Realization of Web-based Simulation Services

Sven Meyer zu Eissen Benno Stein

Department of Media Systems
Bauhaus University Weimar

Abstract

Web-based simulation is a collective term used for various applications and with different meanings: simulation as hypermedia, simulation research methodology, Web-based access to simulation programs, distributed modeling and simulation, and simulation of the WWW (1).

Here, the term Web-based simulation relates to the first three areas, where we see great potential in bringing simulation technology to the Web: Exciting applications include the development of new MIME types for technical documents or the realization of standardized service building blocks, which make agile workflow modeling possible in the technical departments of many companies.

The contributions of this paper are as follows. The different realization alternatives for Web-based simulation services are explained and discussed with respect to their advantages and disadvantages. Moreover, the prototypic implementation of a Web service is presented, which allows for the analysis and execution of technical models described in the well-known Modelica modeling language. While existing simulators use proprietary or non-interactive communication concepts for Web access, our service is built on the proposed W3C Web interface stack. In particular, it integrates the professional simulation engine YANOS, wich is employed among others in the simulation software FluidSIM of FESTO. Our Web service enables the electronic mailing of technical documents, which may contain model descriptions that can be simulated in the Web browser of the recipient.

Key words: Web-based Simulation, Web Service, SOAP, Modelica

1 Rationale of Web-based Simulation Services

The term Web-based simulation as it is used in this paper relates to non-distributed, single-user simulation tasks. This is an important problem class since the simulation of technical systems is usually carried out in this way, be

Email addresses: sven.meyer-zu-eissen@medien.uni-weimar.de (Sven Meyer zu Eissen), benno.stein@medien.uni-weimar.de (Benno Stein).

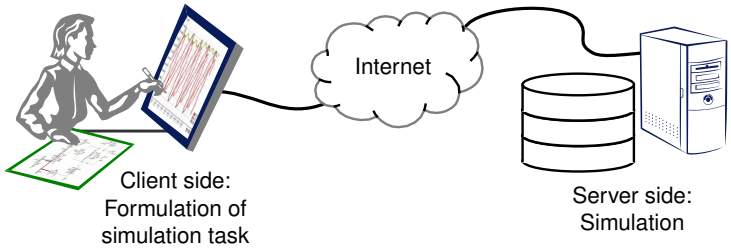

Figure 1. Simulation scenario: Model libraries, simulation and model formation algorithms as well as algorithms for information visualization are made available as Web-services.

it discrete event simulation, continuous time simulation, or hybrid simulation. The term "non-distributed" means that a simulation job is not distributed on several compute servers but processed on a single machine. Note that, elsewhere, Web-based simulation is also associated with distributed processing or with multi-user scenarios, and that existing Web-based simulation applications primarily focus on discrete event system simulation (DESS) (2; 3).

Figure 1 illustrates the typical setting for our problem class: At the client side, a user formulates a model in a high-level modeling language such as VHDL-AMS or Modelica (4). 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, or syndication.

Note that a number of apparent as well as future use cases become possible if the aforementioned tools are operationalized in the form of Web-based services (see also (5)). 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.
- Dedicated 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 diagnosis (6).
- Simulation services for the purpose of third party analyses and comparative evaluations can be realized.
- 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 Web service for simulation can open a new quality of *interactive documents*: 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 interactive 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.

The goal of our research is the development of powerful and robust Web-based simulation services, the evaluation of the software engineering requirements, and the preparation of feasibility studies with different industrial and educational partners.

2 Realization Approaches for Web-based Simulation Services

Apart from enabling the aforementioned Internet-based use cases, a Web-based simulation service addresses a major challenge in industrial engineering: the integration of simulation software within the process of system analysis and design. Integration difficulties fall into two categories: (a) model exchange, model coupling, and model reuse, and (b) workflow integration and optimization. The former category includes the exchange of models for different simulation purposes, as well as the coupling of device models from different domains, e. g. when simulating a hybrid system. The latter category relates to the integration of simulation software into a company's business information systems infrastructure in order to support the design, the production, the maintenance, and the redesign of a product during its whole life cycle.

From a technical point of view, a simulation engine that is realized in the form of a *service* can be accessed globally, via the Internet, or locally, via a company's intranet. We understand a service as a function that is well-defined, self-contained, and that does not depend on the context or state of other services (7). Moreover, services are loosely coupled and invoked through

communication protocols that emphasize location transparency and interoperability (8). Loosely coupled means that a service requester and a service provider know only a minimum about each other. For self-documentation, a service should provide its own description (9), which may comprise a semantic description of its function and its interface specification. Another requirement for services, which follows directly from the interoperability, is platform independence, since services may be realized on different operating systems and in different programming languages. If the core intelligence in a company's intranet is realized through the composition of services, one speaks of a so-called "service-oriented architecture" (SOA).

