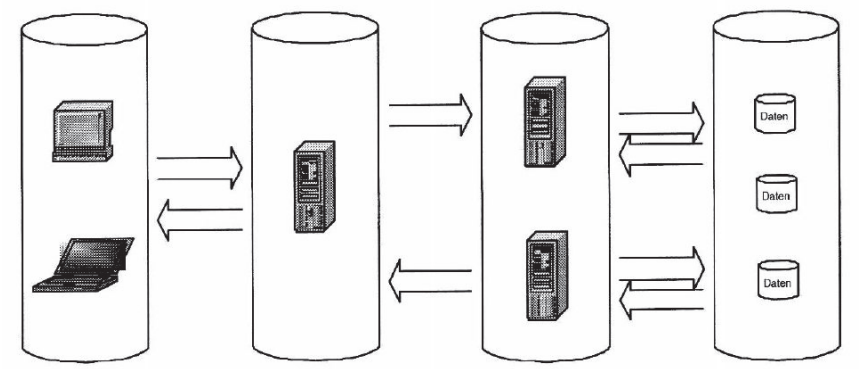
FORTSCHRITTSBERICHT
SIMULATION FBS 13



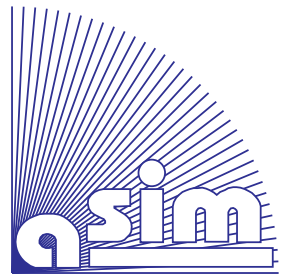
Michael Gyimesi

Simulation Service Providing unter Verwendung von Web Service Technologie

ARGESIM

ISBN Ebook 978-3-903347-13-7
ISBN Print 978-3-901608-63-X

DOI: 10.11128 / fbs.13




ARGESIM

Fortschrittsberichte Simulation

FBS Band 13

Herausgegeben von **ASIM**
Arbeitsgemeinschaft **Sim**ulation, Fachausschuß 4.5 der GI

Michael Gyimesi

Simulation Service Providing unter Verwendung von Web Service Technologie

ARGESIM / ASIM – Verlag, Wien, 2005
ISBN Print 978-3-901608-63-X

Ebook Reprint 2020
ISBN Ebook 978-3-903347-13-7
DOI: 10.11128/fbs.13

ASIM Fortschrittsberichte Simulation / ARGESIM Reports

Herausgegeben von **ASIM**, Arbeitsgemeinschaft Simulation, Fachausschuß 4.5 der **GI** und der **ARGESIM**

Betreuer der Reihe:

Prof.. Dr.-Ing. Th. Pawletta (ASIM)
Hochschule Wismar
Phillip-Müller-Str., 23952 Wismar, Germany
Tel: +49-3841-753-406
Email: pawel@mb.hs-wismar.de

Dr.-Ing. habil. D.P.F. Schwarz (ASIM)
Fraunhofer-Institut für Integrierte Schaltungen
Zeunerstr. 38, 01069 Dresden, Germany
Tel: +49-351- 4640 - 730
Email: schwarz@eas.iis.fhg.de

Prof. Dr. F. Breitenecker (ARGESIM / ASIM)
Abt. Simulationstechnik, Technische Universität Wien
Wiedner Hauptstraße 8 - 10, A - 1040 Wien
Tel: +43-1-58801-10115
Email: Felix.Breitenecker@tuwien.ac.at

FBS Band 13

Titel: Simulation Service Providing unter Verwendung von Web Service Technologie

Autor: Michael Gyimesi
Michael.Gyimesi@tuwien.ac.at

Begutachter des Bandes:

Prof. Dr. F. Breitenecker, TU Wien; Prof. Dr. A. Javor, Univ. Budapest

ARGESIM / ASIM – Verlag, Wien, 2005
ISBN Print 978-3-901608-63-X

Ebook Reprint 2020 (Scan)
ISBN Ebook 978-3-903347-13-7

DOI: 10.11128/fbs.13

Abstract

The worldwide globalization and the rapidly changing consumer demands challenge the e-Business market and the convergence of simulation and the technology of the internet is part of research in the simulation domain since 10 years. As Simulation Service Providing plays a bigger role in the field of business to business applications for every year, it is of major interest how the new technologies can fit in a scenario of Simulation Service Providing.

In the domain of webtechnology XML, Web Services, Grid Computing and Semantic Web are the main research areas of all big software companies and organisations concerned with the development of webstandards. These technologies have the capability to be used in the field of distributed simulation and to provide Quality of Service and security standards for the provision of simulation services.

This dissertation investigates the use of XML and follow up technologies in the field of Simulation Service Providing. A framework, which combines the development in simulation techniques and the Web Service technology and is capable of providing simulation services via internet, was developed.

The study is split up into four parts with eleven chapters and three appendices. First the meaning and implications of Simulation Service Providing and the scope of this work is defined more precisely. This is

followed by a short introductory overview about modelling and simulation techniques and a survey of relevant internet technologies. In the third part the SimASP Framework including the used concepts is presented and Descrete Event Simulation examples are defined to validate the developed framework. The work ends up with some conclusions and remarks and an outlook for future extensions of the framework as well as the future development of simulation techniques in respect of web-technologies. The three appendices include a glossary, the definitions of used XML schemes and the definitions of the validating examples.

Danksagung

An dieser Stelle möchte ich mich bei allen, die mich während meines Lebens begleitet, unterstützt und motiviert haben, bedanken.
Hervorheben will ich meine Frau, Freundin und Lebensgefährtin Uli Ziegler und meine Familie, ohne deren Unterstützung ich diese Arbeit wahrscheinlich nicht begonnen und sicher nicht zu Ende gebracht hätte.
Außerdem möchte ich besonders meinem Betreuer Univ.Prof. Felix Breitenecker für seine Unterstützung und Kooperation an der TU Wien und meinem Praktikanten Alexander Schirrer, der einen großen Teil der Programmierarbeit geleistet hat, danken.

Inhaltsverzeichnis

Abbildungsverzeichnis

Vorwort

> *"I think there is a world market for maybe five computers."*-
> Thomas Watson, chairman of IBM, 1943

Dieses Zitat von Thomas Watson aus den Anfängen des Computerzeitalters zeigt, dass damals die Bedeutung und das Potenzial des Computers wohl noch nicht richtig eingeschätzt wurde. Aber auch Anfang der 90er Jahre, als die Vernetzung der Computer mittels Internet eine bereits bekannte und ausgereifte Technologie war, kannte kaum jemand Tim Berners-Lee's World Wide Web (WWW) geschweige denn wurde die Bedeutung, die das WWW heutzutage besitzt, vorhergesehen. Mittlerweile ist das Suchen und der Austausch von Daten über das Internet Allgemeingut. Diese Aktionen werden aber noch immer fast ausschließlich von Menschen ausgeführt, was zu einem großen Teil an der zugrunde liegenden Technologie und Konzeption liegt.
Durch die Einführung von XML (eXtended Markup Language) deren Bedeutung Anfangs nicht als derart wesentlich eingestuft wurde, stehen wir vor einer neuen Stufe der Internetrevolution: Die Perspektive des Internets als einen großen Informationsraum, in dem es keinen Unterschied macht, ob Daten von einem (menschlichen) Benutzer oder von einem anfordernden Programm manipuliert werden.

Parallel zur Entwicklung des Internets fand eine Entwicklung im Bereich der Applikationsprogramme statt. Obwohl Distributed Computing angewendet wird, seit es Computernetze gibt, werden erst seit kurzer Zeit Applikationen in größerem Umfang entwickelt, die scheinbar als ein Programm, aber darunterliegend durch Vernetzung vieler kleiner Recheneinheiten funktionieren. Auch dieser Fortschritt wurde auf Grund der Entwicklung neuer XML-basierter Protokolle ermöglicht, die für Maschine-Maschine-Interaktion entwickelt wurden. Die wichtigsten Protokolle neben XML sind SOAP, WSDL und UDDI.

Im Bereich der Simulation, die es seit den späten 50er Jahren gibt, gewinnt der Bereich der *Webbasierten Simulation* und des *Simulation Service Providing* (SSP - Simulation als Application Service Providing) an Bedeutung.
Das Ziel der vorliegenden Arbeit war nun, diese Entwicklung der Simulationstechnik und die neuen Internettechnologien, die man unter dem Schlagwort *Web Services* zusammenfassen kann, zu verbinden und weiters zu untersuchen, inwieweit XML und darauf aufbauende Technologien geeignet sind, Simulation Service Providing zu unterstützen. Es wurde ein Framework - das *SimASP* Framework-entwickelt, mit Hilfe dessen SSP mittels Web Service Technologie angeboten werden kann.

Die Arbeit ist folgendermaßen strukturiert:
Im ersten Teil, wird die Aufgabenstellung definiert und in einem Abschnitt über Application Service Providing im Allgemeinen und Simulation Service Providing im Speziellen wird das Thema der Arbeit definiert und abgegrenzt.
Im darauf folgenden Grundlagenteil wird zuerst eine kurze Einführung in die Simulation samt deren Entwicklung im Bereich der Webbasierten Simulation gegeben und es werden die für diese Arbeit benötigten Inter-

nettechnologien eingeführt.

Im dritten Teil wird das SimASP Framework mitsamt der verwendeten technischen Lösungen und Konzepte dargelegt. Es wird zuerst eine allgemeine Darstellung des Frameworks gegeben. Dann wird das Environment beschrieben, das die Client-Server Kommunikation mittels einer serviceorientierte Grid Infrastruktur ermöglicht. Danach wird das XML Schema generischer *Discrete Event Simulation* (DES)-Modelle präsentiert und es wird erläutert, inwieweit XML als Beschreibung solcher Simulationsmodelle geeignet ist. Abschließend werden die Simulatoren und die Schnittstellen der Modelldefinition zu den Simulatoren präsentiert, die für das SimASP Framework dieser Arbeit eingesetzt wurden. Damit lassen sich alle wesentlichen Schritte der Prozesskette, die von der Inanspruchnahme von Diensten durch einen Klienten bis zur Abarbeitung der Simulationsanforderung durch den Service Provider und dem Übersenden von Outputinformation reichen, nachvollziehbar.

Der vierte Teil behandelt Beispiele der DES, anhand derer die Funktion des Frameworks validiert wurde. Um die Flexibilität der generischen Beschreibung von DES-Modellen zu zeigen, werden die Beispiele sowohl mittels Eventgraphen als auch in ressourcenorientierter Darstellung modelliert.

Den Abschluss bildet eine Zusammenfassung der Ergebnisse dieser Arbeit und es wird ein Ausblick in die zukünftige Entwicklung des Frameworks und die weitere Entwicklung des Simulation Service Providings gegeben.

Im Anhang befindet sich ein Glossar und Verzeichnis der verwendeten Abkürzungen, die verwendeten XML Schema Definitionen, die Darstellung der Beispielmodelle in XML-Form und ein Literaturverzeichnis.

