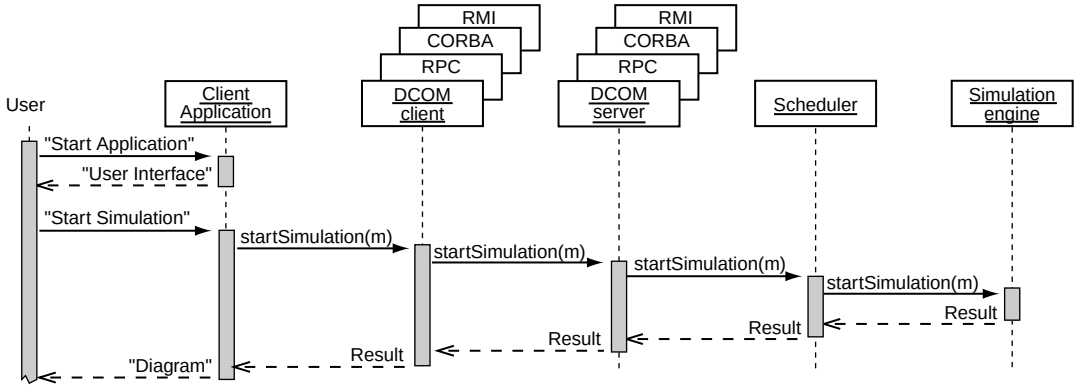

Figure 1 UML sequence diagram for a Web-based simulation service using classical interfaces.

a designer to integrate several Web services that provide several of these interfaces. Since Web services should be available for every interested user, the use of these techniques must be called into question.

A typical application flow for Web-based simulation is depicted in Figure 1. After the client application has been started, the user can request actions, say, function calls in terms of a programming language. Instead of executing the functions locally, their parameters are packed in a special format and transferred to the server. The server unpacks the parameters and calls the corresponding function. In our case, a scheduler observes the load on one or more simulation servers and deploys the execution. Once the simulation finished, the results are again packed in the RPC (DCOM, RMI, CORBA) format and transferred back to the client.

Advantages. (a) The underlying binary protocols for parameter and result transfer ship with the particular implementations.

Disadvantages. (a) The mentioned interface types are pairwise incompatible, and it is unlikely that a client application can integrate several services of the mentioned types. (b) The client programmers choice of the programming language, platform, and middleware depend on the implementation of the server. (c) Clients cannot be run in a Web browser.

2.2. Proprietary TCP/IP Protocol

The invention of application-specific protocols based upon TCP/IP has a long tradition; typical examples are Internet-services like FTP, Finger, or IMap. Each of them is text-based and requires a specialized client that can interpret the protocol. The application flow is similar to Figure 1: Instead of using an RPC (DCOM, RMI, CORBA) communication protocol, a tailored parsing engine and message generator must be implemented. An example for a Web-based simulation service that uses an applet as frontend and a proprietary communication protocol is described in (DYNAST Development Team, 2003b).

Advantages. (a) The commands of a tailored protocol constitute only a small overhead. (b) Data transfer happens at maximum performance. (c) Web clients, like Java applets, can implement the protocol and be run in a Web browser.

Disadvantages. (a) Users who want to access the service without using the standard client must implement the entire protocol. (b) Users who want to compose a service out of several services of this kind will have to implement all of the protocols—a fact which renders a network of Web-based services hard to be set up. (c) Apart from standard search engines, there is no service with which the simulation service might be found.

2.3. HTTP/HTML

Another way to access a Web-based simulation service is to offer an HTML frontend. Figure 2 shows the application flow of an HTTP/HTML-based simulation service. A user enters the URL of a service and usually gets a frontend that contains a text box where a model definition can be entered. Clicking a submission button will transmit the model via the HTTP POST command to the Web server, which in turn passes the contents of the text box to a scripting engine such as a CGI, Perl, PHP, or JSP. The script starts the scheduler and passes the model with a request for simulation. After the simulation is finished, the script picks up the simulation results and dynamically generates either an HTML page that contains raw simulation data or data including an HTML