Figure 2 illustrates the SOA principle: it shows how services for retrieval, simulation, and optimization are integrated into a product designer's workflow. At our institute we have developed tools that support the mentioned functions within a system analysis and design process (10; 11); Web services are the means of choice to integrate these tools under the SOA paradigm. [1]

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 (12). 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. (1; 2)):

$$\text{"Web Service} = \text{HTTP} + \text{XML} + \text{SOAP} \\ (+ \text{ WSDL} + \text{UDDI} + \text{WSFL})\text{"}$$

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 implementation, Web services can be realized in different ways, so long as they fulfill the mentioned properties of services.

In the following we discuss deployment and communication concepts of well-known realization approaches for a Web-based simulation service and outline problems to be solved.

[1] One scenario we have been focusing on lets a human designer formulate his mental model of an interesting system by means of a CAD-like editor. The resulting raw design is refined by an expert system using a database of design cases along with a rule-based modification and repair language. Since each modification of the model may entail substantial changes in its behavior, simulation is used to analyze and control evolution of the design.

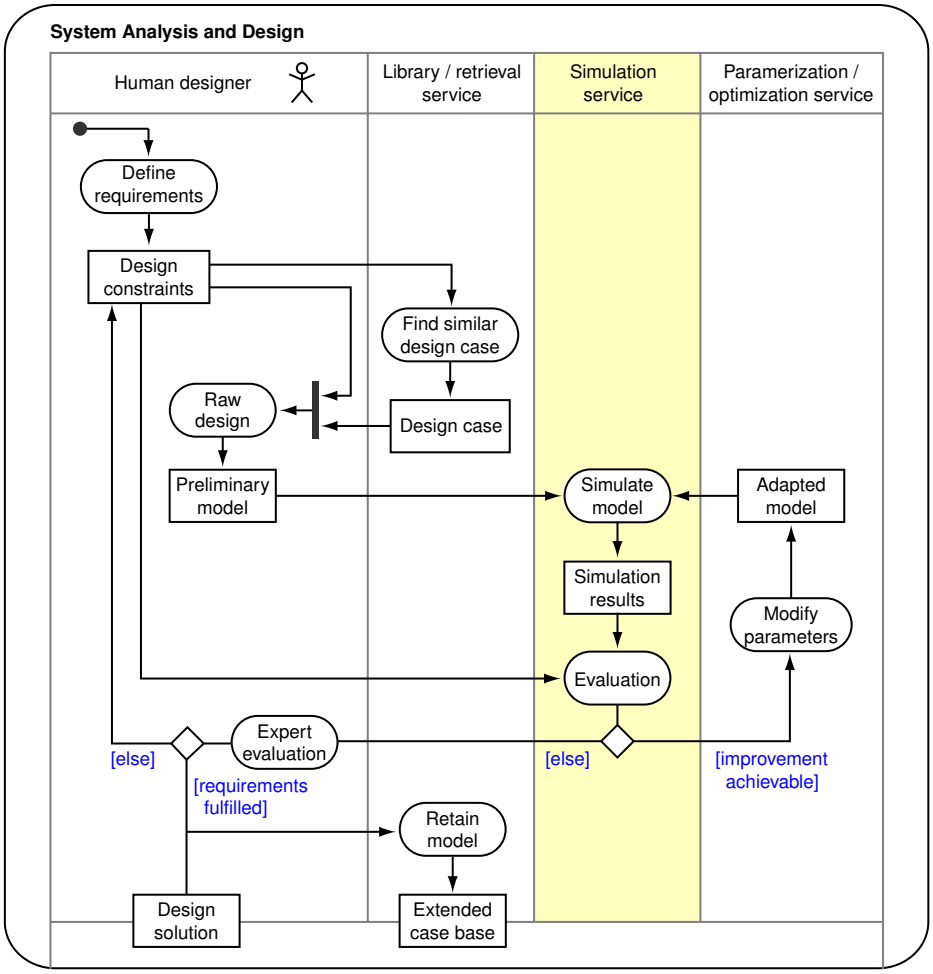

Figure 2. Simplified UML activity diagram that shows the interplay of a case-based retrieval service, a simulation service, and a service for model optimization and parameter finding as part of a product designer's workflow. In the ideal case, it keeps transparent for the human designer whether the shown functions are provided by a Web service or by locally installed tools.