Teil I

Einführung

Kapitel 1

Simulation Service Providing

Die Konvergenz der Simulation mit der Technologie des Internets und im Besonderen des WWW ist bereits seit ca. 10 Jahren Teil der Forschung und Entwicklung im Bereich der Simulation (siehe Kapitel 2.9).
In der kommerziell eingesetzten Simulation gewinnt der Begriff des Simulation Service Providings immer mehr an Bedeutung [67, 107, 110]. Simulation Service Providing ist dabei ein Begriff aus dem Bereich der Business to Business (B2B)-Simulation und bedeutet Simulationsdienste als Application Service Providing (ASP) über das Internet anzubieten.

1.1 Motivation und Definition von ASP

Application Service Providing ist eine Dienstleistung, die Anwendern die Nutzung von Software-Lösungen über das Internet oder andere

Netze ermöglicht.
Hersteller von Simulationssoftware erwirtschaften ihren Gewinn traditioneller Weise durch den Verkauf von Softwarelizenzen. Dieses Geschäftsmodell bedeutet für den Nutzer eine hohe Erstinvestition im Rahmen einer Anschaffungsgebühr und in den meisten Fällen eine erhebliche Einarbeitungszeit, da moderne Simulatoren viele Möglichkeiten, Libraries und Plugins anbieten (um damit ihren Preis zu rechtfertigen). Abbildung 1.1 zeigt eine schematische Darstellung des Kosten/Nutzen-Vergleichs für einen Benutzer von Simulationssoftware im klassischen Geschäftsmodell.

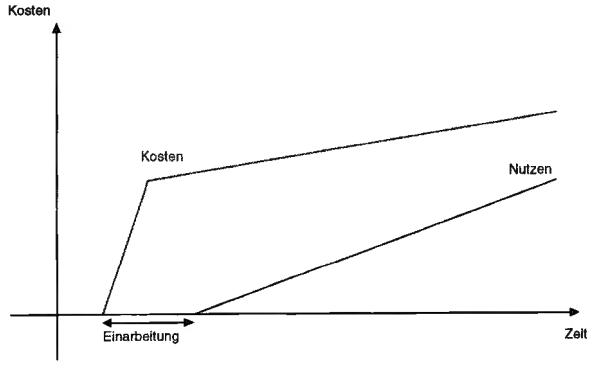


Abbildung 1.1: Das Kosten-Nutzenverhältnis eines konventionellen Geschäftsmodelles

Im Gegensatz dazu wird bei Application Service Providing die Dienstleistung eines (in diesem Fall Simulations-)Dienstleisters angeboten und nach Dauer oder Menge der erbrachten Rechenleistung und Betreuung verrechnet. Wenn man zusätzlich die Funktionalität auf die Bedürfnisse des Benutzers hin entwickelt, kann man damit die Wirtschaftlichkeit

von Simulation aus Benutzersicht steigern.
Der Benutzer kann seine Expertise im Problemfeld einbringen und muss sich nicht um Skalierung, Modellgröße oder aktuelle Softwareversionen kümmern. Der Service Provider hingegen kann sein Know How in den Simulations- und softwarespezifischen Fragen einbringen. Im günstigen Fall erreicht man damit nicht nur ein früheres Erreichen des Break Even Points für den Benutzer, der sich damit leichter für den Einsatz von Simulation entscheiden kann, sondern eine Nutzensteigerung für beide Seiten.
Abbildung 1.2 zeigt eine schematische Darstellung des Kosten/Nutzen-Vergleichs für einen Benutzer von Simulationssoftware im Geschäftsmodell des Application Service Providing.

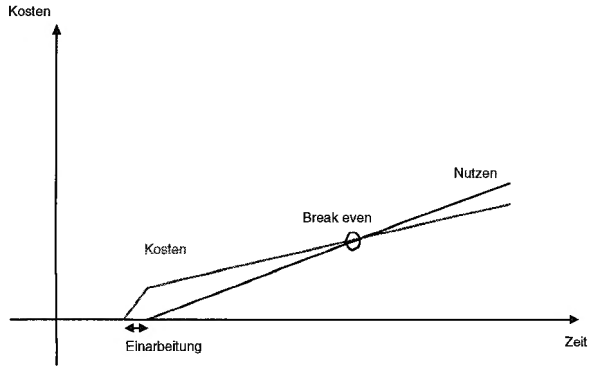


Abbildung 1.2: Das Kosten-Nutzenverhältnis eines ASP-Geschäftsmodelles

1.1.1 Die ASP-Architektur

Das Anbieten der Dienstleistung erfordert eine Rechnerarchitektur, die der Trennung der Aufgaben zwischen Service Provider und Servicenutzer entspricht und die Kommunikation zwischen beiden Servicepartnern ermöglicht.

Um die Logik eines Serviceprozesses klarer zu veranschaulichen wird der Prozess je nach Funktionalität in mehrere Ebenen (oder Schichten bzw. *Tiers*) zerlegt und indem nur die Kommunikation von einer Schicht zu ihrer Nachbarschicht zugelassen wird, gekapselt. Abbildung 1.3 zeigt eine sogenannte *4-Tier Architektur* [1].

Die Clientebene (Ebene 1) repräsentiert den Nutzer eines Services, der über einen webfähigen Computer an das Internet angeschlossen ist. Die Webfähigkeit kann auf verschieden Weise realisiert werden. Die Standardvariante ist ein Webbrowser, mittels dessen der Nutzer mit dem Serviceanbieter kommuniziert. Der Client kann so ausgelegt werden, dass er hat kaum Applikationslogik implementiert (ein sogenannter *dummer* Client). Für komplexere Anwendungen ist es allerdings sinnvoll, den Client mit Teilen der Gesamtfunktionalität auszustatten, um nicht für jede kleinste Teilaufgabe eine (manchmal zeitaufwändige) Kommunikation Client-Server-Client aufbauen zu müssen. Je nach Umfang der am Client verankerten Funktionalität spricht man von *Thin Client* bzw. *Thick Client*

Der Client schickt sogenannte *Requests* an einen Service Provider, der die ankommenden Requests auf der Ebene 2 bearbeitet und die benötigte Applikationslogik auf der Ebene 3 anstößt. Die nutzer- und applikationsspezifischen Daten werden in einer Datenbank (Ebene 4) gehalten. Nach Beendigung der Applikation (in unserem Fall des Simulationslau-

[1] Das (im Allgemeinen) n-Tier Modell ist die Grundlag der Kommunikation über das WWW und das Zusammenfassen mehrer Tiers zu einer Ebene ist (bei kleinen) Anwendungen häufige Praxis.

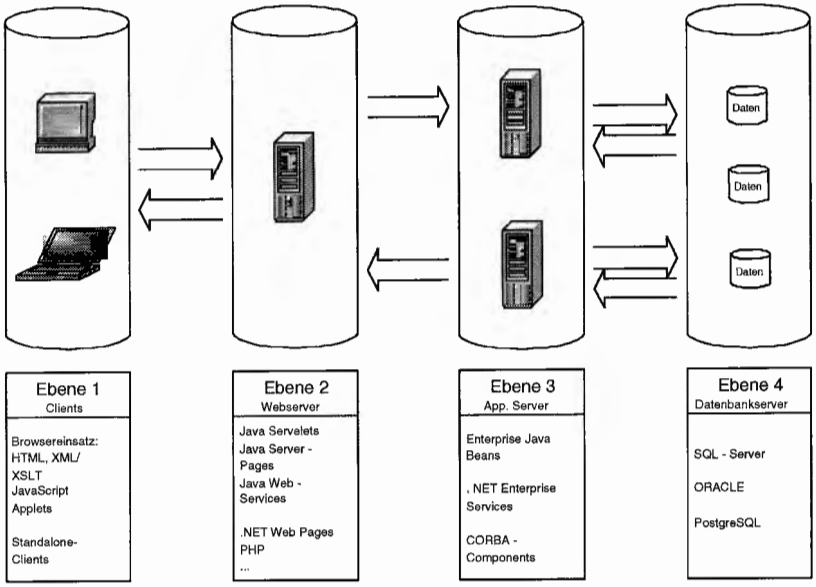


Abbildung 1.3: 4 Tier Architektur

fes) werden Ergebnisdaten über den Webserver via Internet der Clientebene übermittelt.

1.2 ASP mittels Web Services

Die Welt des klassischen Application Servcie Providing basiert noch immer auf der Interaktion zwischen Mensch und Maschine beziehungsweise der Interaktion zwischen Mensch und Applikation. Die Entwicklung von XML und damit verwandter Technologien hat die Möglichkeiten des Datenaustausches verändert und wird in Zukunft auch die Struktur des Internets nachhaltig beeinflussen. Durch die Definition von standardisierten Schnittstellen und Kommunikationsprotokollen wird die Kommunikation von Applikationen untereinander ermöglicht. Web Services, Grid Computing und Semantic Web sind die zentralen Schlagwörter dieser Internettechnologien, denen sich alle großen Softwarehersteller und Konsortien verpflichtet haben [13].
Der Begriff der Web Services entstand in dem Bestreben, die verschiedenen Informationstechnologielösungen in Großunternehmen zu integrieren. In den letzten Jahren wurden Unternehmensaufgaben zunehmend von Software unterstützt. Da es bis heute keine integrierte Gesamtlösung für alle Aufgaben gibt, wurden typischerweise Einzellösungen entworfen. Das hat dazu geführt, dass Unternehmen viele verschieden Softwaresysteme unterschiedlicher Anbieter oder auch Eigenentwicklungen im Einsatz haben.
Unter dem Schlagwort Web Services wird nun versucht, mittels Verwendung von XML-Technologie eine Infrastruktur zur Integration von Softwaresystemen zur Verfügung zu stellen um auch Softwaresystemen das Nutzen von anderen Applikationen und Services zu ermöglichen [57, 64]. Die Verwendung von Web Service Technologie bedeutet, dass der Austausch von Nachrichten zwischen einem Service Client und dem Service

Provider mittels SOAP abgewickelt wird. Die Verwendung von SOAP als Kommunikationsprotokoll, welches unabhängig von der Netzwerkschicht MIME Attachments, RPC (Remote Precedure Call) und Messaging unterstützt, ermöglicht die Kommunikation Client-Server auf Basis von HTTP und damit den einfachen Zugang zu Netzwerken, die heutzutage im Allgemeinen durch Firewalls geschützt sind.

1.3 Simulation als ASP

Um Simulation Services anzubieten, müssen in einem n-Tier Modell die folgende Aufgaben bewältigt werden:

- *Application Schicht*: Die Funktionalität von Simulationsstudien müssen so analysiert und aufbereitet werden, dass der Client von Remote darauf zugreifen und Simulaitonsläufe steuern kann.
- *Web Server Schicht*: Es muss auf der Serverseite sicher gestellt werden, dass Clientrequests bearbeitet werden können und in der Application Schicht erzeugte Ergebnisse dem Client übermittelt werden können.
- *Datenbank Schicht*: Simulatiosmodelle samt der zum Abarbeiten einer Simulationsstudie nötigen Information müssen in strukturierter Form in einer Datenbank gehalten werden.
- *Client Schicht*: Die konsistente Kommunikation zwischen Client und Server muss auch auf Clientseite implementiert werden. Außerdem müssen Werkzeuge zur Verfügung gestellt werden, die die Erzeugung von Simulationsmodellfiles und die Anzeige von Ergebnisfiles ermöglichen.