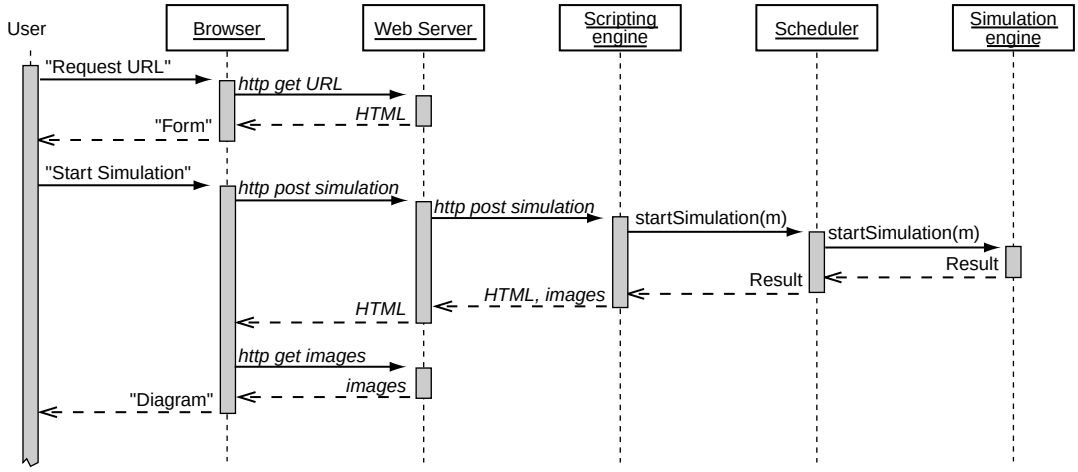

Figure 2 UML sequence diagram for a Web-based simulation service using HTTP/HTML.

reference to server-generated diagrams. In the latter case, the browser loads the diagrams as images from the Web server and displays them. An example for such an implementation is described in (Mann & Ševčenko, 2003) and can be found at the URL given in (DYNAST Development Team, 2003a).

Advantages. (a) Users need only a standard browser.

Disadvantages. (a) The deployment of subtasks to the client is not possible: the entire job must be processed by the server. (b) The result is an HTML-document, which lacks structural information and makes a subsequent formatting difficult. (c) Such a service can hardly be integrated into other applications. (d) Interactions require either large caching capabilities or computing power: If a user wants to zoom into a diagram, a new image has to be generated and transferred. This implies that the simulation data either has to be stored on the server or the simulation data must be recomputed for each interaction.

2.4. HTTP/SOAP

SOAP (W3C Consortium, 2003) is a simple XML-based protocol to let applications exchange information over HTTP. It combines the assets and overcomes the drawbacks of the aforementioned approaches. SOAP is independent of platforms, programming languages, and vendors, and most implementations offer automatic protocol generation for several programming languages. Moreover, concepts for publishing Web services with regard to both semantic and syntax are inclusive. The former is implemented in the form of UDDI (universal description, discovery and integration of Web services) that enables Web service providers to publish the missions along with the addresses of their services in a directory of Web services. The latter relates to WSDL (Web service definition language), a language with which Web service interfaces can be described in an XML representation. Among others, WSDL covers the formulation of complex data types, function names, and parameters. Related to our simulation application, Listing 1 shows that part of the WSDL specification where the complex data type “VariableValues” and the input and output data types of the function “startSimulation” are defined.

Based on an interface definition in the form of Java or C# code for example, a complete WSDL definition can be automatically generated. A SOAP server uses this definition to parse and map incoming SOAP requests to the interface functions. In turn, the gener-

```
<wsdl:definitions name="Simulation"
  targetNamespace=
    "http://www.themindelectric.com/wsdl/Simulation/">
  <wsdl:types>
    <xsd:schema targetNamespace=
      "http://www.themindelectric.com/package/aisim.server/">

      <xsd:complexType name="VariableValues">
        <xsd:all>
          <xsd:element name="numberOfVariables" type="xsd:int"/>
          <xsd:element name=
            "numberOfValuesPerVariable" type="xsd:int"/>
          <xsd:element name=
            "values" type="xsd:ArrayOfArrayOfdouble"/>
        </xsd:all>
      </xsd:complexType>
    ...
  </wsdl:types>

  <wsdl:message name="startSimulationIn">
    <wsdl:part name="arg0" type="xsd:string"/>
  </wsdl:message>

  <wsdl:message name="startSimulationOut">
    <wsdl:part name="Result" type="VariableValues"/>
  </wsdl:message>
  ...
</wsdl:definitions>
```

Listing 1: A part of the generated WSDL definition.