2.1 Classical Communication Concepts: RPC, DCOM, RMI, CORBA

Several vendors developed technologies to let a client remotely invoke functions on a server (cf. Figure 1). Well-known examples are Sun's Remote Procedure Call (RPC) (13), Microsoft's Distributed Component Object Model Technology (DCOM) (14), Sun's Java-specific Remote Method Invocation (RMI) (15), and OMG's Common Object Request Broker Architecture (CORBA) (16). 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 typically implemented on Unix systems, DCOM is Microsoft-specific, and RMI is Java-specific. More-

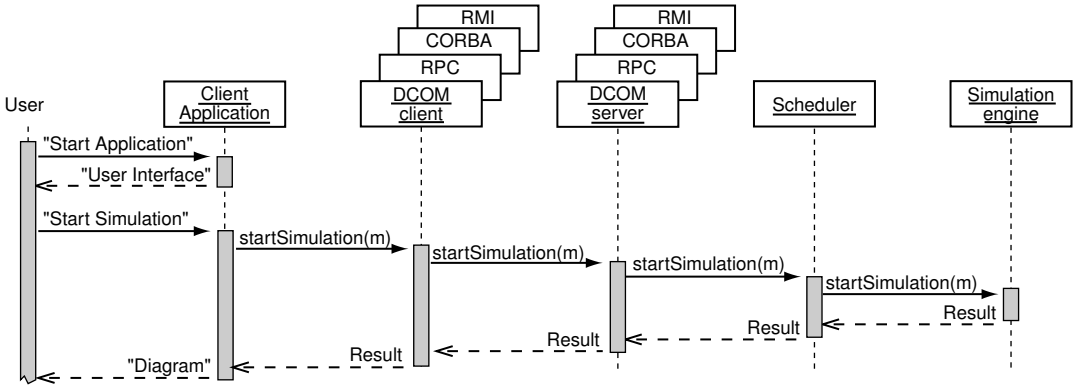

Figure 3. UML sequence diagram for a Web-based simulation service using classical interfaces.

over, RPC does not offer access to object-oriented programs, and Microsoft discourages the use of DCOM and pushes the use of SOAP (17). 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 (18). As all of the aforementioned interface types are pairwise incompatible and for the most part platform or language-dependent, it is impossible for 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 UML sequence diagram for a Web-based simulation service using classical interfaces is depicted in Figure 3. 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 protocol implementations. Consequently, applications that use these protocols do not have to implement protocol parsers. (b) Binary protocols allow a compact representation of function parameters and results, yielding to high transmission speed.

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

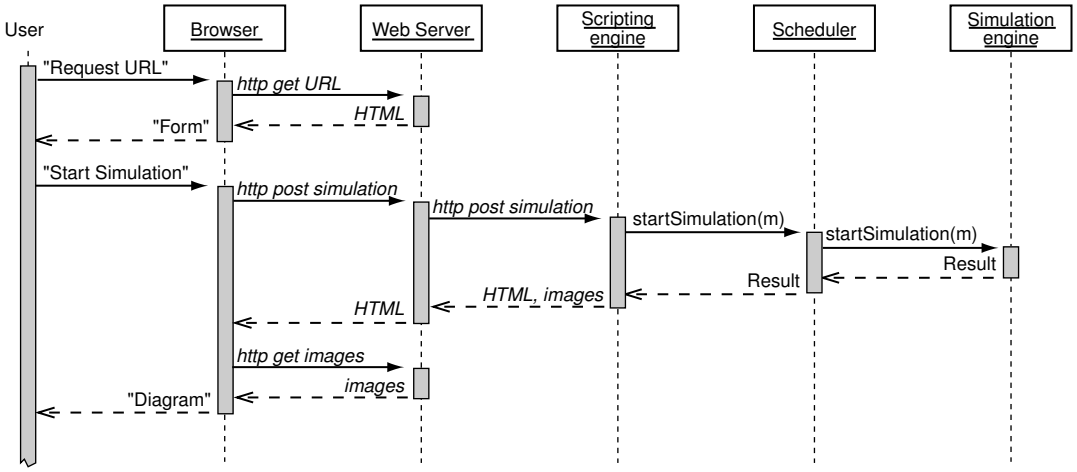

Figure 4. UML sequence diagram for a Web-based simulation service using HTTP/HTML.

server. (c) Clients cannot be run within a Web browser.

2.2 Proprietary TCP/IP Protocol

The invention of application-specific protocols based on TCP/IP has a long tradition; typical examples are Internet-service protocols like the File Transfer Protocol (FTP) (19), the Simple Network Management Protocol (SNMP) (20), or the Internet Message Access Protocol (IMAP) (21). Each of them is text-based and requires a specialized client that can interpret the protocol. The application flow is similar to Figure 3: 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 (22).

Advantages. (a) The commands of a proprietary protocol constitute only a small overhead as they are tailored to the underlying application. (b) Data transfer happens at maximum performance since a protocol designer will minimize the size of messages that are to be transferred. (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.