Abbildung 1.4 zeigt ein mögliches Szenario einer ASP-Architektur im Bereich Simulation.

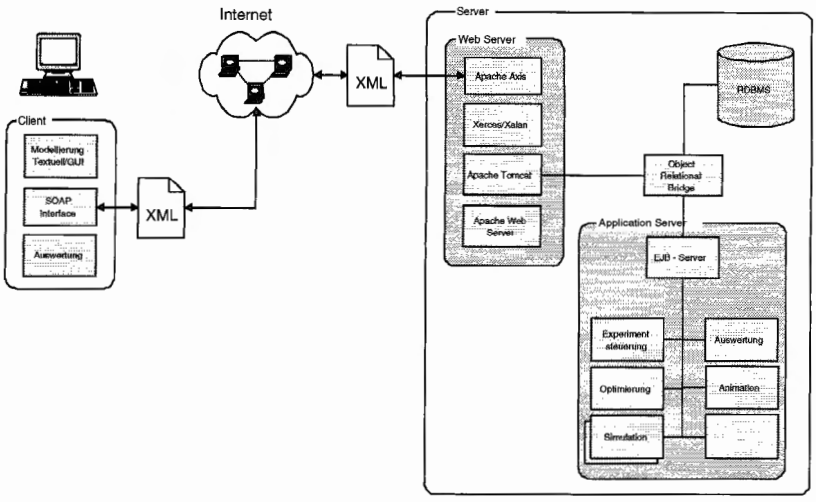


Abbildung 1.4: Szenario einer Simulation Service Providing Architektur

Der Client enthält eine Modellierungsschicht und eine Auswertungsschicht, die es ihm ermöglicht, Simulationsmodelle zu entwerfen und Ergebnisdaten anzuzeigen. Über ein SOAP-Interface kommuniziert er via Internet mit dem Webserver des Service Providers. Der Webserver besteht in diesem Beispiel aus einer Apache Produktsuite. Die Applikationslogik wird über einen EJB-Server abgewickelt und beinhaltet die Funktionen, die in einer Simulationsstudie angewendet werden. Die dafür benötigten Daten sind in einer relationalen Datenbank gehalten und der Datenaustausch zwischen Webserver, Applikationsserver und Datenbank wird über eine Object-Relational-Bridge bewerkstelligt.
Dieses Beispiel mit den spezifischen Produkten ist nur eine von vielen Möglichkeiten, eine mehrschichtige Architektur zu realisieren. Die im SimASP-Framework benötigte Technologie ist davon abweichend und wird im Weiteren näher erläutert.

Teil II

Grundlagen

Kapitel 2

Simulation

2.1 Was ist Simulation und wofür wird Simulation eingesetzt

Da mit Simulation sehr verschiedene Begriffe assoziiert werden, möchte ich eine Klärung des Begriffs Simulation an den Anfang dieser Arbeit stellen. Das Wort *Simulation*, abgeleitet vom lateinischen Wort *„simulare"*[1], hat im normalen Sprachgebrauch unterschiedliche, teilweise missverständliche Verwendungen. Für die Definition von Simulation im technischen Kontext eignet sich zum Beispiel die VDI-Richtlinie 3633 heranziehen:

> *"Simulation ist das Nachbilden eines Systems mit seinen dynamischen Prozessen in einem experimentierfähigen Modell, um zu Erkenntnissen zu gelangen, die auf die Wirklichkeit übertragbar sind. Im weiteren Sinne wird unter Simulation das Vorbereiten, Durchführen und Auswerten gezielter Expe-*

[1] nachbilden, nachahmen, jemandem etwas vortäuschen

rimente mit einem Simulationsmodell verstanden."[102]

Unter Simulation im technischen Sinne versteht man hier also das Verfahren, ein reales System nachzubilden (zu *modellieren*) und mit dieser Nachbildung (= *Simulationsmodell*) zu experimentieren, um dessen Verhalten zu untersuchen, ohne in das möglicherweise noch nicht existierende reale System eingreifen zu müssen.
Um ein reales System so behandeln zu können, muss man mittels einer entsprechende Analyse das System bzw. die Abläufe im System bedingt durch äußere Einflüsse verstehen. Folgende Fragen sind zu beantworten:

- Was ist das Ziel einer Simulation?
- Welche Rahmenbedingungen müssen eingehalten werden (z.B. Zeit, Kosten ...)?
- Welche Zustände kann das System annehmen (Modellzustände, die im realen Systemablauf nicht vorkommen können oder dürfen, müssen ausgeschlossen oder umgangen werden)?
- In welchem Ausmaß besteht die Möglichkeit, und welche Beschränkungen existieren, in das System eingreifen zu können?
- Wie reagiert das System auf äußere Einflüsse? Kann es zu Störungen im System kommen und wie müssen solche im Modell abgebildet werden.

Die Ergebnisse der Simulation werden dann mathematisch anlysiert, formalisiert und am Computer implementiert, also *modelliert.*

2.2 System

Ähnlich wie der Begriff Simulation sind auch für den Begriff System verschiedene Interpretationen gebräuchlich. Für diese Arbeit wird folgende Definition verwendet:

> *Ein System ist die Menge von Objekten einer gewissen Struktur, die durch regelmäßige Abhängigkeiten voneinander oder Wechselwirkung untereinander verbunden sind.*

Wesentlich ist die Beachtung der Systemgrenzen nach außen. Idealerweise wird die Interaktion des Systems mit seiner Umwelt durch standardisierte Schnittstellen (Eingänge, Ausgänge) beschrieben (siehe Abbildung (2.1)).

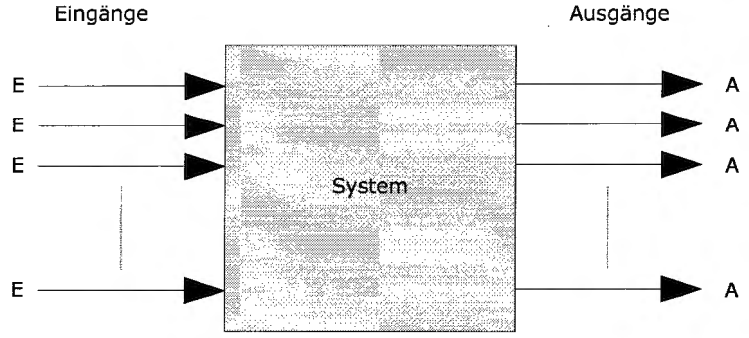


Abbildung 2.1: System

Weiterführende Beschreibungen siehe [71].

2.3 Einsatzgebiete der Simulation

Im Wesentlichen sind zwei Gründe für die Verwendung von Simulation ausschlaggebend:

1. Ein System bzw. die Systemeigenschaften eines realen Systems kann nicht mehr analytisch erfasst werden. Eine mögliche Ursache dafür ist zum Beispiel die zu hohe Komplexität des Systems.

2. Ein Experiment am realen System ist nicht möglich oder sinnvoll. Gründe dafür sind z.B.:

- Zu zeitaufwendige Experimente
- Zu hohe Kosten der Experimente
- Zu risikoreiche Experimente
- Zu schwierige bzw. nicht durchführbare Experimente
- Experimente, die das System irreversibel verändern würden

Simulation stellt oft einen einfacheren oder auch billigeren Weg dar, Information über ein System und dessen Verhalten zu gewinnen. Im Zusammenhang damit ergeben sich folgende Mglichkeiten:

- Prüfen von Hypothesen
- Vorhersagen von Systemverhalten
- Verbesserungen von Systemverhalten
- Entwürfe von Lösungsansätzen zu einem System

2.4 Grenzen der Simulation

2.5 Vom Modell zur Simulation

Unter einem *Modell* versteht man die idealisierte und vereinfachte Nachbildung eines realen Systems. Ein *mathematisches Modell* bezeichnet ein Modell. das mit mathematischen Hilfsmitteln (Gleichungen, Graphen, Formeln) beschrieben wird. Bei der Modellbildung werden - zwecks Vereinfachung - äußere Einflüsse mitunter vernachlässigt. Dies ist auch dann gerechtfertigt, wenn kein formaler Beweis die Vereinfachung stützt. Voraussetzung dafür ist, dass die relevanten Eigenschaften des Systems erhalten bleiben, da das Modell sonst keine Rückschlüsse auf das Verhalten des realen Systems zulassen würde. Der Übergang vom realen System besteht formal aus den Schritten (siehe Abbildung (2.2)):

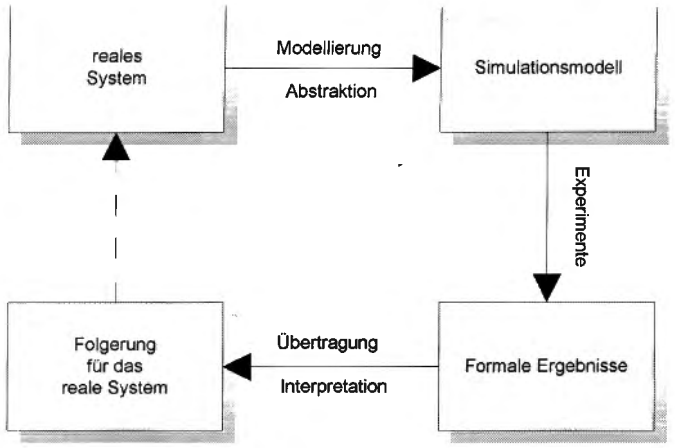


Abbildung 2.2: Simulationskreislauf

- Abstraktion des realen Systems und Erstellen des Simulationsmodells (= *Modellierung*)
- Validierung des Simulatinsmodells gegen das reale System
- Experimentieren mit dem Simulationsmodell um zu (formalen) Ergebnissen zu kommen.
- Interpretation der Simulationsergebnisse durch Übertragung auf das reale System
- Umsetzung der Ergebnisse

2.6 Aufbau einer Computersimulation

Im Folgenden wird der Projektablauf einer Simulation (hier Computersimulation) beschrieben:

2.6.1 Problemformulierung und Zielsetzung der Simulation

Als wichtige Voraussetzung für eine erfolgreiche Simulation muss eingangs das Problem, dessen Umfeld und das Ziel der Simulation präzise beschrieben werden. Dabei müssen nicht notwendigerweise quantitative Aussagen im Vordergrund stehen.

2.6.2 Modellierung

Nach der Analyse erfolgt die formale Beschreibung mit dem Ergebnis eines formalen Modells. Dieses Modell kann unter Umständen mit mathematischen Hilfsmitteln weiter analysiert und strukturiert werden. Hat das abstrakte Modell die benötigte Struktur, kann mit der softwarepaketbezogenen, computerunterstützten Implementierung begonnen werden. Zur Modellierung gehört außerdem die Auswahl geeigneter numerischer Verfahren für kontinuierliche Modelle und die Anwendung statistischer bzw. wahrscheinlichkeitstheoretischer Methoden im diskreten Bereich, um einen Kompromiss zwischen Genauigkeit und Aufwand zu schließen.

