Universität Paderborn

Fakultät für
Elektrotechnik, Mathematik und Informatik

Studienarbeit

Konzeptionierung einer Web-basierten Simulationsanwendung für Modelica-Beschreibungen

Daniel Karuseit

vorgelegt bei
PD Dr. Benno Maria Stein
&
Prof. Dr. Stefan Böttcher

Zusammenfassung

Mit dieser Studienarbeit soll das Design einer Web-basierten Simulationsanwendung vorgestellt werden. Da der Server schon existiert und ein Client entwickelt werden soll, gibt es die beiden Schwerpunktsthemen Präsentation und Kommunikation der Simulationsergebnisse. Zum einen wird die Diskussion geführt, in welcher Form die Ergebnisse dem Nutzer präsentiert werden sollen. Da es sich um eine Browserbasierte Lösung handeln soll, werden entsprechende Realisierungsalternativen vorgestellt. Dabei zeigte sich, dass ein Java-Applet die beste Lösung ist. Zum anderen wird besprochen, wie der Client mit dem Server kommunizieren soll. Dabei werden die Vorteile eines Web-Service-basierten Ansatzes (hier im Sinne der Nutzung von SOAP,WSDL und UDDI) gegenüber anderen Verfahren zur verteilten Kommunikation diskutiert. Die diskutierten Ergebnisse werden schließlich umgesetzt, und es wird eine Design-Studie für eine Implementierung erstellt.

Inhaltsverzeichnis

1 Einleitung

Die AG „Wissensbasierte Systeme“ der Universität Paderborn hat, in Kooperation mit der ArtSystems Software GmbH[1], mit FluidSIM ein Programm zur Simulation fluidischer Systeme entwickelt. Es handelt sich dabei um ein komplettes Paket zur Erstellung und Simulation von hybriden Systemen. Die Modelle werden in der Spezifikationssprache Modelica beschrieben. Das Programm existiert in einer Desktop-Version und ist bei entsprechend komplizierten Modellen sehr Ressourcen-intensiv. Es existiert ein Server, der sich ausschließlich mit Simulationsaufgaben beschäftigt und der sehr performant ist.

Deshalb entstand die Idee, diesen Dienst interessierten Nutzern anzubieten, die es sich nicht leisten können – oder für die der Aufwand nicht lohnt – einen eigenen Computer für Simulationszwecke anzuschaffen. Es soll über das Internet, genauer gesagt im Browser, möglich sein, Modelica-Modelle auszuwählen und zu simulieren. Der Client kümmert sich dann ausschließlich um die Anzeige der Ergebnisse, die Ressourcen werden auf dem entfernten Computer belegt. Die Parameter der Simulation sollen angegeben werden können, und die Simulationsergebnisse sollen dem Benutzer als Kurven-Diagramm angezeigt werden. Außerdem soll es möglich sein, eine aktive Simulation zu pausieren. Damit gewährleistet wird, dass die Web-basierte Applikation die Ergebnisse genauso schnell präsentieren kann wie die Desktop-Version, ist es nötig, die Latenzzeiten für die Datenübertragungen gering zu halten. Denn schon während der Simulation werden Ergebnisse produziert, so dass ein einmaliges Übertragen der Daten nach Abschluss der Simulation die Möglichkeiten der Anwendung – Präsentation von Simulationsergebnissen vor Abschluss der gesamten Simulation und Unterbrechen einer Simulation – nicht ausnutzen würde. Es sollte daher ein Web-Service geschaffen werden, der die Funktionen der Desktop-Version implementiert. Die Simulationsparameter sollen über den Browser definiert werden können, auf keinen Fall über eine externe Textdatei. Besonders hervorzuheben ist die Kompatibilität zu Modelica, es gibt keine eigene proprietäre Sprache zur Beschreibung der Modelle. Einem Nutzer soll eine intuitive Nutzung des Simulators ermöglicht werden. Außerdem sind Lizenz-Modelle denkbar, nach denen per Simulation abgerechnet wird. Für den Anbieter bietet dieser Ansatz den Vorteil, dass Dienste modular miteinander verknüpft werden können. Beispielsweise könnte ein Dienst fertige Modelle zur Verfügung stellen, ein anderer behandelt die Abrechnung und ein weiterer stellt die Simulationsfunktionalität zur Verfügung.

[1] http://www.artsystems.de

Meyer zu Eissen und Stein [1] geben weitere Beispiele, welchen Nutzen ein Web-Service hat:

- Man benötigt einen Internetzugang, und damit stehen einem alle nötigen Mittel zur Simulation eines Modells zur Verfügung, ohne den Installationsaufwand von Desktop-Versionen. Ebenso ist es denkbar, dass ein Editor zur Erzeugung von Modellen integriert wird.
- Um Plattform-grenzen zu überwinden, brauchen bestehende Simulationstechnologien nicht portiert oder neu implementiert werden, wenn sie als Web-Service zur Verfügung gestellt werden. Web-Services sind plattformunabhängig.
- Aus kommerzieller Sicht werden neue Lizenzierungsmodelle möglich, die auch für Privatpersonen oder kleinere Firmen erschwingliche Dienste ermöglichen. Komplette Systeme zur Simulation sind sehr teuer. Neben den Kosten für die Hardware fallen auch Lizenzgebühren an. Für Dymola von DynaSim[2] sind dies zur Zeit z.B. etwa 6.400,- Euro pro Einzelplatzlizenz. Simulink von The MathWorks[3] kostet etwa 3.500,- Euro, hierzu kommt allerdings noch die Lizenz für das benötigte Matlab.

In dieser Studienarbeit wird ein geeignetes Szenario entwickelt, wie dieser Web-Service umgesetzt werden kann. Dabei werden folgende Probleme diskutiert:

- Welche Art von Web-basierter Anwendung sollte eingesetzt werden?
- Wie wird die Kommunikation realisiert?
- Welchen Anforderungen muss ein Client für dieses System genügen?
- Was gilt es für die Sicherheit zu beachten?

Erst wenn diese Fragen geklärt sind, kann eine endgültige Design-Entscheidung getroffen werden.

Die Arbeit gliedert sich in folgende vier Abschnitte:

[2] http://www.dynasim.se
[3] http://www.mathworks.de

1. **Einleitung**: Dieser Abschnitt enthält die Motivation für die Studienarbeit. Des Weiteren werden die Aufgabenstellung, gegebene Voraussetzungen und Ziele beschrieben.

2. **Technologische Grundlagen**: Hier werden mehrere Diskussionen geführt. Es muss entschieden werden, wie die Simulationsdaten dem Nutzer angezeigt und aufbereitet werden. Dabei kommen verschiedene Lösungen in Frage, die Server-seitig oder Client-seitig orientiert sein können. Des Weiteren wird gezeigt, auf welche Art Daten über das Internet kommuniziert werden können. Hierbei kommen sowohl proprietäre Lösungen, als auch vorhandene Middleware-Applikationen in Frage. Besonderes Augenmerk wird auf die Technologie der Web-Services gelegt.

3. **Design-Entscheidung**: Nachdem im vorherigen Abschnitt die grundlegenden Möglichkeiten diskutiert wurden, soll ein Lösungsvorschlag ausgearbeitet und vorgestellt werden. Dazu wird eine Anforderungsdefinition erstellt. Anschließend wird deren Realisierung beschrieben.

4. **Fazit**: Abschließend wird ein Fazit über das gesamte Projekt gezogen. Dabei wird erläutert, welche Herausforderungen aufgetreten sind. Es wird ein Ausblick gegeben, wie dieses Projekt weitergeführt werden kann, dazu werden die aktuellen Entwicklungen im Bereich Web-Services vorgestellt. Außerdem wird ein Fazit über das gesamte Projekt gezogen.

2 Technologische Grundlagen

Neben einer Präsentation der Historie der Web-basierten Simulation wird geklärt, welche Konzepte zur Verfügung stehen, um Dienste im Netz anzubieten. Des Weiteren soll dargestellt werden, welche Szenarien es für Web-Anwendungen gibt bzw. welches davon am geeignetsten ist, den Simulationsservice auf das Netz zu bringen. Es werden mögliche Wege der Kommunikation in verteilten Systemen diskutiert, und im Fazit wird dargestellt, aus welchem Grund die Wahl auf ein Java-Applet fiel, das SOAP zur Kommunikation verwendet.

2.1 Historie Web-basierte Simulation

1996 erwähnte Fishwick [2], dass sich das Internet immer neuen Anwendungsfeldern öffnet. Insbesondere zählt er auf, welche Möglichkeiten sich für Simulationsanwendungen ergeben. Dabei fokussiert sich seine Betrachtung auf drei Anwendungsfälle: 1. Ausbildung und Lehre, 2. Veröffentlichungen und 3. Simulationsprogramme. Für die Erstgenannten sieht er den größten Vorteil in der zusätzlichen Verfügbarkeit von Daten per Internet. Wo vorher Disketten oder Magazine verteilt werden mussten, können die Daten nun über das Netz erreicht werden. Für Simulationsprogramme nennt Fishwick neben der permanenten Verfügbarkeit zwei weitere Argumente: Zum einen können Firmen vorab Modelle ihrer Produkte anbieten, so dass Endkunden diese im Vorfeld simulieren können, um sich von deren Qualiät zu überzeugen. Zum anderen sieht er eine mögliche hohe Kostenersparnis, da Web-basierte Simulation durch Werbung günstig angeboten werden kann. Fishwick denkt dabei an Werbeeinblendungen bei der Präsentation der Ergebnisse, wie es beispielsweise von Google schon betrieben wird.

Auch Lorenz et al. [3] haben 1997 erwähnt, dass das Internet eine große Bereicherung und Erleichterung für die Anbieter von Simulationsdiensten bringen wird. Ihr Hauptargument besteht darin, dass durch die Verfügbarkeit per Internet zum einen Plattformunabhängigkeit gegeben ist und zum anderen die permanente Verfügbarkeit des Dienstes gewährleistet werden kann. Jeder potenzielle Nutzer kann, sofern er einen Internet-Zugang besitzt, auf den Dienst zugreifen. Sie beschreiben dabei drei Realisierungsmöglichkeiten für Web-basierte Simulationen: 1. eine CGI-basierte Lösung: Die Daten werden dabei per Webform an den Server übermittelt; sobald die Simulation beendet ist, wird eine neue HTML-Seite generiert und so dem Benutzer das Ergebnis

angezeigt. 2. Der Client lädt ein Java-Applet vom Server herunter, das den Simulationscode enthält. 3. Wieder lädt der Client ein Applet vom Server herunter. Die Simulation findet aber entfernt statt und das Applet dient dazu, die Ergebnisse anzuzeigen und Parameter zu verändern. Die dritte Idee haben sie wiederum von Berger und Leiner [4]).