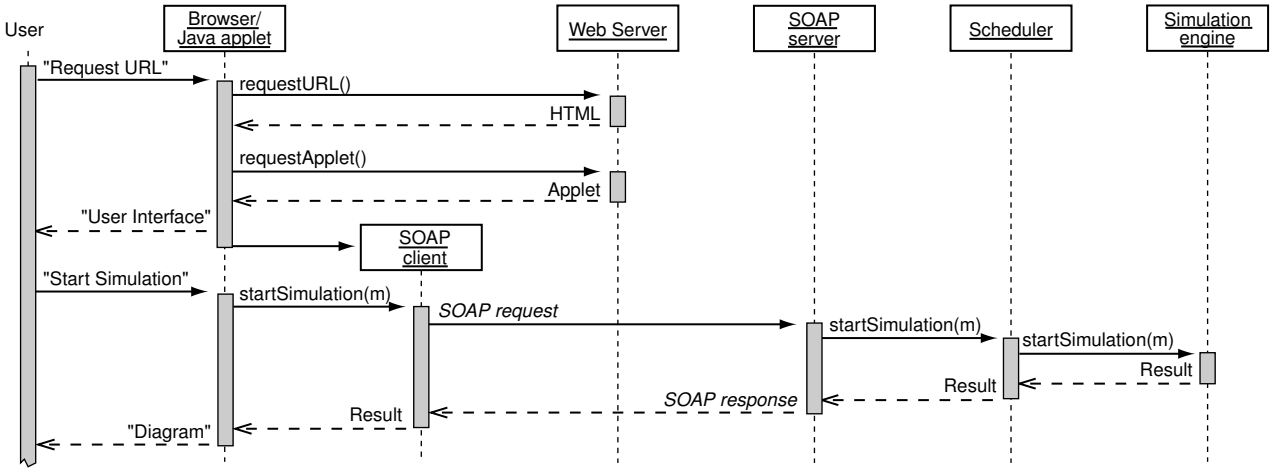

Figure 5. UML sequence diagram for a Web-based simulation service using SOAP.

2.3 HTTP/HTML

Another way to access a Web-based simulation service is to offer an HTML frontend. Figure 4 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 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 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 (23) and can be found at the URL given in (24).

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.

```
<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>
```

The header denotes that object types and messages belong to the "Simulation" service.

Definition of the complex data type "VariableValues". It consists of an integer called "numberOfValuesPerVariable" and an array of arrays of double precision values called "values".

Definition of the function "startSimulation". It takes as input a String called "arg0" (in this case, the name of the model to be simulated), and returns as result an object of type "VariableValues", which has been defined above.

Figure 6. A part of the generated WSDL definition.

2.4 HTTP/SOAP

The Simple Object Access Protocol (SOAP) is an XML-based protocol to let applications exchange information over HTTP (25). It combines the advantages 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. We chose SOAP for implementation because the aforementioned properties fulfill all requirements for *service* implementation, as discussed in the beginning of this section, enabling a service-oriented architecture. Related to our simulation application, the listing in Figure 6 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

```
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>
```

The HTTP header gives the information that the function "startSimulation" is requested to be invoked at the server side.

The SOAP header ("envelope") defines that the XML message contains a SOAP request.

The SOAP message is a request to remotely invoke the "startSimulation" function with the model "circuit.mo" as argument.

Figure 7. SOAP request for a call of the function “startSimulation” with the parameter “circuit.mo”.

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

Figure 5 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 the listing in Figure 7 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.

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) Meta-data 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.

Although the list of advantages for SOAP-based communication is large SOAP must not be seen as a cure-all: A virtual enterprise will have to arrange an agreement with its business partners as to employed protocols.

2.5 Unsolved Problems

The outlined realization alternatives from Subsections 2.1 to 2.4 address the SOA implementation question, say, 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, 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, or 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 contain 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

```
model Circuit1
  Manometer mano1;
  ReliefValve relief1(op=60);
  CheckValve check1;
  ...
equation
   connect(mano1.A, relief1.P);
   connect(relief1.P, check1.B);
   ...
end Circuit1;

model CheckValve
  PinH A, B;
  parameter Pressure pNom=1;
  ...
equation
   if dp > pNom then
      A.q = sqrt(dp/Rh);
   else
      A.q = 0;
   end if;
  ...
end CheckValve;
```