2.6.3 Identifikation

Werden für ein Modell bestimmte Parameter benötigt, die nicht auf einfache zeitliche und räumliche Weise bestimmt werden können (weil sie z.B. nicht gemessen werden können), so müssen deren Werte geschätzt werden.
Für diese Schätzung führt man eine hinreichend große Anzahl an abgeschlossenen Teilexperimenten durch und vergleicht die Systemantworten (Messdaten) mit denen des Modells. Dieser Schritt wird iterativ durchgeführt, sodass die Modellantworten eine bestmögliche Übereinstimmung mit den gemessenen Systemantworten liefern. Diesen Vorgang nennt man *(Parameter–)Identifikation.*

2.6.4 Validierung

Bei der Validierung des implementierten Modells, werden wiederum zu einander gehörende Ein– und Ausgangsgrößen mit den beobachteten Werten des Systems verglichen. Wesentlich ist, das reale System genau zu überprüfen, um eventuell benötigte Änderungen am Modell selbst durchführen zu können.

Bevor also mit den echten Simulationsläufen begonnen wird, werden bereits etliche Simulationsläufe durchgeführt, um eine der Aufgabe entsprechend bestmögliche Übereinstimmung von realem System und Modell zu erreichen.
Es besteht ein enger Zusammenhang zwischen Modellbildung und Simulation. Dieser ist in Abbildung (2.3) deutlich zu sehen. Einerseits wird zur Simulation ein Modell benötigt, die Simulation selbst dient aber auch als Unterstützung der Modellbildung.

2.6.5 Dokumentation der Ergebnisse

Zwecks Nachvollziehbarkeit von Simulationsläufen müssen außer der Interpretation der Modell–Ergebnisse und der Übertragung auf das reale System auch die Problemformulierung und alle weiteren Schritte der Modellbildung (Eingrenzung der Umgebung, Vereinfachungen, Genauigkeit der numerischen Verfahren usw. ...), sowie die Phase der Identifikation und Validierung dokumentiert werden.

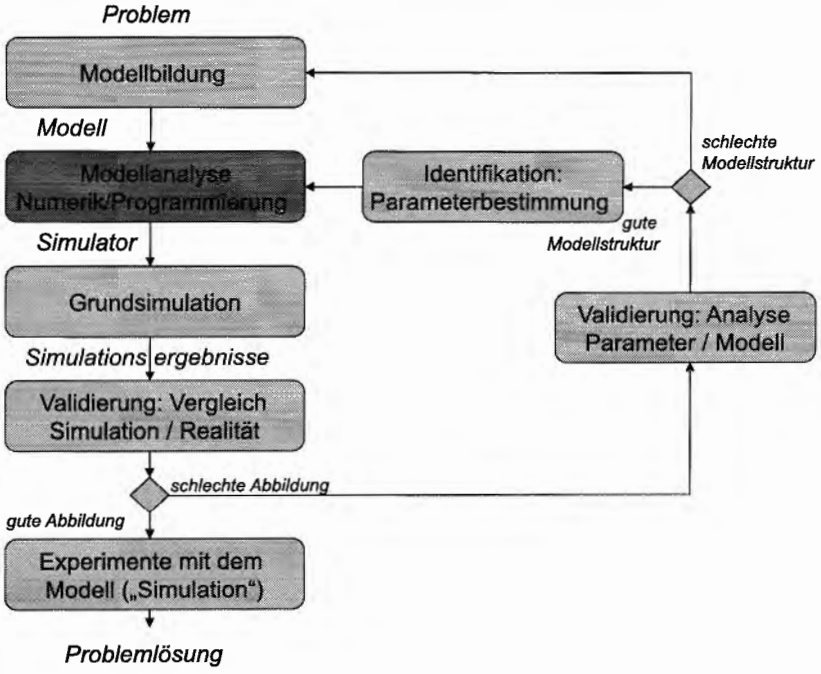


Abbildung 2.3: Iterative Identifikation und Validierung

2.7 Simulationsarten

2.7.1 Kontinuierliche Simulation

Charakteristisch für die kontinuierliche Simulation ist die permanente (im Abhängigkeit von der Zeit) Änderung der relevanten Kenngrößen. Solche Systeme werden typischerweise durch Differentialgleichungen beschrieben.
Klassische Bereiche, in denen kontinuierliche Simulation zum Einsatz kommen:

- Physik
- Elektrotechnik
- Mechanik
- Medizin
- Regelungstechnik
- Robotik
- Chemie
- Wettervorhersage
- Physiologie
- Umweltsysteme
- Wirtschaft
- Sozialwissenschaften

2.7.2 Diskrete Simulation

Die relevanten Größen der Simulation sind diskreter Art. D.h. der Systemzustand ändert sich nur bei endlich vielen zeitlichen Punktenzwischen zwei benachbarten Punkten ist der Systemzustand konstant. Ein spezieller Teilbereich der diskreten Simulation ist die Simulation stochastischer Modelle, bei denen die vorliegenden Daten einer gewissen Streuung unterliegen.
Diskrete Simulation findet sich z.B. in folgenden Bereichen:

- Fertigungssysteme
- Montage– und Materialflußsysteme
- Transportsysteme
- Verkehrssysteme
- Computernetzwerke
- Kommunikationssysteme
- betriebliche Abläufe
- Lagerstandshaltung
- Signalsteuerung
- Prozessdatenverarbeitung

2.7.3 Hybride Simulation

Hybride Systeme sind dadurch gekennzeichnet, dass in ihnen kontinuierliche Abläufe von diskret auftretenden Ereignissen geändert werden, d.h. dass ein Verbindung des kontinuierlichen und diskreten Simulationsansatzes auftritt. Durch hybride Simulation können Konzepte der klassischen diskreten bzw. kontinuierlichen Simulation zusammengeführt bzw.

erweitert werden.
Dementsprechend ähneln die Anwendungsgebiete der hybriden Simulation auch den klassischen Anwedungsfeldern der diskreten und kontinuierlichen Simulation:

- Verkehrssimulation
- Militär
- ökologische Systeme
- Wirtschaft
- Finanzwesen
- Sozialwissenschaften
- u.a.

Je nach Aufgabenstellung sind die Anforderung an die Güte der Modellösung verschieden. Zum Beispiel werden im gleichen Anwendungsfeld Simulationen mit völlig unterschiedlicher Genauigkeit gerechnet. Die Konzeption der Simulation muss entsprechend angepasst werden.

2.8 Geschichte der Simulation

Die ersten Simulationen im zivilen Bereich entstanden in den späten 50er Jahren, als in vereinzelten Großindustrien (Stahl– und Luftfahrtindustrie) Simulationsanwendungen in Fortran implementiert wurden. Diese Simulationsanwendungen waren aufwendige Entwicklungen für einzelne Probleme und waren dementsprechend nicht allgemein einsetzbar und auch sehr teuer.
Anfang der 60er Jahre wurden die ersten Simulationssysteme für diskrete Simulation entwickelt, z.B. GSOP, OPS-1, SIMULA, GPSS,

SIMAN, ...
Etwas später wurden die ersten Simulationssysteme für kontinuierliche Simulation eingeführt, z.B. DYNAMO und es wurden bereits erste Schritte in Richtung graphische Benutzerschnittstellen unternommen.
Mit der Entwicklung von Werkzeugen für breitere Einsatzgebiete, war diese Zeit durch ein Ausweitung von Simulation in andere Industriezweige geprägt. Da zu dieser Zeit für Computersimulation noch Großrechner notwendig waren, blieb der Einsatz der Simulation trotzdem relativ großen Firmen und Forschungseinrichtungen vorbehalten.
Die nächste Revolution wurde durch die Einführung des PCs in den 80ern und die Verbesserung der numerischen Methoden eingeleitet. Plötzlich wurde Simulation auch für kleinere Firmen erschwinglich. Mit der rasanten Entwicklung von PC und Netzwerken begann der Einsatz von Simulation auf breiter Front und Simulation ist bis heute ein Standardwerkzeug, das mit erweiterten Methoden (parallele Simulation, verteilte Simultion, ...) in verschiedenen Anwedungsfeldern verwendet wird.

2.9 Webbasierte Simulation

Die Webbasierte Simulation ist ein weitläufiges Feld, da praktisch alle Simulationsvarianten, die das WWW in irgendeiner Weise nutzen, in dem Kontext genannt werden. Darunter fallen unter anderem

- Anwendungen, die auf über das Web zugängliche Datenbanken zugreifen.
- Webgestützte Modellierung.
- Der Austausch von Szenarien- und Outputdaten per Email.

- Für das Projektmanagement von Simulationsstudien errichtete Internet-Marktplätze.
- Bereitstellung von unterstützender Literatur zur Modellbildung und Simulation über das Internet.
- u.a.

Von besonderem Interesse für diese Arbeit sind die Entwicklungen der Webbasierten Simulation, die auf einem Client-Server Ansatz beruhen [21, 29, 30, 70, 80, 91, 95, 108, 110].

2.9.1 Vergleich Simulations- und Webtechnologie

In einem Vergleich der Simulations- und Webtechnologien erkennt man sofort signifikante Unterschiede, die Ursachen für den Erfolg des Webs im Gegensatz zur Simulation sind [107, 110] (siehe Tabelle 2.1).
Aus dieser Gegenüberstellung ergeben sich folgende Konsequenzen um den Erfolg der (Webbasierten) Simulation zu erhöhen [107, 110]:

- Modularisierung: Seit den 90er Jahren gibt es die Entwicklungstendenz, dass Simulationssysteme versuchen, möglichst alle Funktionen einer Simulationsstudie von der Modellierung über die Simulation bis zur Optimierung zu vereinen. Diese Softwaresysteme sind daher teuer, meist propietär und damit schlecht adaptierbar.
- Einführung von (Quasi-) Standards im Simulationsbereich
- Die Kombinaton von Simulaton und Optimierung: Die Optimierung ist zumeist der zentrale Beweggrund von Firmen, Simulation anzuwenden.

	Webtechnologien	Simulations-technologien
Gebräuchliche Standards	Ja: HTML, XML, TCP/IP, SOAP	Kein Standard für Modellierung und Resultatdarstelung
Datahandling	Per eindeutigem URL	Proprietär
Informations-strukturen	Baumstrukturen, Listen	komplex
Expertenwissen notwendig	Nein: klicken und lesen	Ja: aus verschiedenen Gebieten
Navigation	Einfach	Sehr kompliziert
Bedienungskomfort	Gut	Gering

Tabelle 2.1: Vergleich von Simulations- und Webtechnologie

- Bereitstellung von Simulationsdienstleistung: Damit wird für den Simulationsnutzer der Zeitpunkt des Return of Investment schneller erreicht und auch besser einschäzbar (siehe Kapitel 1.1)

Das SimASP Framework unter Verwendung von XML und Web Services versucht diese Forderungen zu erfüllen. Eine Einschätzung der zukünftigen Entwicklung findet sich im Kapitel 11.2.