1998 erweiterte Page [5] die drei Anwendungsfälle von Fishwick um weitere zwei. Damit existieren folgende fünf Anwendungen für Web-basierte Simulation: 1. Simulationen als Zusammensetzung von diversen Medien (Videos, Bilder, Musik, Text). 2. Neue Forschungsmethoden durch größere Verbreitung von Modellen und Artikeln durch das Internet. 3. Web-basierter Zugriff auf Simulationen (per CGI und Applets). 4. Verteiltes Modellieren und Simulieren. 5. Simulation des WWW (für Performance-Analysen und Optimierungen). Des Weiteren erwähnt Page, dass auf Grund der High Level Architecture Initiative [6] des Department of Defense der Interoperabilität von Programmen eine deutlich größere Funktion beigemessen wird. Das bedeutet, dass vorhandene Programme miteinander agieren sollen, nicht ausschließlich durch Wiederverwendung von Code, sondern auch zur Laufzeit.

Veith et al. [7] zeigen die Vorzüge von objektorientierten Simulationsansätzen auf, im Speziellen von Java-Applets. Als besonders vorteilhaft wird hervorgehoben, dass auf Java basierende Web-basierte Simulationen eine hohe Verfügbarkeit haben. Sie finden durch Plattformunabhängigkeit eine weite Verbreitung und können auch außerhalb normaler Öffnungszeiten genutzt werden.

Kuljis und Paul [8] diskutierten 2000, wie es mit Web-basierten Simulationen weitergehen wird. Seit Fishwick 1996 das erste Mal darüber gesprochen hat, gab es einen großen Hype um das Thema. 1998 und 1999 war die Kombination von Java und CORBA der aktuelle Stand der Dinge. Danach gab es jedoch keine großen Neuerungen mehr auf Konferenzen. Die dominante Programmiersprache im Umfeld der Web-basierten Simulation ist Java, dabei beziehen sie sich auf Kilgore et al. [9, 10]. Im Weiteren stellen sie einige in Java geschriebene Pakete zur Web-basierten Simulation vor. Kuljis und Paul sind der Meinung, dass die vorhandenen Anwendungen Machbarkeitsstudien sind, die zeigen sollen, dass es möglich ist, Simulationen auf das Web zu bringen. Sie kritisieren, dass es keine Simulationsanwendungen gibt, die von Nutzern gefordert werden. Page et al. [11] stellen die Frage, ob es sich bei Web-basierter Simulation um eine Evolution oder um eine Revolution handelt. In ihrem Artikel skizzieren sie dabei die öffentlich geführte Diskussion. Eine Antwort auf ihre Frage geben sie jedoch nicht. Zumindest lässt sich

sehen, dass es keine einheitliche Meinung zu dem Thema gibt.

2002 setze sich Kilgore [12] damit auseinander, wie man Web-Services und Web-basierte Simulationen verbinden kann. Er schlägt eine Brücke zwischen dem Konzept der Web-Services und den Web-basierten Simulationen. Die größten Vorteile in der Verbindung sieht er einmal darin, dass Simulationen Industrie-Standards benötigen und Web-Services diese bieten. Außerdem sieht er sie darin, dass Web-Services eine gute Interoperabilität von Simulationsanwendungen bedeuten.

Bei der Winter Simulation Conference 2003 – viele der bisherigen Beiträge wurden auf den vorherigen WSCs vorgestellt – gab es keine Sparte mehr für die Web-basierte Simulation.

2.2 Konzepte

Es gibt verschiedene Möglichkeiten einen Dienst auf das Internet zu bringen. Deswegen folgt hier eine Vorstellung von verschiedenen Konzepten bzw. Möglichkeiten:

- Web-basierte Anwendungen: Unter einer Web-basierten Anwendung versteht man eine Anwendung, die per Web-Browser ausgeführt werden kann. Turau [13] stellt verschiedene Konzepte zur Realisierung Web-basierter Anwendungen vor. Sie werden unterteilt in Client- und Server-seitige Ansätze. Mehr dazu in Sektion 2.2.1 (Seite 7).

- Web-Services: Nach Borghoff und Schlichter [20] gilt: „Unter Service verstehen wir eine Software-Einheit, die bestimmte Dienste erbringt. Sie läuft auf einer oder mehreren Maschinen ab.“ Demzufolge handelt es sich bei einem Web-Service um eine Software-Einheit, die bestimmte Dienste per Web erbringt. In den Medien und ebenso in der Fachliteratur hat es sich etabliert, genau dann von einem Web-Service zu sprechen, wenn die Technologien SOAP, UDDI und WSDL zum Einsatz kommen. Mehr dazu in Sektion 2.2.2 (Seite 10).

- Kommunikation: Wenn eine Anwendung auf einen Service per Internet zugreifen soll, müssen Daten ausgetauscht werden. Dazu stehen verschiedene Mechanismen der Kommunikation zur Verfügung. Die bekanntesten sind COM/DCOM von Microsoft, Java RMI von Sun, CORBA von der OMG und SOAP vom W3C. Zusätzlich kommen noch pro-

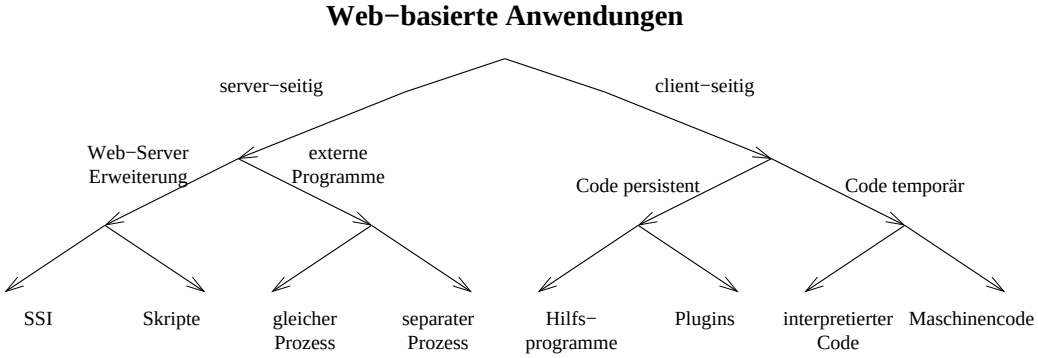

Abbildung 1: Klassifikation Web-basierter Anwendungen [13].

prietäre Lösungen in Frage, bei denen ein eigenes Protokoll definiert und z.B. über reine TCP-Verbindungen realisiert wird. Mehr dazu in Sektion 2.2.3 (Seite 11).

2.2.1 Web-basierte Anwendungen

Turau [13] stellt fest, dass das WWW weitestgehend auf Client-Server Architekturen beruht. Deswegen teilt er Web-Anwendungen in zwei Sparten ein: Client-seitig und Server-seitig. Client-seitige-Anwendungen übernehmen einige Aufgaben um den Server zu entlasten, wohingegen Server-seitige-Anwendungen den Browser ausschließlich zur Ein-/Ausgabe benutzen. Im Folgenden werden beide Ansätze erläutert. Zur Einteilung siehe auch Abbildung 1.

2.2.1.1 Client-seitige-Anwendungen Sie zeichnen sich dadurch aus, dass sie im Vergleich zu Server-seitigen Anwendungen weniger Server-Last, aber oft mehr Netz-Last erzeugen. Die Ursache hierfür liegt in der Idee der Client-seitigen Anwendung begründet. Client-seitige Anwendungen sollen einige Aufgaben des Servers übernehmen, um diesen zu entlasten. Somit entsteht weniger Server-Last. Da die Ergebnisse der abgenommenen Aufgaben dem Server mitgeteilt werden müssen, entsteht mehr Netzlast. Turau unterscheidet zwischen persistent und temporär gespeicherten Anwendungen.

Persistent gespeicherte Anwendungen Einmal existieren vom Browser unabhängige Hilfsprogramme, die sich automatisch öffnen, sobald eine ent-

sprechende Datei im Browser ausgewählt wird[4]. Des Weiteren gibt es Plugins, diese integrieren sich automatisch in den Browser und belegen dort einen festen Bereich, der in der HTML-Seite vorgegeben ist. Wichtig ist, dass es sich bei den Plugins um dynamisch ladbare Module handelt, die die Funktionalität des Browsers auf eine transparente Art erweitern.

Beide Arten von persistent gespeicherten Anwendungen haben gemein, dass sie einmal auf dem Client installiert werden müssen und dann solange genutzt werden können, bis ggf. eine neuere Version zur Verfügung steht. Sie müssen dementsprechend für jede Ziel-Plattform programmiert werden und sind somit plattformunabhängig.

Temporär gespeicherte Anwendungen Diese Form der Client-seitigen-Anwendungen zeichnet sich dadurch aus, dass der benötigte Programmcode einmal auf den Client-Rechner geladen und dann nach der Sitzung wieder gelöscht wird (Wobei es natürlich möglich ist, dass er in einem Cache aufbewahrt wird und so bei erneuter Anwendung ggf. schneller zur Verfügung steht). Es gibt dabei verschiedene Möglichkeiten, in welcher Form der Programmcode übertragen werden kann, z.B. als Maschinencode oder als interpretierbarer Code.