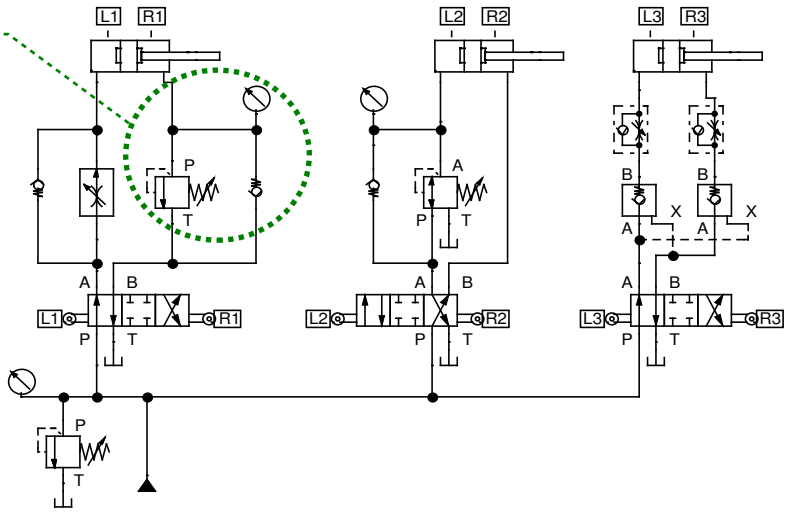

Figure 8. The diagram shows a manufacturing sequence consisting of a feed unit, a press, and a holding unit. The box on the left contains the Modelica definition of the behavior for the encircled diagram part.

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, explains how SOAP is used to deploy the functionality of the YANOS simulator as a Web service.

3.1 On Modelica™

Modelica is a language for modeling physical systems; it is attractive for Web-based simulation for several reasons: it is an open specification, it is standardized, and it incorporates state of the art modeling technology.[2] A fourth point, which is technically more involved, is introduced below.

[2] See `www.modelica.org` (26; 27)

Consider the circuit in Figure 8. It consists of valves, cylinders, throttles, and pumps and represents a manufacturing sequence from a metal processing application. The behavior specification at the left shows the descriptive power of the Modelica language. The specification both declares the components and, introduced by the keyword `equation`, defines the circuit topology. For example, the line

```
ReliefValve    relief1(op=60);
```

declares the variable `relief1` being of class `ReliefValve` and sets its opening pressure `op` to the value of 60. The line

```
connect  (relief1.P, check1.B);
```

states that pin `P` of the relieve valve is connected to pin `B` of the check valve. Note that by virtue of the connect construct also the necessary compatibility and continuity conditions are implicitly defined, which, in fluidic engineering, correspond to pressure identity and the law of the conservation of mass.

Observe that 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. Hence, this kind of modeling is also called "causal", whereas the modeling that is oriented at the device structure is called "non-causal" (28). From the modeling viewpoint, non-causal modeling is by far superior to causal modeling where the burden of the algorithmic reformulation of the underlying mathematical equations is shifted to the user. In fact, this property is also very interesting from the viewpoint of Web-based simulation: The specification of a simulation problem (at the client side) is declarative and happens completely uncoupled from its solution (at the server side). As a consequence, user interaction and simulation feed-back during the specification of models, simulation constraints, and simulation tasks is not necessary. 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.

Finally, it should be noted that Modelica has several features that are known from object-oriented programming languages. Among others, it supports type checking, multiple inheritance as well as redeclaration of behavior descriptions, and basic mechanisms for access control. Moreover, it provides support for matrices, units, quantities, and even for the specification of processing hints for numerical algorithms.

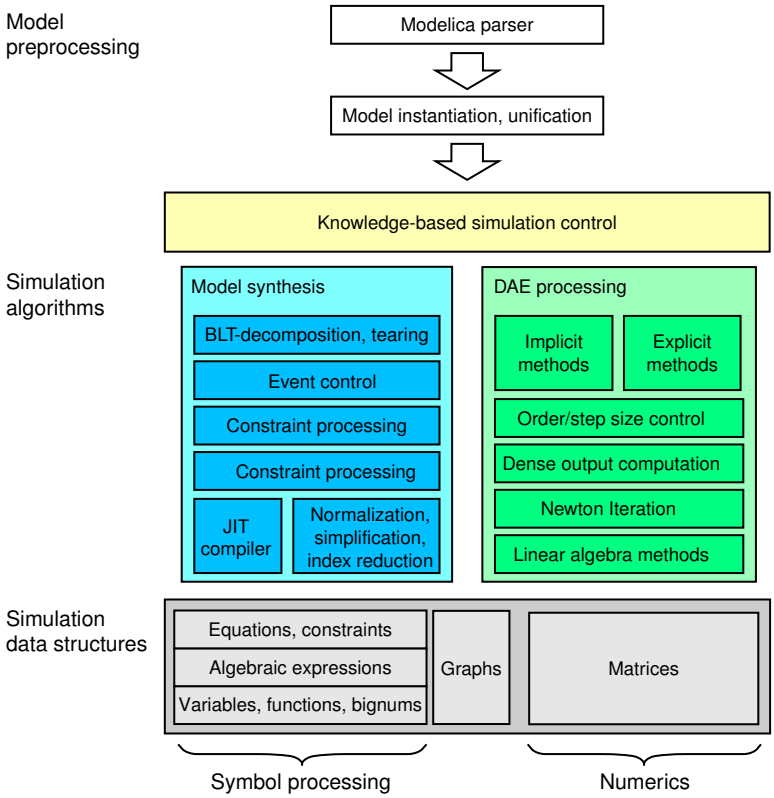

Figure 9. Overview of the core modules in the YANOS simulation engine.