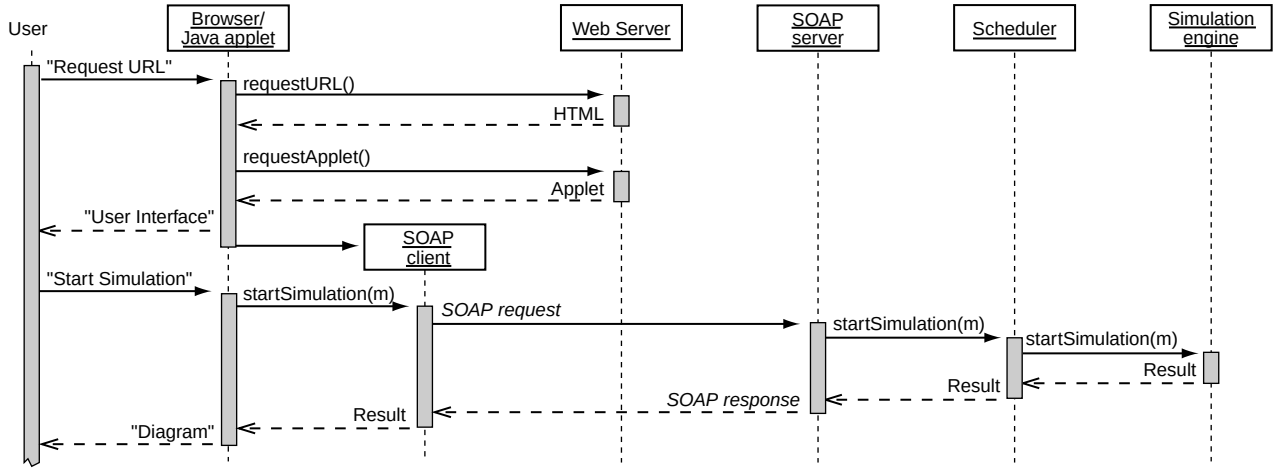

Figure 3 UML sequence diagram for a Web-based simulation service using SOAP.

ated WSDL definition can be used by potential clients to automatically produce the client side communication protocol along with function stubs. This renders calls of remote functions completely transparent for clients. Major software developers like Microsoft, IBM, Sun, and Apache support the SOAP technology.

Figure 3 shows a UML sequence diagram for the invocation of our simulation engine, where an applet is used as frontend. Whenever a user requests the respective URL, the corresponding Web server delivers HTML code which embeds an applet. After its launch the applet instantiates the generated SOAP client, displays the user interface and waits for input. Once the user hits the "start simulation" button, the SOAP message shown in Listing 2 is generated by the SOAP client and sent to the SOAP server.

The message contains the name of the function to be called and its parameters along with their types. The SOAP server strips the HTML wrapper from the SOAP message, parses the content, reconstructs the parameter data types, and calls the requested function. The SOAP server wraps the results in a SOAP envelope similar to the request and sends it back to the SOAP client. The client reconstructs the delivered data types and passes them to the client application, in our case to the Java applet.

```
POST /glue/simulation HTTP/1.1
Host: 131.234.41.48:8004
Connection: Keep-Alive
User-Agent: TME-GLUE/4.1.2
SOAPAction: "startSimulation"
Content-Type: text/xml; charset=UTF-8
Content-Length: 482

<?xml version='1.0' encoding='UTF-8'?>
<soap:Envelope xmlns:xsi=
    'http://www.w3.org/2001/XMLSchema-instance'
  xmlns:xsd='http://www.w3.org/2001/XMLSchema'
  xmlns:soap='http://schemas.xmlsoap.org/soap/envelope/'
  xmlns:soapenc='http://schemas.xmlsoap.org/soap/encoding/'>
  <soap:Body soap:encodingStyle=
    'http://schemas.xmlsoap.org/soap/encoding/'>
    <startSimulation>
      <arg0 xsi:type='xsd:string'>circuit.mo</arg0>
    </startSimulation>
  </soap:Body>
</soap:Envelope>
```

Listing 2: SOAP request for a call of the function "startSimulation" with the parameter "circuit.mo".