Der Vorteil von Maschinencode liegt darin, dass kein Interpreter benötigt wird. Dadurch kann der Code schneller ausgeführt werden als interpretierter. Da nativer Code vorliegt, ist diese Vorgehensweise jedoch nicht plattformunabhängig, da für jede Ziel-Plattform entsprechender Maschinencode vorliegen muss. Ein typischer Vertreter dieser Variante ist z.B ActiveX. Bei interpretiertem Code wird ein spezifischer Interpreter benötigt, der für sämtliche Ziel-Plattformen zur Verfügung stehen muss. Liegt der zu interpretierende Code dabei in ASCII-Form vor, so spricht man von Skripten. Ein Beispiel für interpretierbaren Code, der nicht in ASCII-Form vorliegt, ist Java-Bytecode, wie ihn z.B. Java-Applets verwenden. Um diesen Code ausführen zu können, benötigt der Client einen Interpretierer, der den Bytecode ausführt. Deser Interpretierer heißt Java-Virtual-Machine (JVM). Um die Geschwindigkeit zu steigern, gibt es so genannte Just-in-time-Compiler, die den Bytecode vor der Ausführung in Maschinencode übersetzen. Der große Vorteil von Java-Applets besteht darin, dass ein umfangreiches Sicherheitskonzept zur Verfügung steht, das den Benutzer vor Gefahren schützt. Das so genannte Sandbox-Konzept verhindert, dass Java Applets

[4]Ein Beispiel hierfür ist Ghostview.

- lesend/schreibend auf lokale Dateien zugreifen
- andere Verbindungen öffnen als zu dem Host, von dem sie geladen wurden
- Verbindungen auf privilegierten Portnummern akzeptieren
- Benutzer bezogene System-Eigenschaften lesen
- Haupt-Fenster ohne Warnhinweise erzeugen
- externe Programme ausführen
- System-Bibliotheken laden
- die virtuelle Maschine beenden.

2.2.1.2 Server-seitige-Anwendungen Sie zeichnen sich dadurch aus, dass es kaum mehr Interaktionsmöglichkeiten gibt als das Klicken von Hyperlinks oder dem Ausfüllen von Variablen-Feldern. Dadurch ergibt sich, dass dem Client keine weiteren Aufgaben zukommen, als Dokumente anzuzeigen. Turau unterscheidet hier zwischen Web-Server-Erweiterungen und externen Programmen.

Web-Server-Erweiterungen Die Web-Server-Erweiterungen können unterteilt werden in Server-Side-Includes (SSI) und Skripte. Standen einst nur statische Webseiten zur Verfügung, können mit diesen Mechanismen bei jedem Abruf dynamisch Elemente (komplette Seiten oder Teile einer Seite) erzeugt werden. SSI stellen dabei die einfachste Art von Anwendungen dar, da nur einzelne Anweisungen in eine Webseite eingebettet werden können. Skripte sind mächtiger, da sie auch aus Programmcode einer Programmiersprache bestehen können (JavaScript, Java, VBScript). Der Web-Server stellt in diesem Fall einen Interpreter zur Verfügung, der die in die HTML-Dokumente eingefügten Programme interpretiert und die Ausgabe in die Dokumente einfügt.

Für beide Formen der Server-Erweiterungen gilt, dass sie einfach zu handhaben und dementsprechend schnell zu erlernen sind. Sie werden direkt in HTML-Dokumente eingebettet und der Server entscheidet anhand der Datei-Erweiterung, ob der Inhalt des Dokuments auf Anweisungen analysiert wird. Da das Analysieren der Dokumente Zeit benötigt, wird durch diese Form der

dynamischen Inhalts-Erstellung immer die Auslieferung von Dokumenten an den Nutzer verzögert und außerdem der Server belastet, da ein entsprechender Interpreter bei jeder Anweisung gestartet werden muss. Bei sehr vielen Anfragen kann dies zu einer kompletten Auslastung des Servers führen.

Externe Programme Die externen Programme können noch einmal unterteilt werden in Common Gateway Interface (CGI) basierte Programme, Server APIs und Servlets. Unter einem Gateway versteht man ein Programm, das einen Zugriff auf eine Informationsquelle realisiert. Ein CGI ist nun der Mechanismus zur Kommunikation zwischen Gateway und Web-Server. Dabei wird für jede Anfrage ein neuer Prozess gestartet, der Eingabedaten über die Standard-Eingabe oder per Umgebungsvariablen erhält. CGI-Anwendungen können in jeder Programmiersprache realisiert werden. Am weitesten verbreitet sind C/C++, Perl und Tcl. Unter Server APIs versteht man dynamische Bibliotheken, die bei der ersten Anfrage geladen werden und dann so lange wie möglich im Speicher verbleiben. Dadurch entfällt der Aufwand der Prozess-Erzeugung und die Effizienz wird somit gesteigert. Es gibt z.B. die Internet Server API (ISAPI) von Microsoft oder die Apache API. Nachteilig ist einerseits, dass Anwendungen von Server APIs im gleichen Prozess-Raum wie der Web-Server laufen und diesen somit zum Absturz bringen können, und andererseits, dass Anwender an die Programmiersprache der Server API gebunden sind. Bei Java Servlets handelt es sich um eine in Java geschriebene Server API von Sun Microsystems. Ein HTML-Dokument enthält dabei den Servlet-Code oder referenziert ein Servlet über eine URL. Das Servlet kann dabei auch auf einem anderen Rechner zur Verfügung gestellt werden. Der Server führt dann das entsprechende Servlet aus und ersetzt die Referenz durch die Ausgabe des Servlets. Die Java Klassen werden dynamisch geladen und bleiben dann im Speicher, insofern gleichen Servlets den dynamisch ladbaren Bibliotheken. Im Gegensatz zu Server APIs sind als Servlet realisierte Anwendungen plattformunabhängig. Servlets benötigen zwar eine Java-Laufzeitumgebung, jedoch keinen speziellen Web-Server.

2.2.2 Web-Services

Ein Web-Service ist ein Dienst, der nicht von einem Menschen, sondern von einem Softwaresystem konsumiert werden soll. Der Web-Service wird dabei genutzt um automatisiert Daten auszutauschen oder Funktionen auf entfernten Rechnern aufzurufen. Die Beschreibung des Dienstes kann dabei in

einer zentralen Registrierung (Registry) hinterlegt werden. Kossmann und Leymann [14] identifizieren die folgenden Schlüsseleigenschaften eines Web-Services:

- Ein Web-Service wird durch einen Uniform Resource Identifier [15] identifiziert.
- Die Schnittstelle eines Web-Services ist maschinenlesbar und wird durch die Web Service Definition Language (WSDL) [16] beschrieben. Diese Beschreibung geschieht dabei in XML.
- Ein Web-Service kommuniziert mit anderen Softwarekomponenten mit Hilfe des Simple Object Access Protocols (SOAP) [17]. Die Nachrichten sind ebenfalls in XML beschrieben. Der Nachrichten-Austausch kann insbesondere mit Hilfe von Internet-Protokollen (z.B. HTTP oder SMTP) stattfinden.
- Da Web-Services autonom sind, lässt sich nicht beeinflussen, ob und wie eine Nachricht von einem Web-Service verarbeitet wird. Es ist somit auch nicht möglich, Antwort-Garantien oder Laufzeit-Garantien zu geben. Dies muss auf andere Weise geschehen, gegebenenfalls mit einem Kontrakt zwischen Anbieter und Nutzer eines Dienstes.

Es lässt sich also sagen, dass eine WSDL-Datei in einer zentralen Registrierung hinterlegt wird. Dort kann ein potenzieller Kunde sich nach Dienst-Anbietern umsehen. Bei dieser zentralen Registry handelt es sich um eine Implementierung der OASIS Spezifikation, der Universal Description Discovery and Integration (UDDI) [18]. Hierin wird spezifiziert, wie Informationen über Web-Services publiziert und abgerufen werden sollten. Nachdem ein Kunde sich für einen Anbieter entschieden hat, bindet er sein Programm zunächst an den Service des Anbieters, bevor dann ein Nachrichtenaustausch per SOAP stattfindet.

2.2.3 Kommunikation

Je nachdem, für welche Art von Web-Anwendung man sich entscheidet, kommen unterschiedliche Möglichkeiten zur Kommunikation in Frage. Aus diesem Grund folgt ein Überblick über die gängigen Methoden zur Kommunikation im Internet. Die Informationen stammen dabei von Münchhausen [23], sowie Meyer zu Eissen und Stein [1].

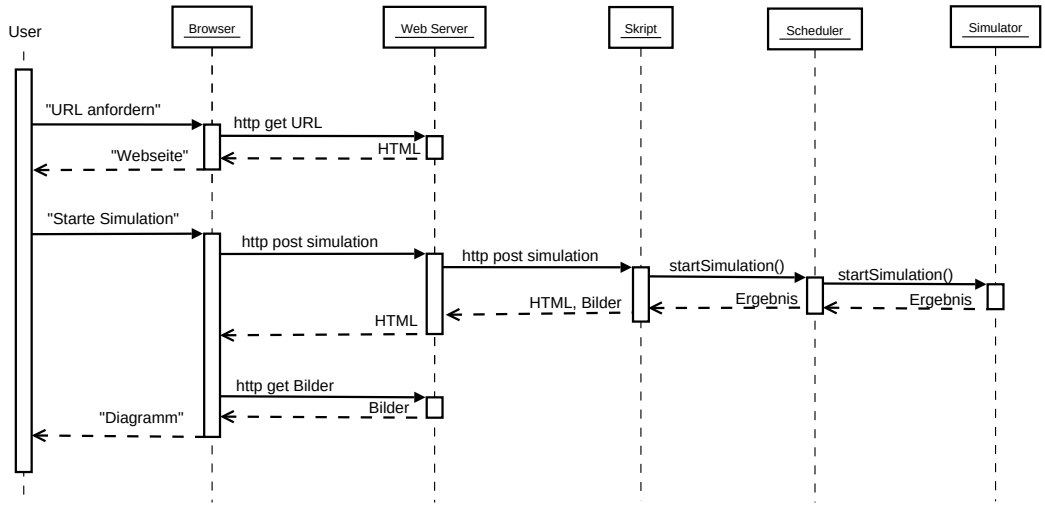

Abbildung 2: Kommunikation per http/HTML [1].