Kapitel 3

Internettechnologien

Dieses Kapitel soll eine Einführung in die grundlegenden Technologien und Methoden des Internets geben. Es wird allerdings nur auf aktuelle Techniken der webbasierten Anwendungsentwicklung näher eingegangen. Obwohl das Konzept des ASP bereits vor XML entwickelt und auch angewandt wurde, hat doch die Entwicklung von XML das WWW und die Zusammenarbeit der verschiedenen Internetkomponenten auf technischer Ebene grundlegend verändert. Daher steht am Beginn ein Abschnitt über *XML*.

Danach werden die Technologien präsentiert, die in dieser Arbeit zur Anwendung gekommen sind. Als erste Basistechnologie für die XML-Darstellung verteilter Systeme werden *Web Services* beschrieben, die genau genommen ein Bündel von Technologien zur Beschreibung von Schnittstellen sind.

Im folgenden Abschnitt wird der bereits aus den 90er-Jahren stammende Begriff des *Grid Computing* erläuteret. Grid Computing bezeichnete damals eine Architektur für verteilte Syteme, die auf dem WWW aufbauten. Mittlerweile wurde dieser Ansatz in Verbindung mit Web Services

zu sogenannten *Grid Services* weiterentwickelt.
Da die Internetentwicklung immer stärker in Richtung Kooperationsarbeit geht und *Open Source* Software beziehungsweise *Open Protocol* Spezifikationen einen wesentlichen Aspekt der modernen kollaborativen Internetentwicklung darstellen, möchte ich zum Abschluss dieses Kapitels diese Begriffe zumindest klar definieren und kurz deren Vor- und Nachteile im Gegensatz zu proprietären Technologien anreißen.

3.1 XML

Die *eXtensible Markup Language*, abgekürzt XML, ist ein Standard zur Erstellung maschinen- und menschenlesbarer Dokumente und definiert dabei die Regeln für den Aufbau solcher Dokumente. Sie wurde auf der Basis der *Standard Generalized Markup Language* (SGML) entwickelt, die Regeln für die Definition von plattformunabhängigen Auszeichnungssprachen vorgibt. Ein bekanntes Beispiel einer SGML Auszeichnugnssprache ist die HyperText Markup Language (HTML), die seit Anfang der 90er Jahre gemeinsam mit dem HyperText Transfer Protocol (HTTP)und den Unified Ressource Locators (URL), die Basis des WWW bildet [25, 79].
Der große Sprachumfang und die große Flexibilität von SGML bedingen deren hohe Flexibilität und sind die Ursache dafür, dass SGML-Werkzeuge teuer sind und in der Regel nur von Großunternehmen oder -organisationen eingesetzt werden [79].
Diese Umstände führten zu der Entwicklung von XML als einer Teilmenge der SGML, die seit 1998 als W3C Recommendation (siehe 3.1.1) vorliegt. XML ist eine Sprache zur Definition von Auszeichnungssprachen beziehungsweise Datenformaten, die gleichzeitig deren Syntax vorgibt [25]. XML-Dokumente enthalten Texte oder Dateien, denen logische Struktur mitgegeben wird aber sie enthalten keine

Layoutinformation. Diese wird in einem eigenen Stylesheetdokument gespeichert (siehe Abbildung 3.1).

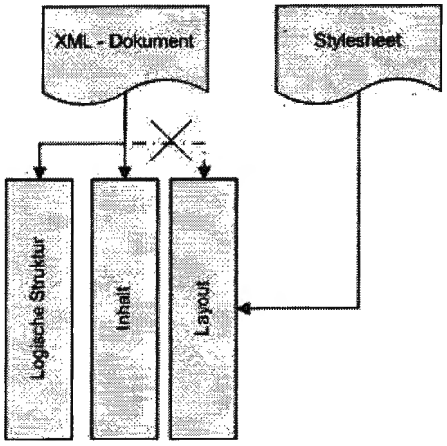


Abbildung 3.1: Information in einem XML-Dokument

Damit werden die Daten vollständig von ihrer Reppräsentation getrennt. Mit Hilfe der eXtensible Stylesheet Language (XSL) können XML-Dokumente in verschiedene Präsentationsformate (wie HTML, PostScript oder LaTex) umgewandelt werden-oder die gleichen XML-Daten können als Tabelle oder Grafik ausgegeben werden.

Im Folgenden werden einige grundlegende Begriffe der XML samt einiger ergänzender Technologien erläutert.

3.1.1 World Wide Web Consortium - W3C

Das W3C wurde 1994 am Massachusetts Institute of Technology (MIT) und im europäischen Kernforschungszentrum (CERN) gegründet. Es ist ein inoffizielles Gremium und kann im Gegensatz z.B. zu ISO keine

Standards herausgeben und verabschiedet statt dessen so genannte Recommendations, die in der Softwareentwickler Community aber de facto Standards sind [25]. Der Standardisierungsprozess enthält folgende Schritte:

1. *Working Draft*: Der Arbeitsentwurf einer Arbeitsgruppe zu einem Thema, in dem sich das W3C um einenStandardisierung bemüht.

2. *Last Call Working Draft*: Ein technischer Bericht, der vom W3C und der Öffentlichkeit begutachtet werden soll.

3. *Candidate Recommendation*: Ein Empfehlungskandidat, mit dem Erfahrungen bei der Implementierung gemacht werden sollen.

4. *Proposed Recommendation*: Ein Entwurf zur Recommendation, in dem die Implemetierungserfahrungen bereits eingearbeitet wurden und der dem Beratungskomitee zur Begutachtung vorgelegt wird.

5. *Recommendation*: Eine offizielle Empfehlung des W3C. Z.B.: HTML, XML, CSS, SOAP, WSDL, UDDI, ...

3.1.2 Aufbau einer XML-Datei

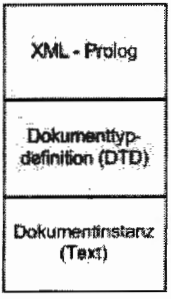


Abbildung 3.2: Aufbau eines XML-Dokuments

Abbildung 3.2 zeigt die Teile eines XML-Dokuments. Zuerst kommt der optionale (und meistens vorhande) XML-Prolog. Er enthält einige grundlegende Eigenschaften des XML Dokuments, wie zum Beispiel die XML-Versionsnummer oder den verwendeten Zeichensatz fest.
Die (ebenfalls optionale) Dokumenttypdefinition (DTD) legt die Struktur der Dokumentinstanz fest.
Anschließend folgt die Dokumentinstanz selbst, die den strukturierten Text enthält.

Der logische Aufbau eines XML Dokumentes ist ein hierarchisch strukturierter Baum, der genau ein Element (siehe unten) in der obersten Ebene enthalten muss. Als Baumknoten gibt es:

- *Elemente*: Sie bestehen aus je einem Start-Tag (<Tag-Name>) und einem End-Tag (</Tag-Name>) oder einem Empty-Element-Tag (<Tag-Name />).
- *Attribute*: Sie sind in einem Start-Tag oder Empty-Element-Tag geschriebene Schlüsselwort-Werte-Paare (Attribut-Name = "Attribut-Wert")
- *Processing Instruction*: Sie werden in der Praxis meist eingesetzt, um in XML-Dokumenten Verarbeitungsanweisungen in anderen Sprachen einzubauen und haben die Form <?Ziel-Name Parameter ?>
- *Text*: Er kann als normaler Text oder in Form eines CDATA-Abschnittes (<![CDATA[beliebiger Text]]>) auftreten.
- *Kommentare*: Sie sind durch folgende Syntax gekennzeichnet: <!– Kommentar-Text –>

XML Dokumente, die diesen Regeln entsprechen heißen *wohlgeformt*.

3.1.3 Kerntechnologien

Die Kerntechnologien können in drei große Gruppen aufgeteilt werden. Zum Einen APIs zur Verarbeitung von XML, zweitens Sprachen zur Beschreibung von XML-Dateien und drittens auf der Basis von XML definierte Sprachen, die die Grundfunktionalität zur Arbeit mit XML-Dokumenten anbieten.

3.1.3.1 APIs zur Verarbeitung von XML

Es gibt zwei wesentliche Programmierschnittstellen (APIs) für den Zugriff auf XML-Dokumente. Die ereignisorientierte *Simple API for XML* (SAX) und das baumorientierte *Document Object Model* (DOM). Beide Modelle haben aufgrund ihrer Vor- und Nachteile unterschiedliche Anwendungsgebiete.

3.1.3.1.1 SAX: SAX ist eine standardisierte ereignisorientierte Möglichkeit, eine XML Datei durch einen Parser zu bearbeiten. Die Eingabedatei wird als Dateistrom rein sequentiell verarbeitet und in einen Strom von Ereignissen umgewandelt. Applikationen können sich registrieren und werden beim Auftreten betimmter Ereignisse benachrichtigt. Der Vorteil von SAX liegt darin, dass durch die sequentielle Bearbeitung eines XML Dokuments nicht die ganze Datei im Speicher gehalten werden muss. Allerdings ist es nötig, beim Zurückspringen auf einen weiter vorne liegenden Knoten, die ganze Datei noch einmal von Anfang an zu bearbeiten.

3.1.3.1.2 DOM: DOM verwendet dagegen eine baumorientierte Verarbeitung, die aus einer XML-Datei einen Objektbaum erzeugt, in der die Elemente in einer hierarchischen Struktur vorliegen. Dadurch kann auf alle Elemente zugegriffen werden. Der Nachteil von DOM ist

der hohe Speicherbedarf, der sich proportional zur Gröe der Eingabedatei verhält.

3.1.3.2 Sprachen zur Beschreibung von XML Dateien

Die Struktur von XML Dokumenten wird durch Metasprachen in Form von Grammatiken definiert. Ein XML Dokument, dass den Vorgaben einer Grammatik entspricht und wohlgeformt ist, heißt *gültig.* Die gebräuchlichsten Metasprachen sind *Dokumenttypdefinition* (DTD) und *XML Schema.*

3.1.3.2.1 DTD: In einer DTD können Vorgaben an XML Dokumente in Form von Grammatiken ausgedrückt werden. Der Sprachumfang von DTDs wurde gemeinsam mit XML standardisiert, ist allerdings relativ gering und bietet nur geringe Möglichkeiten, den Inhalt von Elementen und Attributen zu beschränken. Weitere Nachteile sind die Notwendigkeit einer eigenen Auszeichnungssprache und die geringe Anzahl von Datentypen [22, 48].

3.1.3.2.2 XML Schema: Im Unterschied dazu ist ein XML Schema selbst ein XML Dokument, welches erlaubt, komplexere Zusammenhänge als mit einer DTD zu beschreiben. Zum Beispiel bietet XML-Schema die Möglichkeit, den Inhalt von Elementen und Attributen mittels regulärer Ausdrücke zu beschränken [122].

3.1.3.3 XML Sprachen

Es folgt ein Auszug der wichtigsten XML Sprachen, die zum Teil im Zusammenhang mit dem SimASP Framework verwendet wurden.