Advantages. (a) Client-side as well as server-side protocols can be generated automatically from an interface definition. (b) The data contains a logical structure. (c) SOAP provides meta-information about data structures that are exchanged in the form of WSDL. This enables modern programming languages to reconstruct the data structures at runtime. (d) Metadata concerning the purpose of the simulation service can be provided in a directory of Web services. (e) A standardized network of Web services becomes possible. (f) Standard encryption via HTTP/SSL is possible (HTTPS). (g) Major software vendors support SOAP within their platforms and programming language APIs. (h) SOAP is recommended by the W3C and may get its own Mime type.

Disadvantages. (a) Overhead when wrapping data in HTML/XML/SOAP envelopes. (b) SOAP client code for message parsing in applets is (still) too big.

2.5. Unsolved Problems

The outlined realization alternatives address the communication problem, with the given advantages and drawbacks; nevertheless, there are desirable enhancements that are common to all of them. In a scenario where an application embeds a third party Web service, functions must be called in a given order,

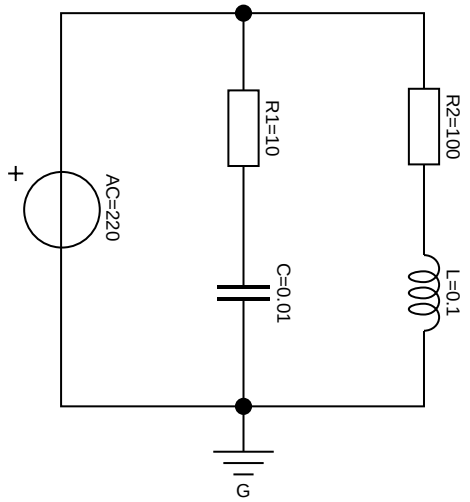

Figure 4 A simple electrical circuit.

```
model circuit
  Resistor   R1(R=10);
  Capacitor  C(C=0.01);
  Resistor   R2(R=100);
  Inductor   L(L=0.1);
  VsourcesAC AC;
  Ground     G;
equation
  connect (AC.p, R1.p);
  connect (R1.n, C.p);
  connect (C.n, AC.n);
  connect (R1.p, R2.p);
  connect (R2.n, L.p);
  connect (L.n, C.n);
  connect (AC.n, G.p);
end circuit
```

Listing 3: The Modelica description of the circuit depicted in Figure 4.

i. e. a model must first be transmitted and then simulated. Let us assume that a remote service offers a function that returns a list of all variable names. Then the question is whether this function may be called directly, after transmitting the model, or whether the variable list shall be accessible only when the model has been parsed for execution. Obviously there are restrictions on the function call order, which could explicitly be modeled in a dedicated language that gets part of the Web service definition. Such a language could be used to detect semantic flaws in a client application. Current approaches, such as WSFL (IBM), XLANG (Microsoft), BPEL4WS (IBM/BEA/Microsoft), WSCI (BEA/SAP/Sun), WSCL (Hewlett Packard) are not recommended yet by the W3C consortium and must be considered being proprietary.

Another concern is encryption. SOAP offers a standard way for channel encryption: HTTP tunneling through the Secure Socket Layer (SSL). Although approved channel encryption technology secures the transmission, the decrypted model is available in a plain form at the server side, as it must adhere to the simulator's model representation. Due to the fact that models may comprise crucial business know-how, the client must trust the service provider. An option that cannot be implemented in a Web service protocol but in a Web service client is model obfuscation, which could substitute inane identifiers for the meaningful model constituents.

From the viewpoint of a company that provides a simulation service there is the need for an efficient load balancing and scheduling mechanism: Simulation jobs may concentrate at a peak-time, within the core working hours of a country. Observe that for an efficient scheduling the duration of a simulation job has to be estimated. Though rules of thumb can be applied for such estimations, a reliable duration estimations is subject of current simulation research.

3. A PROTOTYPIC WEB SIMULATION SERVICE FOR MODELICA

The purpose of this section is twofold. The first two subsections give a very brief introduction to the Modelica modeling language and the YANOS simulator; the remainder, Subsection 3.3 and 3.4, explains how SOAP is used to deploy part of the functionality of the YANOS simulator as a Web service.