Hypertext Transfer Protocol/HTML Ein Großteil der Dokumente im World Wide Web (www) wird per Hypertext Transfer Protocol (http) übertragen. Bei http handelt es sich um ein Text-basiertes Protokoll, das für jede Anfrage eine Verbindung aufbaut. Innerhalb dieser Verbindung gibt es eine Anfrage, genannt http-request, und die zugehörige Antwort, den http-response. Alle HTML-basierten Web-Anwendungen benutzen dieses Protokoll zur Kommunikation. Abbildung 2 stellt den Ablauf der Kommunikation eines HTML-basierten Simulationsdienstes vor. Nachdem ein Benutzer die URL des Dienstes eingegeben hat, bekommt er eine Webseite angezeigt auf der ein Modell zur Simulation ausgewählt werden kann. Wenn der Benutzer die Parameter der Simulation eingestellt und bestätigt hat, werden sie per http POST versendet. Die Daten werden von einem Skript verarbeitet, das den Scheduler startet und das Modell zur Simulation übergibt. Nachdem die Simulation beendet ist, wird das Ergebnis an den Benutzer zurückgesendet. Diese Form der Simulation wurde von Mann und Ševčenko [19] beschrieben.

COM/DCOM Microsoft hat mit seinem Object Linking and Embedding (OLE) eine Schnittstelle geschaffen, um Dokumententeile per Copy and Paste zwischen Anwendungen zu tauschen. Durch wachsende Funktionalität und zunehmender Kommunikation zwischen Anwendungen wurde aus OLE das Component Object Model (COM). Diese Funktionalität sollte nun auch auf verteilte Rechner erweitert werden, was zu Distributed COM (eben DCOM) führte. Ein großer Nachteil war anfangs die ausschließliche Verfügbarkeit von

COM/DCOM für die Microsoft Windows Plattformen. 1996 wurden Microsoft und die Software AG[5] Technologie-Partner, und im Rahmen dieser Partnerschaft entwickelte die Software AG Portierungen von DCOM auf andere Plattformen. Inwiefern diese stabil und zuverlässig funktionieren oder ob es Performanz-Einbußen gibt, ist nicht bekannt. Des Weiteren gibt es COM-CORBA-Bridges, durch die es möglich ist, per CORBA auf COM-Objekte zuzugreifen.

Java RMI Die Java Remote Method Invocation (oder kurz Java RMI) wurde von der Firma SUN[6] zur einfacheren Kommunikation zwischen verteilten Java-Objekten entwickelt. Üblicherweise wird zwischen einem Server und einem Client unterschieden. Der Client greift dabei auf Funktionen des Servers zurück. Für einen Programmierer gestaltet sich der Zugriff auf das entfernte Java-Objekt durch Java RMI wie der Zugriff auf ein lokales Objekt. Insbesondere muss der Programmierer sich nicht um das Versenden von Nachrichten kümmern, dies geschieht transparent durch Java RMI. Dies wird dadurch ermöglicht, dass auf dem Client ein Stellvertreterobjekt existiert. Dieses Stellvertreterobjekt bietet dieselben Funktionen an wie das Objekt auf dem Server. Die Stellvertreter verdecken somit den Übertragungsvorgang. Sowohl Client als auch Server müssen ein Java-Interface implementieren, das die Methoden des entfernten Java-Objekts beschreibt.

Problematisch kann es werden, wenn der Server mit einer Klasse konfrontiert wird, die der entfernen Java Virtual Machine (JVM) bisher nicht bekannt sind. Diese unbekannte Klasse muss von der JVM dynamisch nachgeladen werden, dafür gibt es den RMI-Klassen-Lader java.rmi.RMIClassLoader. Diesem ist dabei egal, woher die Klassen kommen. Sie können im Klassen-Pfad liegen, im aktuellen Verzeichnis oder auf einem Web-Server. Eine ausführliche Beschreibung des Mechanismus findet sich in der Java-Dokumentation [24].

Java ist plattformunabhängig, wodurch sich mit Java RMI eine plattformunabhängige Kommunikationsmöglichkeit eröffnet. Wie der Name sagt, ist Java RMI stark von Java abhängig. Es gibt allerdings die Möglichkeit, über das Java Native Interface nativen Code in Java aufzurufen. Dadurch wäre es möglich, einer nicht in Java geschriebenen Anwendung Java RMI beizubringen, allerdings zu Ungunsten der Plattformunabhängigkeit von Java.

[5] http://www.softwareag.com
[6] http://www.sun.com

CORBA Die Common Object Request Broker Architecture (CORBA) ist 1999 von der Object Management Group[7] (OMG) entwickelt worden. Bei CORBA handelt es sich, im Gegensatz zu den bisher vorgestellten Verfahren, um einen Ansatz zur Kommunikation in verteilten Systemen, der explizit darauf ausgelegt wurde, plattform-, sprach- und herstellerunabhängig zu sein.

Es handelt sich also um ein Meta-Konzept, das in die verschiedensten Programmiersprachen übersetzt worden ist und das auf sehr vielen Plattformen zur Verfügung steht. Für die Kommunikation zwischen Client und Server ist auf beiden Seiten ein Object Request Broker (ORB) verantwortlich, der bei Anfragen transparent das entsprechende Objekt sucht und die Anfrage an dieses weiterleitet bzw. umgekehrt den Empfänger einer Antwort ausfindig macht. Im Gegensatz zu Java RMI ist es bei CORBA nicht möglich, eigenen Programmcode über das Netz zu versenden.

Die Kommunikation läuft über das Internet Inter-ORB Protocol (IIOP) und kann sowohl über TCP als auch über andere Protokolle abgewickelt werden (z.B. http, was besonders sinnvoll erscheint, sollte eine der kommunizierenden Parteien hinter einer Firewall sitzen). Allerdings ist das HTTP-Tunneling bisher nicht sehr weit verbreitet, obwohl es eine entsprechende Spezifikation der OMG gibt [25].

SOAP Das Simple Object Access Protocol (SOAP) hat vor allem im Zusammenhang mit Web-Services auf sich aufmerksam gemacht. Entwickelt wurde es vom World Wide Web Consortium[8] (W3C) im Rahmen der Web-Services Activity. Aktuell ist die SOAP Spezifikation 1.2 [26].

SOAP ist in seiner Grundfunktion ein zustandsloses Kommunikations-Paradigma, das Nachrichten nur in eine Richtung versendet. Es ist für Anwendungen aber möglich, komplexere Kommunikations-Patterns zu realisieren (z.B: request/response, request/multi-response, etc.). Es legt fest, wie Nachrichten auszusehen haben und wie sie verarbeitet werden. Wie CORBA ist SOAP plattform-, sprach- und herstellerunabhängig. SOAP basiert auf XML, d.h. Nachrichten werden in XML codiert versendet und empfangen.

Eine SOAP Nachricht besteht dabei immer aus einem Envelope, optional einem oder mehreren Headern und genau einem Body. Der Envelope ist das Wurzel-Element des XML-Dokuments. Im Body können auch Fehlermeldun-

[7] http://www.omg.org
[8] http://www.w3c.org

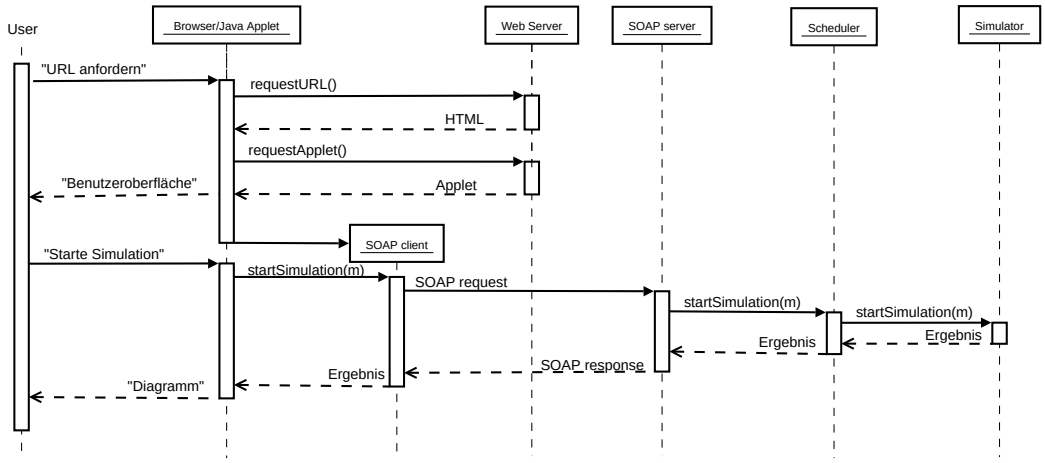

Abbildung 3: Kommunikation per SOAP [1].

gen enthalten sein. In SOAP ist also ein Mechanismus zum Versenden von Fehlermeldungen vorhanden und es muss nicht, wie z.B. bei http, über Status Codes laufen.[9] In SOAP können theoretisch alle beliebigen Datentypen codiert werden, die per XML beschreibbar sind. Es liegt hier beim Entwickler, keine Datentypen zu beschreiben, die nicht in den Ziel-Programmiersprachen existieren. Damit eine Applikation SOAP-Nachrichten empfangen und versenden kann, muss sie in der Lage sein, XML-Dokumente zu verarbeiten und zu generieren.

Die Kommunikation per SOAP funktioniert so, dass ein Client eine Anfrage an einen Server stellt. Dabei kodiert eine lokale SOAP-Engine den Methoden-Aufruf in eine entsprechende SOAP-Nachricht. Diese wird vom Server dekodiert und die zugehörige Funktion wird aufgerufen. Anschließend wird das Ergebnis von der SOAP-Engine des Servers kodiert und an den Client versendet, der es nun benutzen kann.

Abbildung 3 veranschaulicht die Kommunikation eines SOAP-basierten Simulationsdienstes, der ein Java Applet als Benutzeroberfläche zur Verfügung stellt. Nachdem das Applet gestartet wurde instanziiert es den SOAP-Client und zeigt die Benutzeroberfläche. Wenn der Benutzer eine Simulation startet wird von dem SOAP-Client eine Nachricht generiert und zum SOAP-Server gesendet.

SOAP definiert, wie Nachrichten verarbeitet werden sollen, und definiert in diesem Zusammenhang so genannte Zwischenempfänger (Intermediaries).

[9] Wie z.B die Error 404 Fehlermeldung, wenn eine Webseite nicht verfügbar ist.

Abbildung 4: Kommunikation mit Zwischenempfänger.