3.1.3.3.1 XSLT: Die *Extended Stylesheet Language Transormation* ist eine XML Anwendung, die es ermöglicht, ein XML Dokument in ein

anderes XML Dokument umzuformen (siehe [128]).

3.1.3.3.2 XPath: Die *XML Path Language* ist eine Sprache zur Adressierung bestimmter Stellen in einem XML Dokument bzw. zur Navigation in diesem (siehe [126]).

3.1.3.3.3 XQuery: Die *XML Query Language* wird zur Extraktion von Daten aus einem XML Dokument verwendet (siehe [125]).

3.1.3.3.4 XML Digital Signature: XML Dsig ermöglicht es, beliebige XML Dokumente digital zu unterschreiben (siehe [123]).

3.1.3.3.5 XML Encryption: XML Enc dient dazu, XML Daten zu verschlüsseln und verschlüsselte Daten als XML auszudrücken (siehe [121]).

Die im Rahmen von Web Services und Grid Services verwendeten XML Sprachen, werden in den zugehörign Kapiteln beschrieben.

3.1.4 XML Schema

Da im Framework des SimASP die Definition eines generischen Simulationsmodells mittels XML Schema erfolgt, wird im folgenden Abschnitt auf einige grundsätzliche Eigenschaften von XML Schema eingegangen.

3.1.4.1 Datentypen

Im Unterschied zur DTD stellt XML Schema einige grundlegende atomare Datentypen bereit. Diese atomaren Datentypen enthalten die "klassischen" Typen, wie sie zum Teil aus höheren Programmiersprachen bekannt sind:

- xsd:string
- xsd:decimal
- xsd:integer
- xsd:boolean
- xsd:date
- xsd:time

Dazu kommen weitere atomare Typen wie zum Beispiel:

- QName: Qualified Name, Bezeichnung innerhalb eines Namensraumes
- anyURI: Universal Resource Identifier (URI)
- language: Sprachbezeichnung, z.B. de-DE, en-US, fr
- ID: Identifikationsattribute innerhalb von XML Elementen
- IDREF: Referenze auf einen ID-Wert

Zusätzlich können in XML Schema auch einfache und komplexe benutzerdefinierte Datentypen erstellt werden. Datentypen lassen sich zum einen durch die Definition eines neuen Typs, zum anderen durch die Ableitung eines Typs aus bereits bestehenden Typen erstellen. Die Ableitung eines neuen Typs kann durch Erweiterung oder Einschränkung erfolgen [48].

3.1.4.2 Namensräume

Zur Vermeidung von Namenskonflikten werden sogenannte Namespaces definiert. Solche Namensräume sind durch URIs eindeutig identifiziert

und ihnen sind Sammlungen von Element- und Attributnamen zugeordnet. Dadurch werden universell eindeutige Namen als Kombination aus lokalen Namen und URIs definiert. Sie sind als Attribut eines Elements definiert und gelten für alle Nachkommen des Elements wie für das Element selber.

3.1.4.3 Schlüssel- und Fremdschlüsselkonzepte

Vergleichbar den Primärschlüsseln in relationalen Datenbanken lassen sich mittels XML Schema eindeutige Schlüssel definieren. Elemente und Attribute können dabei entweder (in einem bestimmten Bereich) als eindeutig oder auch als Schlüssel definiert werden.
Mittels Fremdschlüsseln werden Verweise auf Schlüssel oder eindeutige Elemente bzw. Attribute definiert. Damit ist sicher gestellt, dass es zu bestimmten Werten woanders eindeutig korrespondierende Werte gibt [25].

3.1.4.4 Import und Include

xml Schema Definitionen lassen sich auf mehrere Dokumente verteilen. Sie definieren einen gemeinsamen Namensraum und werden mittels *include* - Anweisungen zusammengesetzt. Wenn verschieden Zielnamensräume verwendet werden sollen, können andere XML Schemata mittels *include* importiert werden.

3.2 Web Services

Der Begriff der Web Services entstand in dem Bestreben, die verschiedenen Informationstechnologielösungen in Großunternehmen zu integrieren. In den letzten Jahren wurden Unternehmensaufgaben zunehmend von Software unterstützt. Da es bis heute keine integrierte

Gesamtlösung für alle Aufgaben gibt, wurden typischerweise Einzellösungen entworfen was dazu geführt hat, dass Unternehmen viele verschiedene Softwaresysteme unterschiedlicher Anbieter und/oder auch Eigenentwicklungen im Einsatz haben.
Um eine weitere Automatisierung und auch Optimierung der Geschäftsprozesse zu erreichen, ist die Integration der einzelnen Systeme sowohl innerhalb des Unternehmens als auch unternehmensübergreifend notwendig (z.B. zwischen einem CRM-System eines Liefranten und einem e-Procurementsystem eines Kunden) [57, 62].

Man unterscheidet genau genommen zwischen der Integration der Applikationen, der sogenannten *Enterprise Application Integration* (EAI) und der Integration verschiedener Datenbestände eines Unternehmens (oder auch unternehmensübergreifend) - z.B. können verschiedene Aspekte eines Kunden in verschiedenen Datenbanken abgelegt sein und es kann für CRM-Tätigkeiten notwendig sein, alle Information dieser Datenbestände zur Verfügung zu haben. Dann spricht man von der *Enterprise Information Integration* (EII) [57, 63].

Da die meisten Infrastrukturen historisch gewachsen sind und nicht für die Interaktion mit anderen Systemen konzipiert wurden, hat man es sowohl bei der EAI als auch bei der EII mit den folgenden Problemen zu tun [57]:

- *Plattformunabhängigkeit*: Verschiedene Systeme laufen häufig auf verschiedener Hardware unter verschiedenen Betriebssystemen und sind nicht dazu entworfen, mit anderen Systemen zu interagieren.

- *Prozessabhängigkeit*: Bestehende Interaktionen zwischen Systemen sind nicht so entworfen, dass sie problemlos geändert werden können. Die globalen Auswirkungen der lokalen Änderungen an

einem System sind nicht abschätzbar.

- *Datenformatabhängigkeit*: Daten werden in verschiedenen Formaten gespeichert. Z.B. kann ein Teil der Daten mittels einer relationalen Datenbank gespeichert werden, während ein anderes Teilsystem ein proprietäres Datenformat verwendet und keinen Exportfilter anbietet.

- *Optimierung der integrierten Systeme*: Um verteilte Systeme effizient zu betreiben, müssten Techniken wie Caching oder Lastbalancing angewendet werden. Die dazu erforderlichen transparente Softwarearchitektur ist nach einer Systemintegration oft nicht mehr gegeben.

- *Management der integrierten Systeme*: Durch die Änderungen am Gesamtsystem während der Integration ist es oft nicht mehr möglich, Aussagen über Zuverlässigkeit, Sicherheit oder Performance zu machen.

Unter dem Schlagwort Web Services wurde versucht, eine Infrastruktur zur Integration von Softwaresystemen zur Verfügung zu stellen, die sich an der *Service Oriented Architecture* (SOA) orientiert und nach den folgenden Prinzipien aufgebaut ist [57, 64]:

- *Lose Kopplung*: Die einzelnen Softwaresysteme bleiben unabhängig voneinander. Die Kommunikation erfolgt über Nachrichten.

- *Virtualisierung*: Um Services unabhängig von einzelnen Komponenten zu gewährleisten, werden Ressourcen und Softwaresysteme virtualisiert. Dadurch ist es möglich Kommunikationspartner dynamisch erst zur Laufzeit und nach Verfügbarkeit zu bestimmen

- *Einheitliche Konventionen*: Durch WS-Technologien werden Konventionen die die Datenformate, Protokolle und Qualitätseigenschften von WS regeln, festgelegt.

- *Standards*: Alle namhaften Software- und Plattformanbieter (z.B. BEA, IBM, Microsoft, SAP) halten sich an gemeinsame Standards. Das ermöglicht erst den Erfolg von WS.

3.2.1 Web Service Technologien

Es gibt keine einheitliche Definition von Web Services. Aber in allen Definitionsansätzen sind folgende Schlüsseleigenschaften von WS enthalten:

- Identifizierbarkeit durch einen *Uniform Resource Identifier* (URI). Damit können Objekte im Internet eindeutig identifiziert werden (Ein URL des WWW ist ein spezieller URI).

- Die Schnittstelle eines WS ist maschinenlesbar und wird mittels *Web Service Description Language* (WSDL) beschrieben.

- WS kommunizieren untereinander mittels SOAP[1]- Nachrichten.

- WS sind autonom. Das heißt, wie eine Nachricht von einem adressierten WS verarbeitet wird, kann nicht beeinflusst werden. Über *Quality of Service* (QoS) müssen zusätzliche Vereinbarungen getroffen werden

Zusätzlich wird je nach Einsatzgebiet zusätzlich die fogende Eigenschaft gefordert:

[1]SOAP wurde von *Simple Object Access Protocol* abgeleitet. Dieses Akronym wird mittlerweile als eigenständiger Terminus technicus verwendet

- WS werden mittels *Universal Description Discovery and Integration* (UDDI) katalogisiert und so Klienten bzw. andern WS angeboten.

3.2.1.1 SOAP

SOAP ist ein Framework zum Austausch von XML Nachrichten und ist selbst eine Anwendung der XML Spezifikation, die sich auf XML Standards wie XML Schema und XML Namespaces stützt. SOAP hat den Status eines W3C Standards und ist plattformunabhängig.

Eine SOAP Nachricht hat folgende Struktur (vergleiche [92])(siehe Abb. 3.3):

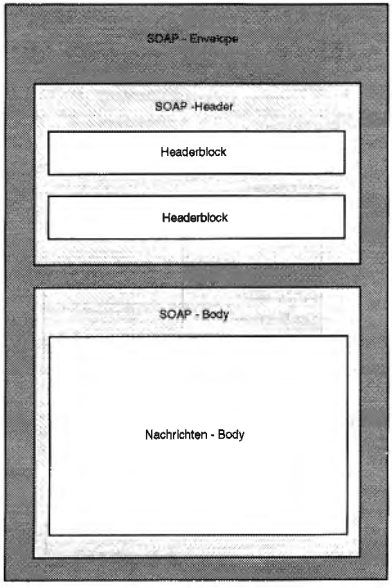


Abbildung 3.3: Struktur einer SOAP-Nachricht

Sie hat ein Envelope-Wurzelelement und optional ein (und dann nur genau ein) Headerelement. Das Headerelement muss als erstes Child-Element vor dem Body-Element auftreten.
Im Header können ein oder mehrere Headerblöcke auftreten. In ihnen werden für den Empfänger Kontroll- und Kontextinformation mitgegeben, die zur Verarbeitung der im Body auftretenden Nutzinformation nötig sind.
Der Body enthält die "Information" der Nachricht. Diese kann wiederum jeden gültigen, wohlgeformeten XML-Code enthalten und aus beliebig vielen Child-Elementen bestehen.

Diese Struktur erlaubt es, SOAP sehr flexibel einzusetzen. Es können sowohl XML- als auch Nicht-XML-Daten gesendet werden; zusätzlich werden *Remote Procedure Calls* (RPC) unterstützt.