3.1. On Modelica™

Modelica is a language for modeling physical systems. What makes Modelica so attractive for Web-based simulation?—At least three points: It is an open specification, it is standardized, and it incorporates state of the art modeling technology. The following text as well as the example rely on articles and information material that can be found at `www.modelica.org` (Modelica Association, 2000a; Modelica Association, 2000b).

Consider the electrical circuit in Figure 4. It consists of a voltage source, a ground point, two resistors, a capacitor, and an inductor. A Modelica description of this circuit is giving in Listing 3.

The description both declares the components and, in-

troduced by the keyword `equation`, defines the device topology. For example, the line

```
Resistor   R1(R=10);
```

declares the variable `R1` being of class `Resistor` and sets the field `R` to the value of 10. The line

```
connect (R1.n, C.p);
```

states that pin `n` of resistor `R1` is connected to pin `p` of capacitor `C`. Note that by virtue of the connect construct also the necessary compatibility and continuity conditions are implicitly defined, which, in electrical engineering, correspond to potential identity and Kirchhoff's current law respectively.

Modelica has a lot of features that are known from the modern, object-oriented programming languages. Moreover, it provides support for matrices, units, quantities, and even for the specification of processing hints for numerical algorithms.

Modeling with Modelica means modeling at the physical component level, as opposed to the classical block-oriented modeling. Block-oriented models follow local relationships and can, in principle, be processed by local propagation. Therefore, this kind of modeling is also called "causal", whereas the modeling that is oriented at the device structure is called "non-causal". From the modeling viewpoint, non-causal modeling is by far superior to causal modeling where the burden of the algorithmic formulation of the underlying mathematical equations is shifted to the user. Clearly, this means on the other hand that the processing of non-causal models, such as Modelica models, is much more demanding since it must afford this model formulation intelligence.

3.2. The Modelica Simulator YANOS

YANOS is a simulation engine for the Modelica language and is being developed by the Art Systems Software Ltd; see Figure 5 for an overview of its core modules. By now, YANOS supports a subset of the Modelica language specification—which currently is at release 2.0—and is continuously extended. In particular, the following major concepts are supported (●) and not supported (○) respectively:

- Solution of implicit, differential-algebraic equation systems.
- Efficient symbolic manipulation of large algebraic systems.

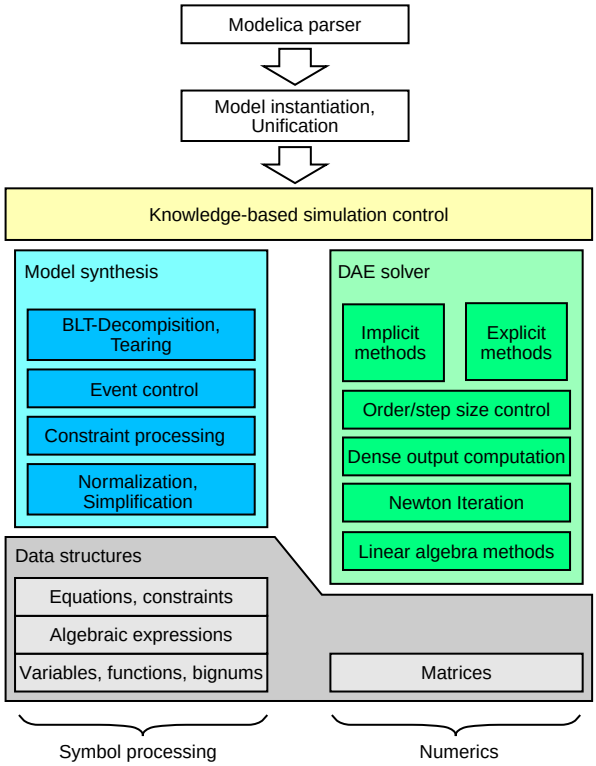

Figure 5 Overview of the core modules in the YANOS simulation engine.

- Normalization, simplification, and substitution of algebraic expressions.
- Arbitrary precision with big integers.
- ○ Formulation of vectors and matrices.
- ○ Formulation of algorithms within models.
- ○ Consistent initialization of higher index systems.