Diese können die Nachricht auf dem Weg vom Sender zum Empfänger bearbeiten, ihre Informationen bekommen sie dabei aus dem Header der Nachricht. Dabei können beliebig viele solcher Zwischenempfänger definiert werden. Sobald ein Zwischenempfänger die Nachricht bearbeitet hat, entfernt er den relevanten Header aus der Nachricht und sendet sie an den nächsten Empfänger weiter. Ein Zwischenempfänger kann dabei immer nur den für ihn bestimmten Header bearbeiten. Außerdem können zusätzlich Zwischenempfänger hinzugefügt werden. Abbildung 4 zeigt den Kommunikationsfluss bei zwei Zwischenempfängern.

Ein weiterer Vorteil von SOAP ist, dass es sehr flexibel einsetzbar ist: Mit der Erweiterung SOAP Messages with Attachments [27] lassen sich zum Beispiel binäre Dateien per SOAP-Message versenden, ohne erst in XML kodiert werden zu müssen. Es gibt auch keine vorgeschriebene Bindung an ein bestimmtes Protokoll. Die Nutzung von SOAP ist mit jedem Netzwerk-Protokoll denkbar.[10].

Proprietär Unter proprietäre Protokolle zur Kommunikation fällt alles das, was nicht standardisiert ist. Es handelt sich dabei um Eigenentwicklungen, die nicht darauf ausgelegt sind, allgemeine Lösungen zu werden. Diese Lösungen für den Hausgebrauch sind sehr beliebt, wenn Standards zu viel Overhead erzeugen würden oder es generell darauf ankommt, sehr performant zu sein. In einem Bandbreitenbeschränkten-Netzwerk wäre es z.B. denkbar, ein sehr spezialisiertes Protokoll zu haben, das nur für eine bestimmte Art von Kommunikation benutzt werden kann. Dadurch würde diese Anwendung effizient kommunizieren können. Hierbei muss allerdings jede an der Entwicklung beteiligte Person die Protokoll-Spezifikation ausgehändigt bekommen.

2.3 Fazit

In diesem Kapitel wurde die historische Entwicklung der Web-basierten Simulation vorgestellt. Anschließend wurden verschiedene Konzepte präsen-

[10]Beispielsweise smtp für asynchrones versenden von Nachrichten, die nicht sofort ausgeliefert werden müssen

tiert, wie aus einer Anwendung eine Web-basierte Anwendung erstellt werden kann. Dabei ist der Begriff der Web-Services eingeführt worden. Außerdem wurden einige Möglichkeiten zur Kommunikation in verteilten Anwendungen erläutert. Dies geschieht, um ein geeignetes Szenario für den vorhandenen Simulationsservice zu finden. Dazu werden im Folgenden zunächst die Anforderungen an den zu erstellenden Dienst beschrieben und dann die vorgestellten Verfahren auf Verwendbarkeit diskutiert.

2.3.1 Anforderungen

Die Anforderungen an den Simulationsservice setzen sich zusammen aus den technischen Anforderungen sowie den Usability-Anforderungen.

Technische Anforderungen Hierbei handelt es sich um technische Anforderungen, die an die Realisierung gestellt werden:

(T1) Echtzeit: Die Simulation soll in Echtzeit geschehen bzw. angezeigt werden.

(T2) Plattformunabhängigkeit: Der Service soll auf jeder Plattform und lediglich mit einem Web-Browser benutzt werden können.

(T3) Erweiterbarkeit: Es sollten eigene Clients entwickelt werden können, wenn dies gewünscht ist.

(T4) Sprachunabhängigkeit: Durch die Wahl der Technologie sollte sich nicht auf eine einzige Programmiersprache beschränkt werden. Es wäre gut, wenn eigene entwickelte Clients in dieser Hinsicht keinen oder kaum Beschränkungen unterliegen.

(T5) Lizenzierung: Die Einbindung eines geeigneten Lizenzierungsmodells soll problemlos möglich sein. Dies soll auch nachträglich ohne größeren Aufwand veränderbar sein.

(T6) Aktualisierung: Der Client soll problemlos aktualisiert werden können. Es wäre wünschenswert, wenn dazu nichts installiert werden muss.

(T7) Authentisierung: Langfristig sollen die Nutzer sich anmelden, um den

Dienst kommerziell nutzen zu können.

(T8) Sicherheit der Übertragung: Die Daten sollen auf dem Weg von Nutzer zum Dienst gesichert sein.

(T9) Sicherheit des Servers: Der Server soll vor dem Zugriff durch Unbefugte geschützt sein.

(T10) Sicherheit des Clients: Ebenso soll der Client vor unbefugtem Zugriff geschützt sein.

Usability-Anforderungen Hierbei handelt es sich um Anforderungen, die mit der Bedienbarkeit des Clients zusammenhängen. Dabei kann es sich um spezielle Anforderungen an den Client oder an die Einfachheit der Bedienung handeln:

(U1) Grafische Oberfläche: Kurven-Darstellung von Ergebnissen, mit automatischer Skala-Erzeugung

(U2) Vergleich von mehreren Simulationsergebnissen: Dies sollte möglich sein, ohne mehrere Instanzen des Browsers geöffnet zu haben.

(U3) Optionen: Simulationsoptionen sollen an zentraler Einstellung und separat für jede Simulation eingestellt werden können.

(U4) Bedienbarkeit: Der Client soll intuitiv bedienbar sein und mit möglichst wenig Bedienelementen auskommen. Es soll vor allem viel Platz für die Ergebnis-Darstellung vorhanden sein.

(U5) Schneller Aufbau: Damit potenzielle Interessenten sich den Client auch ansehen, sollte er schnell zu laden sein.

(U6) Eingabe-Validierung: Vom Nutzer eingegebene Daten sollen auf Korrektheit überprüft und gegebenenfalls Korrekturvorschläge gemacht werden.

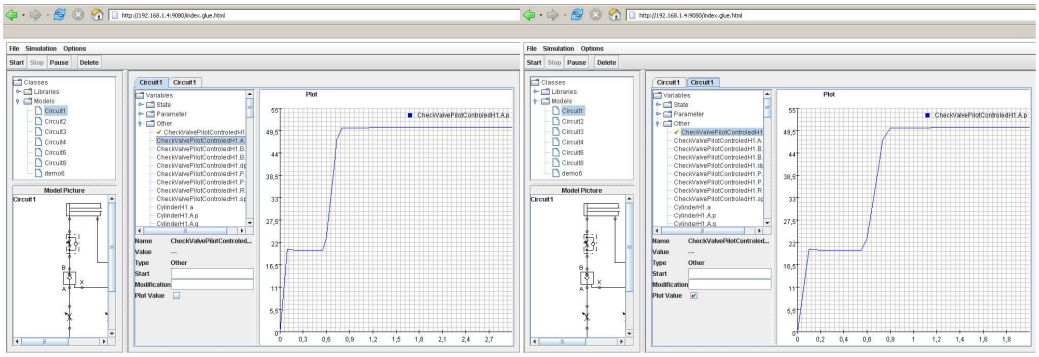

Abbildung 5: Register-Karten in einem Java Applet.

2.3.2 Web-Anwendung

Aus den in in Sektion 2.2.1 vorgestellten Technologien zur Realisierung von Web-Anwendungen muss nun die für dieses Projekt geeignetste ausgewählt werden. Da Plattformunabhängigkeit gefordert ist, fallen viele der Client-seitigen-Anwendungen heraus, speziell die Hilfsprogramme und die Plugins. Diese sind nicht immer für alle Plattformen verfügbar oder bedürfen regelmäßiger Wartung. Ein eigenes zu entwickeln kommt noch viel weniger in Frage, der Installationsaufwand soll niemandem zugemutet werden. ActiveX fällt heraus, da es lediglich auf Windows unterstützt wird bzw. es nicht klar ist, wie gut die Portierung der SoftwareAG ist.

Somit bleiben noch die Server-seitigen-Anwendungen, die alle auf HTML-Dokumenten basieren, und die Java-Applets auf der Client-Seite. Da Münchhausen dies so auch schon herausgearbeitet hat folge ich weiterhin seinem Beispiel und vergleiche die beiden Varianten mit Hilfe von Tabelle 1 (auf Seite 20), die auch auf die Anforderungsdefinitionen aus Sektion 2.3.1 verweist.

Die Entscheidung, ein Java-Applet zur Realisierung zu benutzen, ist nahe liegend und hängt vor allem von den Usability Anforderungen ab. Mit Java Applets, speziell den Swing-Klassen, ist schnell eine komplexe grafische Oberfläche erstellt (U1), die es vor allem ermöglicht mehre Simulationsergebnisse in einer Browser-Instanz zu vergleichen (U2). Dazu können verschiedene Register-Karten benutzt werden, ein Äquivalent in HTML gibt es hierzu nicht. Abbildung 5 veranschaulicht den Vergleich von zwei unterschiedlichen Simulationsergebnissen per Register-Karte.

	HTML-Dokument	JAVA Applet
(T1) Echtzeit	(−−) Nicht realisierbar	(+) Realisierbar
(T2) Plattformunabhängigkeit	(++) Vollkommen unabhängig	(+) Java-Plugin für alle Plattformen verfügbar
(T6) Aktualisierung	(++) Nach aktualisieren der Seite geschehen	(+) Alter Code kann im Cache sein, ansonsten automatisch
(U1) Grafische Oberfläche	(-) Interaktivität sehr begrenzt	(++) Mit Swing auch komplexe Anforderungen realisierbar
(U2) Vergleich von Ergebnissen	(−−) In einer Browser-Instanz nicht zu realisieren	(++) Diverse Möglichkeiten verfügbar
(U5) Schneller Aufbau	(+) Gegeben, hängt von der Realisierung ab	(+) Gegeben, nachdem das Applet heruntergeladen ist
(U6) Eingabe Validierung	(-) Begrenzt möglich	(++) Problemlos realisierbar

Tabelle 1: Vergleich HTML-Dokumente - Java Applet

2.3.3 Kommunikation

Nachdem nun geklärt ist, dass für die Realisierung der Anwendung Clientseitig ein Java-Applet erstellt wird, muss noch die Frage behandelt werden, wie das Applet mit dem Web-Service kommmunizieren soll. Von den in Sektion 2.2.3 vorgestellten Technologien fallen COM/DCOM und Java RMI auf Grund der technologischen Kriterien der Plattformunabhängigkeit (T2) und der Sprachunabhängigkeit (T4) heraus. COM/DCOM beschränkt sich weitestgehend auf Windows-Plattformen, wobei wieder einmal auf die nicht näher bekannte Portierung der SoftwareAG verwiesen werden muss. Die erwähnten COM-CORBA-Bridges könnten langfristig eine Alternative darstellen. Java RMI beschränkt sich, wenn man von dem Umweg über JNI einmal absieht, ausschließlich auf Java-Anwendungen.