3.2 The Modelica Simulator YANOS

YANOS is a professional simulation engine for the Modelica language and has been rigorously designed from scratch with special attention to high simulation performance, a small memory footprint, and extendability; see Figure 9 for an overview of its core modules. YANOS supports a large subset of the Modelica language specification—which currently is at release 2.2. The YANOS simulation engine has proven its suitability: It forms the backbone of Festo's simulation software FluidSIM™, from which more than 140.000 registered licenses have been sold world-wide.

From the mathematical standpoint the YANOS simulation engine implements recent algorithms for the analysis of so-called stiff systems (29; 30); moreover, it 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 the behavior of the famous DASSL algorithm (31), but also apply the inline integration concept to several integration procedures (32).

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:

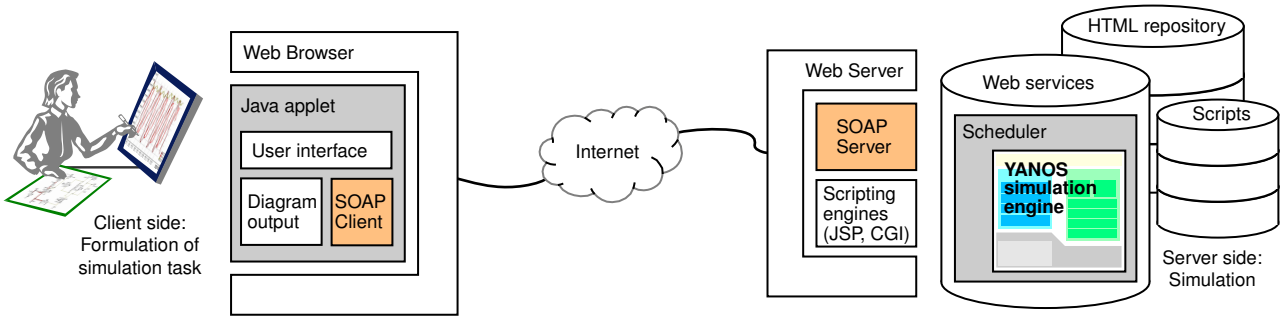

Figure 10. Illustration of the YANOS Web deployment.

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. Though its high flexibility, simulation performance is not compromised since YANOS has a just-in-time compiler built in.

What can not be seen in Figure 9 is that 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 YANOS Web Deployment

The following points summarize the steps that are necessary to add a SOAP interface to YANOS using the GLUE SOAP implementation (33).

(1) *Interface Design.* Figure 10 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 6).

(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.

Figure 11 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 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.

4 Summary and Discussion

Web services for simulation provide platform independence, automatic licensing, version control, and deployment facilities. The purpose of this paper was threefold: it presented application scenarios of Web services for simulation, it discussed technologies to realize them, and it introduced the prototype of a fully-fledged simulation service with a professional simulation engine as backend. Our service has been implemented with SOAP; it follows the service-