3.2.1.2 WSDL

WSDL dient der Beschreibung der Schnittstellen eines Web Service und ist (wie alle Web Service Standards) auf XML aufgebaut.
In einer abstrakten Schnittstelle wird die Funktionalität (Porttypen, Operationen) und das Datenschema (Nachrichtentypen, Datentypen) eines Servicetyps aus Sicht des Anwenders beschrieben. Die Schnittstelle definiert, mit welchem Protokoll und welchem Nachrichtenformat mit einem Service eines bestimmten Typs kommuniziert werden kann. Außerdem kann genau ein Service durch Servicetyp, Protokollbindung und Adresse (entsprechend dem verwendeten Protokoll) identifiziert werden.

3.2.1.3 UDDI

UDDI dient zur Katalogisierung von Web Services und stellt eine Registratur aller Metadaten eines Web Service samt der Funktionalität

zum Verändern und Durchsuchen der Registratur dar.
UDDI ist als logisch zentralisierte aber physisch dezentralisierte Registrierungsdatenbank für Firmen und deren Web Services implementiert und wird selbst als Web Service angeboten. Unter anderem wird WSDL als WS-Beschreibungssprache unterstützt.
Die Informationen über die Web Services und deren Anbieter wird in *White, Yellow* oder *Green Pages* gehalten. Die Green Pages enthalten im Wesentlichen die WSDL Schnittstellen der angebotenen Web Services, in den White und Yellow Pages finden sich Informationen zu den Providern [57].

3.2.2 Funktionsweise von Web Services

Mit den Technologien SOAP, WSDL und UDDI kann man bereits das Schema eines funktionierenden Web Services entwerfen (siehe Abb. 3.4)

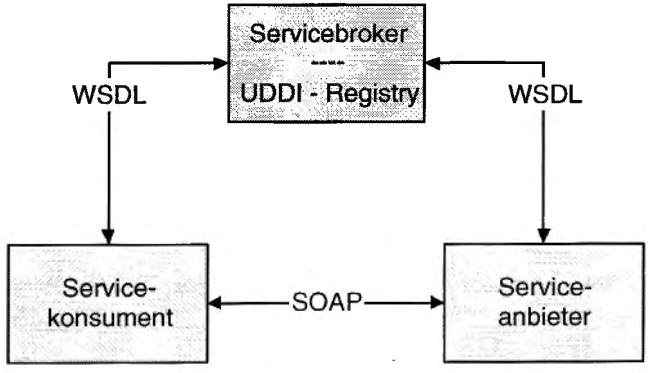


Abbildung 3.4: Funktionsweise von Web Services

Die Abbildung 3.5 gibt einen Überblick über alle nötigen Aktionen, um

einen Web Service zu Verfügung zu stellen und zu nutzen([49]).

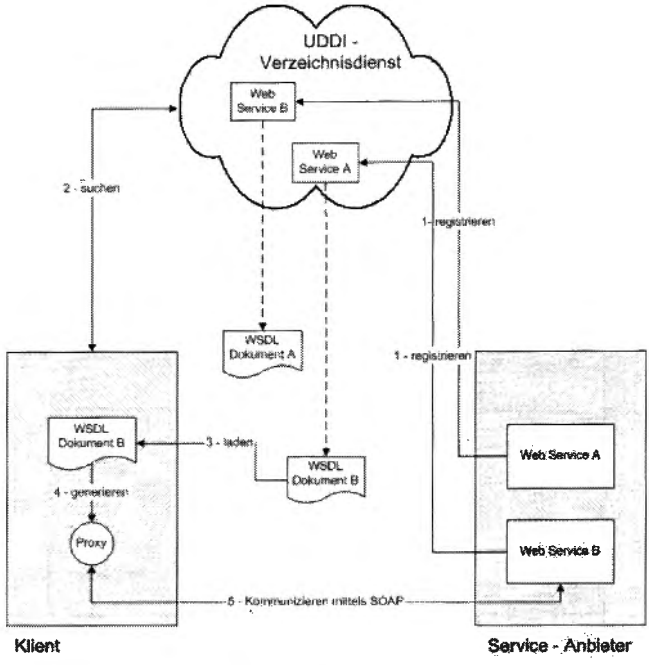


Abbildung 3.5: Aktionen bei dem Einsatz von Web Services

Ein Dienstanbieter registriert seine Web Service Dienste bei einem UDDI-Verzeichnisdienst, um die Information verfügbar zu machen, welche Dienste er anbieten will und wie diese angesprochen werden können. Wie die Kommunikation mit den Diensten aussehen muss, ist dabei in WSDL-Dokumenten beschrieben, die im Internet veröffentlicht sind. Will nun ein Client einen Dienst nutzen, sucht er sich einen für seine Zwecke geeigneten Dienst aus dem UDDI-Verzeichnis aus. Nach der Auswahl eines Dienstes (in der Abbildung Web Service B), wird das

zugehörige WSDL-Dokument geladen und dazu verwendet, einen Proxy für den Web Service zu generieren. Der generierte Proxy wird verwendet, um mit dem tatsächlichen Web Service via SOAP-Nachrichten zu kommunizieren ([49]).

Die Aktionen eines Dienstanbieters werden in Abbildung 3.6 näher dargestellt.

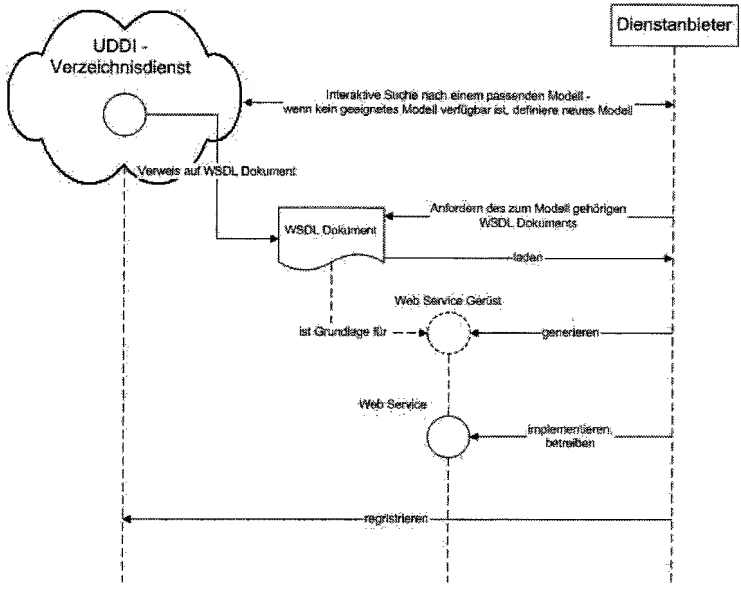


Abbildung 3.6: Sequenzdiagramm der Aktionen eines Web Service Providers

Zuerst verwendet der Dienstanbieter einen UDDI-Verzeichnisdienst, um zu prüfen, ob es ein Modell (ein sogenanntes *tModel*) gibt, das die Art von Dienst beschreibt, den er anbieten will. Gibt es bereits ein tModel,

verwendet der Dienstprovider das zugehörige WSDL-Dokument als Grundlage für seinen Dienst. Existiert kein geeignetes tModel, muss der Anbieter ein neues tModel samt zugehöriger WSDL Beschreibung erzeugen und bei dem UDDI-Verzeichnisdienst registrieren.
Auf dem WSDL-Dokument basierend, wird ein Gerüst für den Web Service generiert, das der Anbieter dann mit der entsprechenden Funktionalität ergänzt. Danach kann der Provider den Web Service betreiben und registrieren. Seine Schnittstelle entspricht nun genau der im WSDL-Dokument geforderten Spezifikation.
Möchte man eine bereits existierende Anwendung als Web Service anbieten, wird zu der Applikation ein neues WSDL-Dokument erzeugt und der Dienst bei einem UDDI-Verzeichnisdienst registriert.

Ein Client sucht zu Beginn in einem UDDI Verzeichnisdienst nach einem spezifischen Web Service. Wenn so ein Dienst existiert, enthält die UDDI Registry sowohl einen Verweis auf den Aufenthaltsort des zugehörigen WSDL-Dokuments als auch der URI unter der der Service im Internet anspechbar ist. Das WSDL-Dokument spezifiziert, wie Nachrichten an den ausgewählten Dienst aussehen müssen. Unter Verwendung eines geeigneten Werkzeugs kann man aus den Informationen des WSDL-Dokumentes und der URI aus dem UDDI-Verzeichnis einen Proxy für die Interaktion mit dem Webdienst generieren lassen. Dies kann zum Beispiel eine Java-Klasse mit einer Methode sein, die alle für den Aufruf des Dienstes erforderlichen Parameter übergeben bekommt. Der Proxy kümmert sich dann darum, dass diese Parameter im richtigen Format in eine SOAP-RPC-Nachricht verpackt an den Web Service geschickt werden. Auerdem verarbeitet der Proxy das Ergebnis des Aufrufes und wandelt es beispielsweise in ein Java-Objekt um. Auf diese Weise verbirgt der Proxy, dass überhaupt ein Web Service benutzt wird (Abbildung 3.7) ([49]).

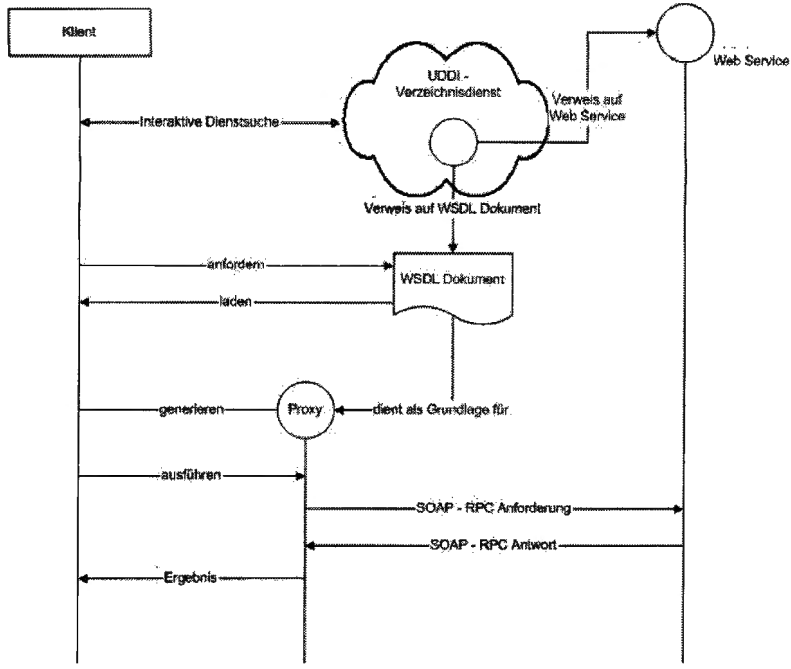


Abbildung 3.7: Sequenzdiagramm der Aktionen eines Web Service Klienten

3.3 Grid Services

Die Erweiterung des in den 90er Jahren entwickelten Grid Computing Ansatzes um die Prinzipien der Web Sevices wird mit Grid Services bezeichnet. Das Ziel ist, auf höherer Ebene heterogene Ressourcen über Verwaltungsdomänen hinweg gemeinschaftlich zu nutzen. Das bedeutet nicht nur, dass einzelne Anwendungen mit verteilten Prozessen realisiert

werden, sondern dass auch eine umfassende Infrastruktur zur Nutzung verteilter Ressourcen geschaffen wird [82].