Aus Gründen der Erweiterbarkeit der Anwendung bieten sich CORBA und SOAP an. Für beide Technologien gilt, dass sie plattform- und sprachunabhängig sind. Für CORBA finden sich in Java schon eigene Bibliotheken, für SOAP gibt es derzeit nur das Java Web Services Development Pack oder andere externe Bibliotheken. Langfristig wird jedoch eine Integration von SOAP in Core-Java erwartet.

Ich habe mich für den Einsatz von SOAP als Kommunikationsmittel entschieden. Dies begründet sich neben dem reinen Interesse an einer neuen Technologie damit, dass im Hinblick auf die Erweiterbarkeit eine langfristige Lösung vorliegt. Ich denke dabei an die Möglichkeit von SOAP, die Nachrichten an Zwischenempfänger zu senden. Damit wäre es möglich, einem externen Anbieter den kompletten Geschäftsprozess der Zugriffsprüfung und Abrechnung aufzuerlegen ohne zusätzlichen Nachrichten-Overhead zu erzeugen. Die Nachricht wird bei dem entsprechenden Web-Service vorbeigeschickt. Des Weiteren spechen die Tatsachen, dass viele große Anbieter den Einsatz von SOAP vorantreiben und dass mit dem W3C eine sehr große Organisation hinter der Standardisierung von SOAP steht, für den Einsatz dieser Technologie.

3 Design-Entscheidungen

Nachdem in Abschnitt 2 diskutiert wurde, dass Client-seitig ein Java-Applet eingesetzt werden soll, das per SOAP auf die Funktionen des Simulators zurückgreift, muss nun das konkrete Design des Applets bestimmt werden. Dazu ist es zunächst nötig, die an das Applet gestellten Anforderungen zu definieren. Deswegen werden die beteiligten Akteure an Hand von Use-Case-Diagrammen vorgestellt. Nachdem die Anforderungen vorgestellt wurden, wird die Realisierung mit Hilfe von Klassen-Diagrammen sowie einem Deployment-Diagramm beschrieben. Hierbei werden die Begriffe der Unified Modeling Language (UML) [28] benutzt.

3.1 Anforderungsdefinition

Bei den Anforderungen muss unterschieden werden zwischen Server-seitigen und Client-seitigen Anforderungen. Dies liegt in der Tatsache begründet, dass Server-seitig der Simulator sämtliche Berechnungen vornimmt. Der Simulator liegt dabei als Windows-DLL vor. Der Client nimmt die Benutzereingaben entgegen, verifiziert sie und bereitet die Simulationsergebnisse grafisch auf.

Server

Der Server besteht aus drei Teilen. Zum einen dem Simulator, der Funktionen zur Verfügung stellt, um ein vorhandenes Modell zu simulieren und die Werte der Simulation zu übermitteln. Da es möglich ist, verschiedene Modelle nebeneinander zu simulieren, ist es auch möglich, die Simulationsparameter für jedes Modell separat einzustellen. Eine Übersicht über die Funktionalität des Simulators gibt Abbildung 6. Des Weiteren beinhaltet der Server einen Web Server, der das Java Applet bereitstellt. Zur Kommunikation zwischen Applet und Simulator beinhaltet der Server eine SOAP-Schnittstelle.

Client

Aufgabe des Clients ist es, dem Benutzer ein Frontend zur Verfügung zu stellen, um den Simulator zu benutzen. Das heißt, Modelle, die auf dem Server liegen, müssen geladen werden können. Wobei laden in diesem Zu-

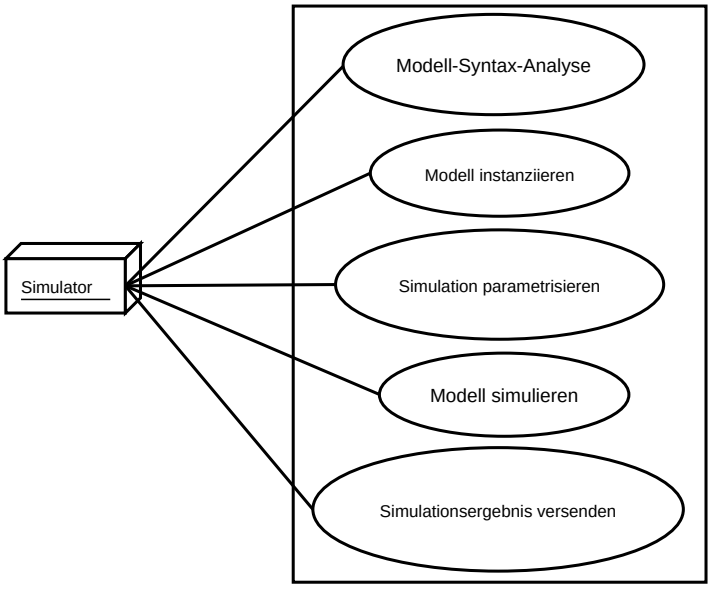

Abbildung 6: Use-Case des Simulators.

sammenhang instanziieren bedeutet. Jeweils eine Instanz eines Modells kann simuliert werden. Dabei ist es durchaus denkbar, dass ein Modell mehrmals instanziiert wird, z. B. um das Simulationsergebnis bei verschiedenen Parametern zu vergleichen. Neben dem Instanziieren und dem Starten, Stoppen und Pausieren einer Simulation soll es über den Client möglich sein, die Simulationsergebnisse grafisch anzuzeigen. Dies soll in Form eines Kurven-Diagramms geschehen. Der Client ist dabei für die Anzeige verantwortlich, vom Simulator bekommt er lediglich die anzuzeigenden Daten übermittelt. Eine übersicht über die Funktionalität des Clients gibt Abbildung 7. Des weiteren benötigt der Client eine Soap-Schnittstelle zur Kommunikation mit dem Simulator

3.2 Realisierung

Nachdem die Anforderungen nun definiert sind, können wir zur Realisierung kommen. Zunächst wird erläutert, wie der Simulator zum Web-Service konvertiert wurde. Danach wird der Client vorgestellt. Abbildung 8 stellt das Deployment-Diagramm des Services dar.

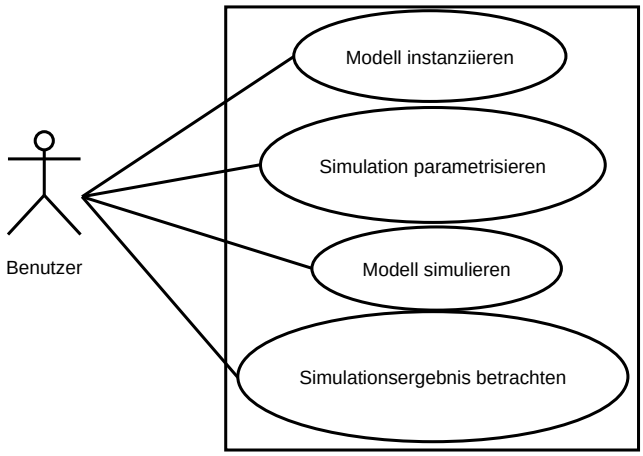

Abbildung 7: Use-Case des Clients.

Server

Für den als Windows-DLL realisierten Server wurde ein Java-Wrapper geschrieben. Durch dessen Hilfe konnte der Simulationsserver mittels der Web-Service-Bibliothek webMethods Glue [29] online gestellt werden. Die Bibliothek stellt sowohl einen http-Server als auch einen SOAP-Server zur Verfügung. Demzufolge wird außer der DLL, dem Java-Wrapper und der Glue-Bibliothek nichts weiter benötigt, damit der Server funktioniert. Des Weiteren generiert Glue bei jedem Start des Services eine WSDL-Datei, die über den http-Server abrufbar und somit immer auf dem aktuellsten Stand ist.

Client

Das Problem der Anwendung besteht darin, SOAP-Nachrichten zu parsen. Es wird also ein Parser benötigt. Ein solcher Parser ist in der Biliothek Glue enthalten. Glue beinhaltet zudem eine sehr praktische Funktionalität zur Implementierung eines Clients für einen Web-Service: Ein Programm namens wsdl2java. Dieses Programm generiert mit Hilfe einer WSDL-Datei Stubs. Dadurch kann auf das entfernt liegende Simulations-Objekt zugegriffen werden, als wenn es lokal vorliegen würde. Die gesamte Kommunikation, die per SOAP durchgeführt wird, wird dabei vor dem Programmierer versteckt und läuft vollkommen transparent ab. Dem Programmierer wird also automa-

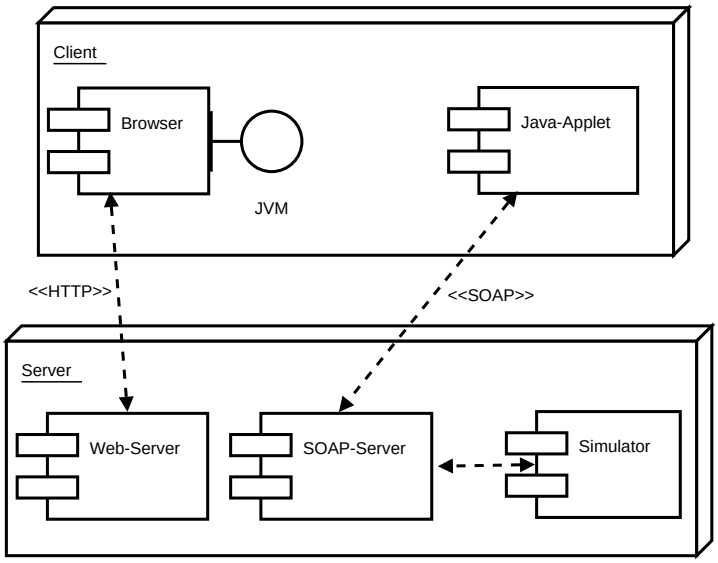

Abbildung 8: Deployment-Diagramm des Simulationsservices.

tisch ein Protokoll generiert, um das er sich nicht zu kümmern hat. Da diese Aufgabe komplett automatisiert ist, besteht die Aufgabe daraus, eine GUI zu entwickeln sowie die Simulationsdaten innerhalb dieser zu präsentieren. Zum Abschluss werden die Klassen zusammen mit der Glue-Bibliothek als jar-Archiv verpackt und über den http-Server der Server-Applikation online gestellt. Damit ist der Web-basierte Simulationsservice verfügbar.