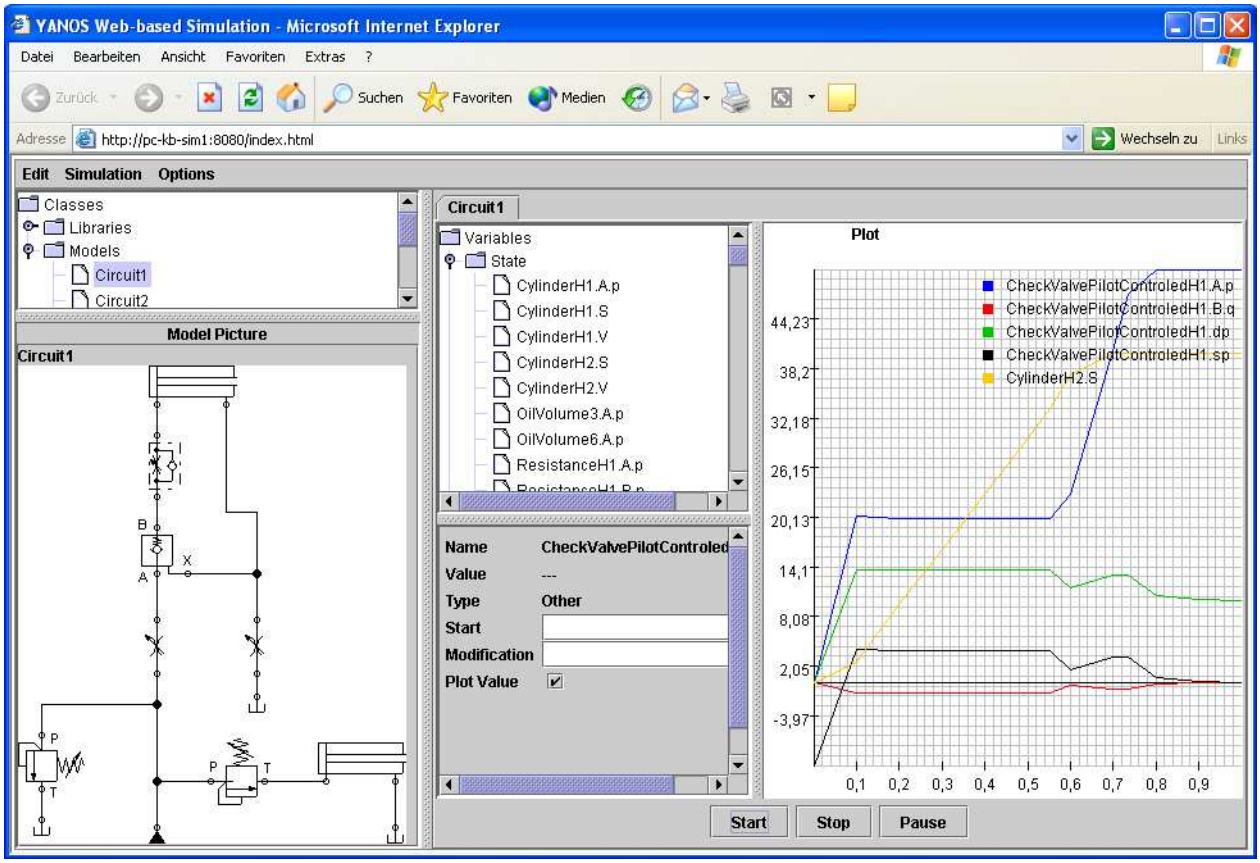

Figure 11. Screenshot of the YANOS Web Interface.

oriented architecture paradigm and can be integrated easily into a company's workflow.

SOAP is a simple protocol that is defined on top of the HTTP protocol stack and that realizes the transparent distribution of request/response-functionality; it has been developed to ease the implementation of Web services: Based on WSDL, communication protocols for SOAP can be generated automatically and meta information regarding the syntax and semantics of a service can be published in designated Web service directories.

4.1 Lessons Learned

As pointed out, we understand a service as a function that is well-defined, self-contained, and that does not depend on the context or state of other services—a definition that is in accordance with (7). When implementing a Web-based simulation service, this function-centric view must be complemented with the message-centric semantics of asynchronous transfer: Complex simulation jobs need time and a client may want to start a simulation, but process the results

later. SOAP offers one-way function calls for this purpose: Instead of waiting for a response when invoking a function, a one-way function delivers to the service a request message only; on completion of the request, the server invokes a one-way function on the client side to deliver the results as a response message. However, the demand of asynchronous transfer entails additional problems related to reliability, persistence, and security. Message queuing strategies that guarantee such quality of service issues are not integral part of the SOAP specification but must be seen as an additional layer on top of the SOAP protocol stack. The messaging frameworks from Microsoft and Sun address these issues: they guarantee reliable message delivery by transparently recovering from transmission failures and system crashes (34; 35).

Though Modelica is a powerful standard that comprises nearly all aspects of the future of modeling languages, the acceptance of a simulation service is closely coupled to the interface question. Modern analysis tools pursue the paradigm of *graphic problem formulation*, where analysis tasks are defined by simply drawing a circuit or another model of the interesting technical system. It should be noted that uncausal simulation languages like Modelica render such a graphical problem formulation actually possible. I. e., for the developers of state-of-the-art simulation tools the separation between a graphical problem definition client as frontend and a Web-hosted simulation engine as backend will presumably be the design-paradigm for the next simulator generation.

4.2 Current Developments

In previous work we showed how technology for domain-specific design support, such as case-based retrieval, automated model synthesis, or diagnosis model construction can be operationalized (11). Currently, we are working on Web-based versions of software that originated from this research. We are convinced that simulation-related applications with SOAP interfaces render powerful engineering applications possible, which are composed of—what we call—"high-level" Web-services. From the industry's point of view this may lead, among others, to new hosting and licensing models.

Other research activities are related to performance and standardization issues:

- Complex simulation tasks need time to be solved, and in this connection it is necessary to schedule simulation jobs on a server. Since advanced scheduling algorithms work on the basis of processing time estimations, we search for heuristics to estimate the model processing time within certain bounds.
- In a joint project with the developers of YANOS we are analyzing the idea of a MIME type for technical documents containing Modelica models that

shall be simulated and analyzed over the World Wide Web with a mouse click.