The YANOS simulation engine implements recent algorithms for the analysis of stiff systems (Dormand, 1996; Hairer & Wanner, 1996) and realizes a knowledge-based interplay between the collection of model equations and the application of an integrator's solution equations. This way it can resemble among others the behavior of the famous DASSL algorithm (Petzold, 1982), but also apply the inline integration concept to several integration procedures (Elmqvist et al., 1995).

A strong point of YANOS is its tight integration of computer algebra at simulation runtime, which provides a high level of flexibility for behavior analysis: It enables YANOS to apply a spectrum of algebraic methods in the course of a simulation, e. g., if a system changes its mode or its structural setup.

The YANOS simulation engine is encapsulated in a scheduler that provides different organizational facilities: The syntactical analysis and instantiation of Modelica models, the user management, the scheduling of different simulation tasks, or the upload and publication of simulation models. Together, these modules form the simulation server. There exist different frontends (clients) for the simulation server. Especially for Web deployment purposes we have developed a client in the form of a Java applet that provides the following basic functionality:

- Selection, upload, and textual manipulation of Modelica models.
- Graphical display of state trajectories.
- Definition of basic experimental constraints.

3.3. Soaping YANOS

The following points summarize the steps that are necessary to add a SOAP interface to YANOS using the GLUE SOAP implementation (The Mind Electric, 2003).

1. *Interface Design.* Figure 6 outlines our plan for function shipping. The client side consists of a Web browser that runs the Java applet; the applet contains the code for handling user interactions as well as the (automatically generated) code for parsing the SOAP responses. We decided to transfer raw simulation data (the trajectories of the variables) to the client and let the client do all presentation-related tasks like the drawing of diagrams with respect to interesting variables. Consequently, the interface can be kept narrow: It contains functions to load Modelica models, to specify simulation parameters, to start the simulation, and to fetch simulated values.

   The server side consists of a Web server, which delivers the applet to the client and which has a SOAP server integrated besides the standard scripting engines. We built a Java wrapper that calls the native YANOS functions and added functionality to schedule simulations, and to buffer simulation data. The buffer concept enables a user to specify the data packet size within a SOAP response and hence to define the frequency by which the client display is updated.

2. *WSDL Generation.* Given the Java wrapper interface, the WSDL definition can be generated using the GLUE java2wsdl-converter (see Listing 1).

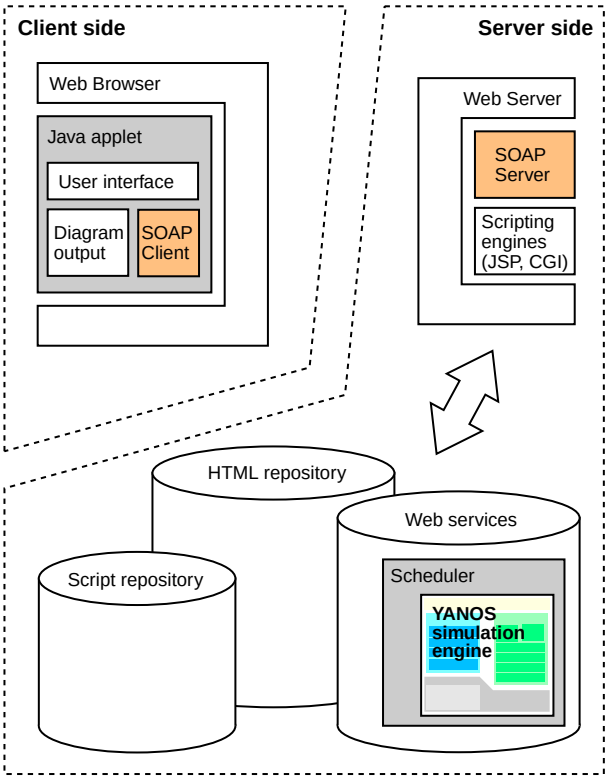

Figure 6 Overview of the YANOS Web architecture.

3. *Client Code Generation.* The generated WSDL definition can be used as input for a client code generator. GLUE offers the wsdl2java-tool that generates SOAP clients along with Java method stubs. We used the stubs as a basis for the Java applet and realized functions for displaying diagrams etc. according to Point 1.

4. *Publication.* If the Web service is published via UDDI there are two alternative invocation scenarios: (a) A user can download our applet client and use it as frontend. (b) A user can generate method stubs from the published WSDL definition and integrate the simulation service in own applications.