3.3 Fazit

In diesem Kapitel wurde beschrieben, welche Schritte nötig sind, um eine Web-basierte Simulationsanwendung zu erstellen. Es wurde dabei davon ausgegangen, dass auf Seiten des Servers bereits eine Anwendung zur Verfügung steht, die aber nicht für den Einsatz im Web konzipiert wurde.

Während der Implementierung sind einige Probleme aufgetreten, über die im Folgenden geschrieben wird. Der Einsatz von Glue bringt eine hohe Automatisierung und versteckt die Kommunikation durch den Einsatz von Stubs. Allerdings erkauft man sich diese Annehmlichkeit damit, dass die Ziel-Anwendung eine enorme Größe von über einem Megabyte erreicht. Dass diese Größe für ein Applet nicht in Frage kommt, versteht sich von selbst. Deswegen wurden Alternativen gesucht. Da das zentrale Problem im parsen von SOAP-

Nachrichten besteht, ist es eine Möglichkeit, einen eigenen SOAP-Parser zu schreiben. Eine andere Möglichkeit bietet die Bibliothek kSOAP [30] von Stefan Haustein et al. Diese Bibliothek beinhaltet einen Parser, hat eine Größe von etwa 50 KB und ermöglicht eine Appletgrösse von ca. 130KB. Erreicht wird dies unter anderem durch den Einsatz eines so genannten Pull-Parsers zur Verarbeitung der SOAP-Nachrichten. Dieser unterscheidet sich folgendermaßen von einem Push-Parser: Anstatt eines Callback-Mechanismus zum Aufruf spezieller Methoden bei Erreichen eines bestimmten Tags wird der Kontrollfluss bei jedem Tag an das aufrufende Programm zurückgegeben. Dieses entscheidet nun, was mit den Daten geschehen soll. Der Pull-Parser behält den aktuellen Zustand bei und startet bei erneutem Aufruf an exakt der Stelle im Dokument, an der vorher abgebrochen wurde. Ein Pull-Parser gleicht damit in seiner Funktionsweise einem Tokenizer, der ein Dokument in einzelne Wörter zerlegt, auf die dann per String-Vergleich eingegangen werden kann. Es folgt ein Beispiel zur Funktionsweise eines Pull-Parsers:

```
Parser parser = new PullParser(input)

public void parse(PullParser parser) throws ValidationException
{
    while(true) {
        int eventType = parser.next();
        if(eventType = parser.START_TAG) {
            String tag = parser.getStartTagName();
            if("this",equals(tag)) {
                //do something for this tag
            } else if("that",equals(tag)) {
                //do something for this tag
            } else {
                throw new ValidationException(
                ünknown field "+tag+" in document");
              }
        } else if(eventType == parser.END_TAG) {
          break;
        } else {
          throw new ValidationException("invalid XML input");
          }
    }
}
```

Leider gibt es für kSOAP bisher noch keine Möglichkeit, den benötigten Code automatisch zu generieren. Außerdem können komplexe Datentypen im Gegensatz zu Glue nicht automatisch serialisiert/de-serialisiert werden. Hier müssen eigene Methoden implementiert werden. Dafür steht mit kSOAP ein sehr funktionaler und vor allem sehr kleiner SOAP-Prozessor zur Verfügung.

Praktische Erfahrungen haben gezeigt, dass die Kommunikation der simulierten Daten ein Flaschenhals ist. Aus diesem Grund wurde ein Mechanismus entwickelt, der die benötigte Kommunikation minimiert. Server-seitig existiert ein Buffer, so dass mit einer SOAP-Nachricht mehrere Ergebnisse übertragen werden können. Der Buffer ist Client-seitig variabel konfigurierbar, so dass im Bedarfsfall auch das häufige Übertragen von neuen Ergebnissen priorisiert werden kann. In diesem Fall ist die Buffergröße auf eins zu setzen. Die Erfahrung hat gezeigt, dass eine Erhöhung der Buffergröße den Kommunikationsoverhead reduziert und der Aufbau der grafischen Darstellung der Ergebnisse hierdurch kaum beeinträchtigt wird. Es gibt also einen Trade-Off zwischen Gesamtdauer der Simulation und Aktualiserung des Kurven-Diagramms.

4 Fazit

Der Haupt-Beitrag dieser Studien-Arbeit ist es, ein Szenario zur Realisierung eines Web-basierten Simulationsdienstes aufzuzeigen. Ein wichtiger Aspekt für diese Applikation ist das Thema der Sicherheit. Darauf wird im Folgenden eingegangen, da es für die Weiterentwicklung der Anwendung von Interesse ist. Des Weiteren traten auch einige Probleme auf, die im Anschluss erläutert werden. Abschließend wird ein Ausblick gegeben, wie die Anwendung in Zukunft verbessert werden kann.

4.1 Sicherheit

Auf das Thema Sicherheit wurde bisher noch nicht eingegangen, es darf aber nicht vernachlässigt werden. Gerade im Hinblick auf eine kommerzielle Nutzung der Applikation ist es wichtig, einen kurzen Exkurs über dieses Thema zu geben. Zum aktuellen Zeitpunkt funktioniert die Anwendung so, dass jeder Nutzer ohne Einschränkung auf sie zugreifen kann. Da es allerdings auch festgelegt ist, welche Modelle simuliert werden können, ist dies ein akzeptabler Zustand. Für die Zukunft ist es wünschenswert, dass ein Benutzer seine Identität bestätigen muss – sich also authentifiziert – damit ihm die erbrachte Leistung z. B. in Rechnung gestellt werden kann. Der einfachste Authentifizierungsmechanismus stellt sicherlich die Eingabe eines Benutzer-Namens in Kombination mit einem Passwort dar. Denkbar wäre auch ein Autorisierungsansatz: Jeder Nutzer darf vorhandene Modelle simulieren, aber nur Autorisierte dürfen z.B. eigene Modelle simulieren oder die Simulationsergebnisse abspeichern.

Ein weiteres Problem liegt in der Tatsache begründet, dass man als Anbieter eines Simulationsservices auch Modelle übermittelt bekommen kann, die der Absender geheim halten möchte. Es besteht also die Notwendigkeit, die übermittelten Daten zu verschleiern, so dass potenzielle Service-Nutzer keine Bedenken haben, diese Modelle zu übermitteln. Vor allem könnte ein im Klartext übermitteltes Modell während der übertragung abgefangen werden. Hierzu sind verschiedene Ansätze denkbar, zwei davon werden hier exemplarisch aufgezeigt:

- **Verschlüsselung**: Damit ist der Einsatz von Kryptographie zum Schutz der ausgetauschten Daten gemeint. Denkbar ist hier zum Beispiel ein RSA basiertes Public-Key-Verfahren.

- **Modell-Verschleierung** Ein Ansatz wäre es, vor der Übermittlung eines Modells die entsprechende Datei zu verschleiern und Variablen-Namen durch sinnlose Strings zu ersetzen. Damit würde zumindest die Rekonstruktion des Modells schwieriger werden.

Da die Problematik der Sicherheit von allgemeinem Interesse ist, gibt es hierzu von der OASIS (Organization for the Advancement of Structured Information Standards) ein eigenes technisches Komitee [31], das sich mit dem Thema Sicherheit von Web-Services befasst und unter dem Namen Web Services Security v1.0 einige Spezifikationen herausgebracht hat: SOAP Message Security 1.0, UsernameToken Profile 1.0 und X.509 Certificate Token Profile. Das W3C arbeitet ebenso an Sicherheits-Standards, namentlich: XML Signature und XML Encryption.

4.2 Probleme

Eines der schwerwiegendsten Probleme ist es gewesen, die Größe des Java-Applets möglichst gering zu halten. Hier wäre es überlegenswert, einen eigenen angepassten, SOAP-Prozessor zu entwickeln. Allerdings ist die mit kSOAP gefundene Alternative sehr viel versprechend, so dass es fraglich ist, ob sich der Aufwand zur Entwicklung eines eigenen Prozessors lohnt.

Die Standardisierung von Konzepten im Bereich Web-Services geht noch recht langsam vonstatten. Ein Grund dafür ist die Tatsache, dass verschiedene Forschungsgruppen an Lösungen für identische Probleme arbeiten. Welches Konzept es schließlich zum Standard schafft, ist im Vorfeld nicht abzusehen. Im nächsten Abschnitt (4.3) werde ich auf einige bereits etablierte Standards näher eingehen.

Da die zu Grunde liegenden Technologien sich noch im Anfangsstadium befinden, ist eine Automatisierung von Aufgaben bisher kaum möglich. Allerdings gibt es immer mehr Tools, die, ausgehend von der WSDL-Beschreibung eines Services, Stubs und Skeletons für die Verwendung generieren. Bisher sind diese noch auf eine spezielle Bibliothek – wie z.B. Apache Axis oder webMethods Glue – beschränkt.

4.3 Ausblick

Das Umfeld der Web-Services wird gerade erst gestaltet. Dies macht sich vor allem daran bemerkbar, dass einige Konzepte erforscht werden, die bisher noch kein offizieller Standard sind. Einige davon könnten für den Simulationsservice von Nutzen sein. Die entsprechenden Konzepte werden kurz vorgestellt und ein möglicher Einsatz skizziert.