References

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

[2] R. Kilgore, 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, 2002, pp. 841–846.

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

[4] H. Elmqvist, S. Mattsson, M. Otter, 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, 1999, pp. 630–639.

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

[6] B. Stein, Engineers Don't Search, in: W. Lenski (Ed.), Symposium on Logic versus Approximation, Schloss Dagstuhl, Germany, Vol. 3075 LNCS of Lecture Notes in Computer Science, Springer, 2003, pp. 120–137.

[7] Service Architecture Team, Service-oriented architecture (SOA) definition, http://www.service-architecture.com/web-services/articles/service-orien%ted_architecture_soa_definition.html (2005).

[8] Martin Keen, Amit Acharya, Susan Bishop, Alan Hopkins, Sven Milinski, Chris Nott, Rick Robinson, Jonathan Adams, Paul Verschueren, Patterns: Implementing an SOA Using an Enterprise Service Bus, IBM Redbooks, IBM, 2004.

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

[10] B. Stein, How Case-Based Methods Can Automate Fluidic Circuit Design, Proceedings of the Fluid Power Network International PHD Symposium (FPNI 00), Technical University of Hamburg Harburg, Germany, 2000, pp. 137–148.

[11] B. Stein, Model Construction in Analysis and Synthesis Tasks, Habilitation, Department of Computer Science, University of Paderborn, Germany (Jun. 2001).
URL http://ubdata.uni-paderborn.de/ediss/17/2001/stein/

[12] B. Sleeper,

http://www.stencilgroup.com/ideas_scope_200106wsdefined.html (2001).
[13] R. Srinivasan, RFC 1831: Remote Procedure Call (RPC), http://www.rfc-editor.org (1995).
[14] Microsoft Corporation, The Distributed Component Object Model (DCOM), http://www.microsoft.com/com/default.mspx (1994).
[15] Sun Microsystems, Remote Method Invocation (RMI), http://java.sun.com/products/jdk/rmi (1997).
[16] OMG, The Common Object Request Broker Architecture (CORBA), http://www.corba.org (1997).
[17] D. Box, 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 (2000).
[18] R. Hoffman, Sneaking Up On CORBA: The Race for the Ideal Distributed Object Model, http://www.networkcomputing.com/1009/1009f2.html (1999).
[19] J. Postel, J. Reynolds, RFC 959: File Transfer Protocol (FTP), http://www.rfc-editor.org (1985).
[20] R. Presuhn, RFC 3416:Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP), http://www.rfc-editor.org (2002).
[21] M. Crispin, RFC 3501: Internet Message Access Protocol Version 4rev1 (IMAP), http://www.rfc-editor.org (2003).
[22] DYNAST Development Team, DYNCAD, http://icosym.cvut.cz/dyncad/applet (2003).
[23] H. Mann, M. Ševčenko, Simulation and Virtual Lab Experiments across the Internet, in: Proceedings of the International Conference on Engineering Education, Valencia, Spain, 2003, pp. 5265–5272.
[24] DYNAST Development Team, DYNAST Collection of Solved Examples, http://icosym.cvut.cz/dyn/examples (2003).
[25] W3C Consortium, SOAP Version 1.2 W3C Recommendation, http://www.w3.org/TR/soap12-part1/ (2003).
[26] Modelica Association, Modelica™—A Unified Object-Oriented Language for Physical Systems Modeling: Tutorial, Modelica Association, Linköping, Sweden (2000).
[27] Modelica Association, The Modelica Specification, version 2.0, Modelica Association, Linköping, Sweden (2000).
[28] H. Nilsson, J. Peterson, P. Hudak, Functional hybrid modeling, in: Proceedings of PADL'03: 5th International Workshop on Practical Aspects of Declarative Languages, Springer Verlag LNCS 2562, 2003, pp. 376–390.
[29] J. Dormand, Numerical Methods for Differential Equations, CRC Press, New York, London, Tokyo, 1996.
[30] E. Hairer, G. Wanner, Solving Ordinary Differential Equations II. Stiff

and Differential-Algebraic Problems, second edition Edition, Springer, Berlin Heidelberg New York, 1996.

[31] L. Petzold, A Description of DASSL, a Differential-Algebraic System Solver, Scientific Computing (1983) 65–68.

[32] H. Elmqvist, M. Otter, F. Cellier, 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, 1995, pp. xxiii–xxxiv.

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

[34] Microsoft Corporation, The Microsoft Message Queue (MSMQ), `http://www.microsoft.com/windows2000/technologies/communications/msmq/d%efault.mspx` (2005).

[35] Sun Microsystems, The Sun Java System Message Queue, `http://www.sun.com/software/products/message_queue/index.xml` (2005).