3.4. The YANOS Web Interface

Figure 7 shows a screenshot of our applet. On the left-hand side, models can be chosen for instantiation; according schematic views are displayed and can be examined. Once a model is instantiated, its variables are sorted according to their type (state, parameter, or other) and shown in a tree. Each variable can be selected to be plotted, and start values can be provided for the state quantities (middle). When the simulation is started, all settings are submitted to the SOAP

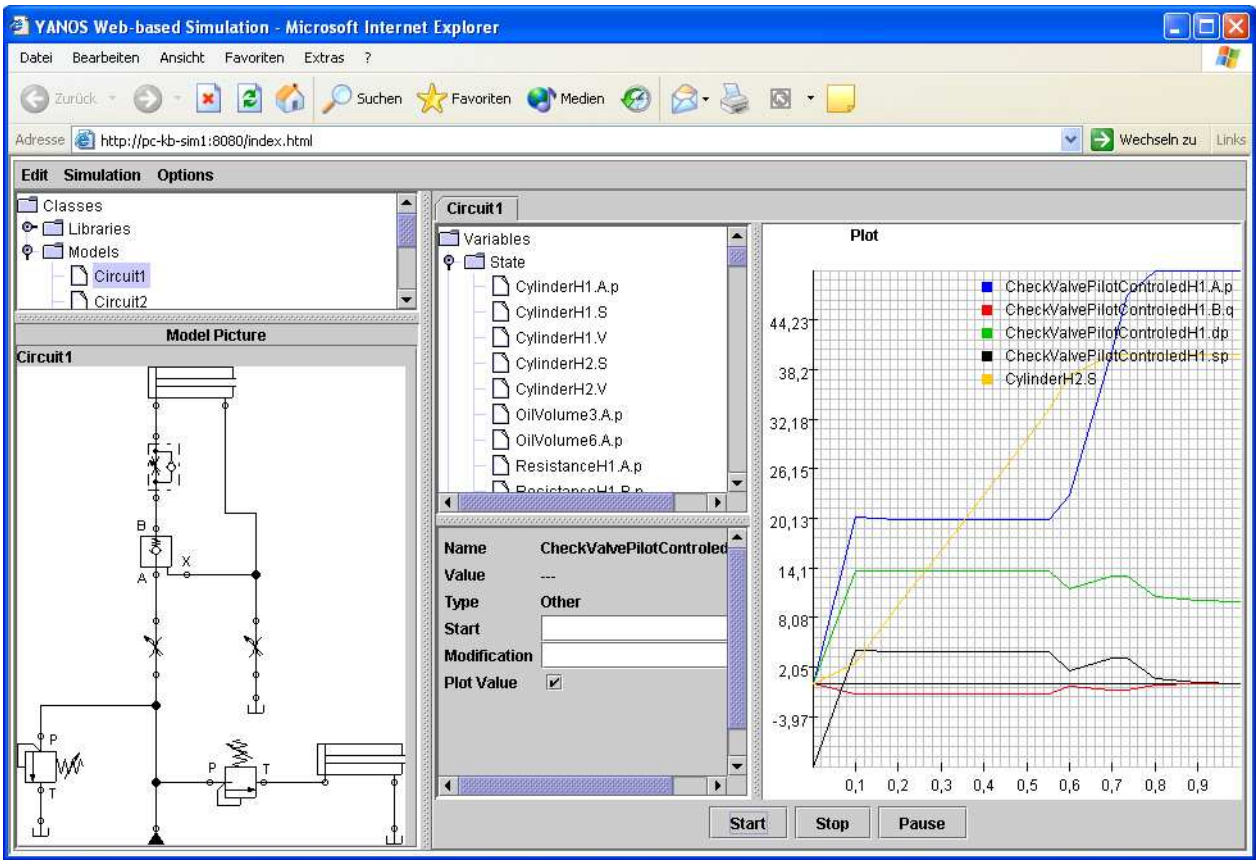

Figure 7 Screenshot of the YANOS Web Interface.

backend. The curves in the plot window (right) are updated whenever the backend sends computed variable values or when a user changes the selection of variables to be displayed. Several models can be simulated in parallel: When a user decides to analyze another model, a new tab is opened. This enables one to compare models and variable curves, and to analyze the impact of parameter and model variations.