WS-Security: Die Spezifikationen zur Sicherheit von Web-Services von OASIS umfasst Web Services Security: SOAP Message Security 1.0, Web Services Security: UsernameToken Profile 1.0, Web Services Security: X.509 Certificate Token Profile 1.0 sowie zwei benötigte XML Schemas [31]. Diese Spezifikationen bringen die technischen Grundlagen für den Einsatz von Sicherheitsfunktionen, wie z.B der Sicherung der Integrität von Nachrichten. Der Inhalt der einzelnen Spezifikationen folgt:

- SOAP Message Security 1.0: Das Ziel dieser Spezifikation ist es, Programmen die nötigen Mechanismen zum sicheren Austausch von SOAP-Nachrichten zur Verfügung zu stellen. Es wird dabei kein spezielles Sicherheitsprotokoll beschrieben, sondern flexible Mechanismen, die sich dafür eignen, eine Vielzahl an Sicherheitsprotokollen zu implementieren. Sie sind darauf ausgelegt zu verhindern, dass Nachrichten zwischen den beteiligten Kommunikationspartnern abgefangen und gelesen oder verändert werden. Da es außerdem denkbar ist, dass eine Nachricht von einem Dritten gefälscht wird, sichern die Mechanismen, dass nur Nachrichten von legitimierten Absendern verarbeitet werden. Um die gesetzten Ziele zu erreichen, werden in der Spezifikation Sicherheitsmarkierungen (Security Tokens) und Signaturen (Signatures) definiert. Sicherheitsmarkierungen beinhalten dabei sicherheitsrelevante Informationen wie z.B: Name, Identität oder Gruppe des Absenders. Signaturen werden dazu benutzt, die Integrität einer Nachricht zu gewährleisten. Außerdem wird ein Mechanismus zur Verschlüsselung vorgestellt, der den XML Encryption [32] Standard einsetzt. Mit diesem Mechanismus kann jeder Block einer SOAP-Nachricht verschlüsselt werden.

- UsernameToken Profile 1.0: Diese Spezifikation beschreibt, wie eine spezielle Sicherheitsmarkierung von einem Web-Service-Benutzer genutzt werden kann, um sich (innerhalb der SOAP-Nachricht) mit einem Benutzer-Namen zu identifizieren und optional zu authentisieren.

- X.509 Certificate Token Profile 1.0: Diese Spezifikation beschreibt, wie

X.509 Zertifikate mit der SOAP Message Security Spezifikation benutzt werden können. Unter einem X.509 Zertifikat versteht man dabei eine Bindung zwischen einem öffentlichen Schlüssel und (mindestens) dem Namen des Nutzers, dem Namen des Ausstellers, einer Seriennummer und einem Gültigkeitszeitraum. Ein X.509 Zertifikat kann benutzt werden, um einen öffentlichen Schlüssel zu verifizieren und eine SOAP-Nachricht so zu authentifizieren oder um den Schlüssel zu identifizieren, wenn die Nachricht verschlüsselt wurde.

WS-Policy Das Web Services Policy Framework [33] ist ein Beispiel für eine Spezifikation, die erst noch bei einem Standardisierungsgremium eingereicht werden muss. Die Spezifikation beschreibt ein allgemeines Modell sowie die benötigte Syntax, um die Richtlinien (Policies) eines Web-Services zu beschreiben. Eine Richtlinie (Policy) beschreibt dabei, unter welchen Voraussetzungen der Anbieter eines Web-Services den Service zur Verfügung stellt. Denkbar ist zum Beispiel eine Richtlinie, die verpflichtend die Nutzung einer bestimmten Sicherheitsmarkierung vorschreibt. Es werden einige Basis-Konstrukte definiert, die von anderen Web-Service Spezifikationen benutzt und erweitert werden können.

WS-SecurityPolicy Diese Spezifikation [34] ist eine Erweiterung der WS-Security. Es wird vorgestellt, wie die in WS-Security vorgestellten Mechanismen als Richtlinien beschrieben werden können.

WS-Reliability Bei Web Services Reliability [35] handelt es sich um ein SOAP-basiertes Protokoll. Es wird benutzt um SOAP-Nachrichten zu senden, die garantiert ankommen, keinesfalls doppelt verschickt werden und bei denen die Ordnung der Nachrichten garantiert beibehalten wird. Es ist definiert als eine Ergänzung des SOAP-Headers und ist unabhängig vom zugrunde liegenden Protokoll.

Literatur

[1] Sven Meyer zu Eissen, Benno Stein: *Web based Simulation: Application Scenarios and realization alternatives*, In Proceedings of the TCME 2004, April 13-17. 2004, Lausanne, Switzerland.

[2] Paul A. Fishwick: *Web-based Simulation: Some personal Observations*, In Proceedings of the 1996 Winter Simulation Conference, pp. 772-779. Society for Computer Simulation International, 1996.

[3] Peter Lorenz, Heiko Dorwarth, Klaus-Christoph Ritter, Thomas J. Schriber: *Towards a web based Simulation environment*, In Proceedings of the 1997 Winter Simulation Conference, 1997.

[4] M. Berger, U. Leiner: *Remote Visualisieren und Manipulieren von Simulationen im Internet*, In Proceedings of Simulation and Animation '97 Magdeburg, 1-11. Society for Computer Simulation Europe, 1997, Ghent, Belgium.

[5] Ernest H. Page: *The rise of web-based Simulation: Implications for the high-level architecture*, In Proceedings of the 1998 Winter Simulation Conference, 1998.

[6] High Level Architecture, *https://www.dmso.mil/public/transition/hla/*, United States Department of Defense, 1996.

[7] Tamie L. Veith, John E. Kobza, C. Patrick Koelling: *World Wide Web-based Simulation*, TEMPUS Publications, 1998, Great Britain.

[8] Jasna Kuljis, Ray J. Paul: *A Review of Web Based Simulation: Whither we wander?*, In Proceedings of the 2000 Winter Simulation Conference, 2000.

[9] R.A. Kilgore, K. J. Healy: *Java, enterprise simulation and the Silk simulation language*, In Proceedings of the 1998 International Conference on Web-Based Modeling & Simulation, ed. P.A. Fishwick, D.R.C. Hill and R. Smith, pp. 195-201, The Society for Computer Simulation International, 1998.

[10] R.A. Kilgore, K. J. Healy, G.B. Kleindorfer: *The future of Java-based simulation*, In Proceedings of the 1998 Winter Simulation Conference, pp. 1707-1712, 1998.

[11] Ernest H. Page, Arnold Buss, Paul A. Fishwick, Kevin J. Healy, Richard E. Nance, Ray J. Paul: *Web-Based Simulation: Revolution or Evolution?*, ACM Transactions on Modeling and Computer Simulation, Vol. 10, No. 1, pp. 3-17, January 2000.

[12] Richard A. Kilgore: *Simulation Web Services with .NET Technologies*, In Proceedings of the 2002 Winter Simulation Conference, 2002.

[13] Volker Turau: *Techniken zur Realisierung Web-basierter Anwendungen*, Informatik-Spektrum 22, Springer, 1999.

[14] Donald Kossmann, Frank Leymann: *Web Services*, Informatik Spektrum 26, Springer, 2004.

[15] T. Berners-Lee, R. Fielding und L. Masinter: *Uniform Resource Identifiers (URI): Generic Syntax*, 1998. IETF RFC 2396

[16] Erik Christensen, Francisco Curbera, Greg Meredith, Sanjiva Weerawarana: *Web Services Description Language (WSDL) 1.1 - W3C Note 15 March 2001*, http://www.w3.org/TR/wsdl

[17] Martin Gudgin, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau, Henrik Frystyk Nielsen: *SOAP Version 1.2 Part 1: Messaging Framework - W3C Recommendation 24 June 2003*, http://www.w3.org/TR/soap12-part1/

[18] OASIS: *UDDI Technical White Paper*, September 6, 2000 , http://www.uddi.org/whitepapers.html

[19] Heřman Mann und Michal Ševčenko: *Simulation and Virtual Lab Experiments across the Internet.*, In Proceedings of the International Conference on Engineering Education, Valencia, Spain.

[20] Uwe Borghoff, Johann Schlichter: *Rechnergestützte Gruppenarbeit - Eine Einführung in Verteilte Anwendungen*, 2. Auflage, Springer, 1998.

[21] Web Service Level Agreement (WSLA) Language Specification, http://www.research.ibm.com/wsla/

[22] M. Tian, A. Gramm, T. Naumowicz, H. Ritter, J. Schiller: *A Concept for QoS Integration in Web Services*, In Proceedings of the Fourth International Conference on Web Information Systems Engineering Workshops (WISEW'03), pp. 149-155, December 13 - 13, 2003, Roma, Italy.

[23] Derk Münchhausen: *Konzeption eines Werkzeugs zur Realisierung von Web-Anwendungen mit Schwerpunkt XML*, Diplomarbeit, TU München, 1999

[24] JavaTM 2 SDK - Standard Edition Documentation Version 1.4.2 - JavaTM Remote Method Invocation (RMI) - Dynamic code downloading using RMI, Sun Microsystems, Inc., http://java.sun.com/j2se/1.4.2/docs/guide/rmi/codebase.html, 2003

[25] CORBA Firewall Traversal Specification, Object Management Group, http://www.omg.org/, 2003

[26] W3C, XML Protocol Working Group, http://www.w3.org/2000/xp/Group/, 2003

[27] John J. Barton, Satish Thatte, Henrik Frystyk Nielsen: *SOAP Messages with Attachments*, W3C Note 11 December 2000, 2000.

[28] OMG Unified Modeling Language™ Specification (UML®), http://www.omg.org/, 2003.

[29] webMethods Glue, http://www.webmethods.com/Solutions/Glue/.

[30] kSOAP, http://ksoap.objectweb.org/.

[31] OASIS, Web Services Security TC, http://www.oasis-open.org/committees/wss/

[32] W3C, XML Encryption Working Group, http://www.w3.org/Encryption/2001/

[33] Siddharth Bajaj, Don Box, Dave Chappell, Francisco Curbera, Glen Daniels, Phillip Hallam-Baker, Maryann Hondo, Chris Kaler, Dave Langworthy, Ashok Malhotra, Anthony Nadalin, Nataraj Nagaratnam, Mark Nottingham, Hemma Prafullchandra, Claus von Riegen, Chris Sharp, John Shewchuk: *Web Services Policy Framework (WS-Policy).*, September 2004.

[34] Giovanni Della-Libera, Phillip Hallam-Baker, Maryann Hondo, Tomasz Janczuk, Chris Kaler, Hiroshi Maruyama, Nataraj Nagaratnam, Andrew Nash, Rob Philpott, Hemma Prafullchandra, John Shewchuk, Elliot Waingold, Riaz Zolfonoon: *Web Services Security Policy (WS-SecurityPolicy)*, Draft 18 December 2002.

[35] OASIS, Web Services Reliable Messaging TC: *WS-Reliability 1.1*, Committee Draft 1.086, 24 August 2004.