CONCLUSIONS AND FUTURE RESEARCH

Web services for simulation provide platform independence, automatic licensing, version control, and deployment facilities. SOAP is a simple protocol that enables a simulation Web service to interact with other applications. In particular, based on WSDL, SOAP generates communication protocols automatically and can provide meta information of the syntax and semantics of the simulation service, which then can be published in directories of Web services.

Currently we experiment with the development of a document format wherein Modelica models can be embedded and that can be simulated interactively over the World Wide Web with a mouse click. Note that for these purposes also a temporary ticket should be generated, which grants a transient license to simulate a model during a fixed period. Although there are several challenges to be mastered, we are optimistic that such a service can become standard in the medium-term future.

REFERENCES

Box, D. (2000). A Young Person's Guide to The Simple Object Access Protocol: SOAP Increases Interoperability Across Platforms and Languages. `http://msdn.microsoft.com/msdnmag/issues/0300/soap/toc.asp`.

Dormand, J. (1996). Numerical Methods for Differential Equations. New York, London, Tokyo: CRC Press.

DYNAST Development Team (2003a). DYNAST Collection of Solved Examples. `http://icosym.cvut.cz/dyn/examples`.

DYNAST Development Team (2003b). DYNCAD. `http://icosym.cvut.cz/dyncad/applet`.

Elmqvist, H., Mattsson, S., & Otter, M. (1999). Modelica—A Language for Physical System Modeling, Visualization, and Interaction. In Proceedings of the IEEE Symposium on Computer-Aided Control System Design, CACSD'99 Hawaii: pp. 630–639.

Elmqvist, H., Otter, M., & Cellier, F. (1995). Inline Integration: A New Mixed Symbolic/Numeric Approach for Solving Differential-Algebraic Equation Systems. In Proceedings of the European Simulation Multiconference, ESM'95 Prague, Czech Republic: pp. xxiii–xxxiv.

Fishwick, P. (1997). Web-based Simulation. In Proceedings of the 29th Winter Simulation Conference (WSC'97): ACM Press pp. 100–102.

Hairer, E. & Wanner, G. (1996). Solving Ordinary Differential Equations II. Stiff and Differential-Algebraic Problems. Berlin Heidelberg New York: Springer, second edition edition.

Hoffman, R. (1999). Sneaking Up On CORBA: The Race for the Ideal Distributed Object Model. `http://www.networkcomputing.com/1009/1009f2.html`.

IBM Web Services Architecture Team (2000). `http://www-106.ibm.com/developerworks/webservices/library/w-ovr/`.

Kilgore, R. (2002). Simulation Web Services with .NET Technologies. In E. Yücesan, C.-H. Chen, J. Snowdon, & J. Charnes (Eds.), Proceedings of the 34th Winter Simulation Conference (WSC'02): ACM Press pp. 841–846.

Mann, H. & Ševčenko, M. (2003). Simulation and Virtual Lab Experiments across the Internet. In Proceedings of the International Conference on Engineering Education Valencia, Spain.

Modelica Association (2000a). Modelica™—A Unified Object-Oriented Language for Physical Systems Modeling: Tutorial. Modelica Association, Linköping, Sweden.

Modelica Association (2000b). The Modelica Specification, version 2.0. Modelica Association, Linköping, Sweden.

Page, E. (1998). `http://www.mitre.org/news/the_edge/august_98/wbs.html`.

Petzold, L. (1982). A Description of DASSL: A Differential / Algebraic System Solver. In Proceedings of 10th IMACS World Congress on System Simulation and Scientific Computation Montreal.

Sleeper, B. (2001). `http://www.stencilgroup.com/ideas_scope_200106wsdefined.html`.

The Mind Electric (2003). The GLUE SOAP Implementation. `http://www.themindelectric.com/glue/index.html`.

W3C Consortium (2003). SOAP Version 1.2 W3C Recommendation. `http://www.w3.org/TR/soap12-part1/`.

Yücesan, E., C.-H. Chen, Snowdon, J., & Charnes, J., Eds. (2002). Proceedings of the 34th Winter Simulation Conference (WSC'02). ACM Press.