3.3.1 Grid Computing

Grid Computing, ein Mitte der 90er Jahre eingeführter Begriff [10, 32] bezeichnet eine Architektur für verteilte Systeme, die auf dem World Wide Web aufbaut und die Web-Vision erweitert. Mit dem Grid Computing werden die Ressourcen einer Gemeinschaft, einer so genannten virtuellen Organisation, integriert. Ein Grid bezeichnete dabei eine nach dem Grid Computing-Ansatz aufgebaute Rechner-, Netzwerk- und Softwareinfrastruktur zur Teilung von Ressourcen mit dem Ziel, die Aufgaben einer virtuellen Organisation zu erledigen[13].
Historisch gesehen entstand das Grid Computing aus dem Wunsch, Hochleistungsrechner zu koppeln um extrem speicher- oder rechenintensive Probleme zu lösen. Dieser Ansatz wurde mittlerweile erweitert, um auch Desktop Computer und Rechencluster in ein Grid integrieren zu können. Das Standardisierungsgremium der Grid Technologie ist das *Global Grid Forum* (GGF) [82, 35].
Im Umfeld des Internet und im Speziellen des WWW unterscheidet man heute zwischen drei Arten des Grids [81] (siehe Abbildung 3.8).
Das Information Grid entspricht weitgehend dem WWW mit all seinen Ausprägungen. Um die Information anzubieten und zu verteilen, wird implizit das Ressource Grid genützt, welches Speicherkapzität, Rechenleistung und Netzwerke zur Verfügung stellt. Das Service Grid virtualisiert die Ressourcen und bildet die Daten- und Rechendienste des Information Grids und Ressourcen Grids auf eine abstrakte Ebene ab, wodurch sie ortstransparent nutzbar gemacht werden.
Grid Systeme können durch folgende Eigenschaften beschrieben werden [82]:

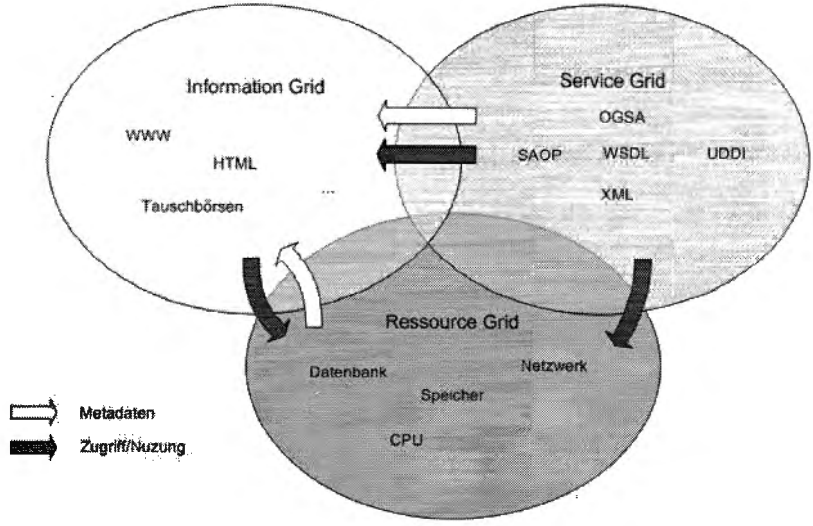


Abbildung 3.8: Grid Arten

- *Lokale Autonomie*: Grid Ressourcen werden nach den Regeln der lokalen Eigentümer betrieben. Die Grid Software hat keinen Einfluss auf lokale Entscheidungen.

- *Heterogenität der Ressourcen*: Grid Ressourcen können verschiedene Rechner, Softwareplattformen und Softwaresysteme umfassen

- *Skalierbarkeit*: Grid Dienste müssen transparent für den Nutzer je nach Aufgabenstellung die Anzahl der Knoten bestimmen und zur Verfügung stellen können.

- *Dynamik und Adaptivität*: Die Heterogenität bedingt die Möglichkeit von Ausfällen verschiedener Grid Ressourcen. Grid Systeme sollen in der Lage sein, flexibel auf Fehler in den verschiedenen Knoten reagieren zu können.

3.3.2 Service Architektur für Grid Dienste

Die Architektur für Dienste im Rahmen eines Grid Systems ist vom GGF im der *Open Grid Service Architecture* (OGSA) spezifiziert [31, 35]. Mit dieser erweiterbaren Spezifikation legt man Methoden fest, die jeder Dienst erfüllen muss, um sich Grid Dienst nennen zu dürfen. Das ermöglicht Anwendungen, die im Grid angebotenen Dienste zu nutzen und miteinander zu koppeln.

Die Spezifikation der Grid Dienste ist so ausgelegt, dass alle Grid Dienste auch gleichzeitig Web Services sind [82]. Darüber hinaus müssen sie als Grid Dienst noch die Operationen unterstützen, die im *Open Grid Service Infrastructure* (OGSI)-Standard [75] festgelegt sind.

3.3.2.1 Schnittstellen

Grid Dienste müssen maschinenelesbare Schnittstellen aufweisen die mit Hilfe der WSDL definiert werden. Ein Grid Dienst bietet Funktionen und Datenelemente (sogenannte *Service Data*) an und wird durch einen sogenannten *portType* beschrieben, der ebenfalls den Netzwerkport, durch den der Dienst angesprochen werden kann, definiert.

3.3.2.2 Zustandsinformation

Grid Dienste können einen Zustand besitzen, der sich in den Datenelementen widerspiegelt. Die Datenelemente können folgende Zustände annehmen [82]:

- *static*: Das Datenelement darf nicht verändert werden und sein Wert ist schon im WSDL Dokument des Dienstes definiert.
- *constant*: Das Datenelement darf nicht verändert werden und der Wert des Datenelements wird beim Starten des Dienstes definiert.

- *extendable*: Das Datenelement ist "erweiterbar" und es können während der Laufzeit des Dienstes neue Werte hinzugefügt werden. Diese Werte dürfen nach dem Hinzufügen nicht mehr verschwinden.
- *mutable*: Werte können dazukommen, dürfen sich ändern und auch wieder wegfallen.

Um Informationen über vorher unbekannte Dienste zu bekommen, stellt jeder Grid Dienst eine Methode *queryByServiceDataNames()* zur Verfügung.

3.3.2.3 Lifetime Management

Da Grid Services dynamisch sein sollen, besitzen sie ein explizites und ein implizites Lifetime Management.
Das explizite Lifetime Management umfasst die Möglichkeit, neue Dienste mittels sogenannter Factories zu instanzieren. Außerdem können Dienste mittels einer *destroy()*-Methode von außen beendet werden.

Das implizite Lifetime Management gibt für den Dienst Zeitpunkte vor, zwischen denen auf ihn zugegriffen werden kann. Als globale Zeit wird die GMT-Zeit (erweitert um den Wert *infinity* verwendet. Jeder Dienst hat zumindest zwei Zeitpunkte definiert:

- *after*: Der Dienst terminiert frühestens nach diesem Zeitpunkt.
- *before*: Der Dienst terminiert immer vor diesem Zeitpunkt.

3.3.2.4 Referenzieren von Diensten

Grid Dienst werden nach dem OGSI-Standard in einem zweisstufigen Verfahren referenziert.

Als erstes wird ein *Grid Service Handle* (GSH) definiert. Ein GSH ist ein abstrakter Verweis auf einen Dienst und ist unabhängig von der konkreten Implemetierung oder Instanz eines Dienstes. Mittels eines HandleResolvers wird ein GSH in ein oder mehrere *Grid Service Reference* (GSR) umgewandelt. Diese GSR verweist auf eine konkrete Implementierung eines Dienstes und enthält die nötige Information um den Dienst mittels dieser Implementierung zu nutzen.

Grid Applikationen verwenden GSHs zur persistenten Nutzung von Grid Diensten. Erst zur Laufzeit wird festgestellt, wo der geforderte Dienst verfügbar ist, indem der GSH in eine GSR umgewandelt wird. Mit der GSR nutzt die Applikation dann einen konkreten Dienst.

Außerdem definiert der OGSI-Standard die Struktur eines *Service Locator*, der dazu dient, Informationen über Dienste und Dienstreferenzen gemeinsam nutzen und austauschen zu können.

3.3.2.5 Aufruf von Grid Services

In Abbildung 3.9 wird ein Szenario für den Aufruf eines Grid Dienstes samt der zugehörigen Aktionen entworfen:

1. Eine Grid Applikation sucht als Klient in einer UDDI-Registry nach einer Factory für einen spezifisches Grid Service.

2. Die Factory wird anschlieend beauftragt eine Instanz des gewünschten Grid-Dienstes zu erzeugen, oder eine passende bereits vorhandene Instanz auszuwählen

3. Die Factory trägt einen Handle des Services im Handle Resolver ein.

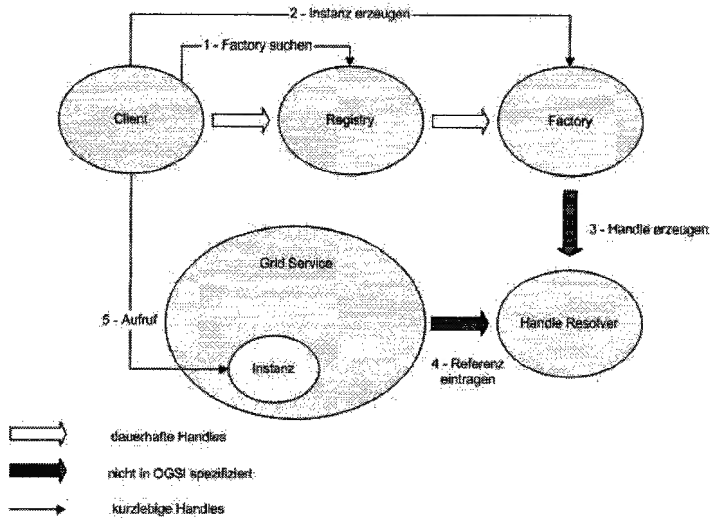


Abbildung 3.9: Aktionen beim Aufruf eines Grid Services

4. Der instanziierte Grid Dienst meldet sich beim Handle Resolver an.

5. Der Grid Service kann vom Klient über die Referenz aufgerufen werden

Die Lebenszeit kurzfristiger Handles wird vom Lifetime Management festgelegt. Ist die Lebenszeit eines Dienstes abgelaufen, muss sich eine Anwendung erneut an den Handle Resolver wenden, um einen aktuellen Verweis auf den gewünschten Dienst zu finden.

3.3.3 Web Services Ressource Framework

Grid Services können auf Basis von Web Services realisiert werden. Die Basistechnologien bleiben die bereits von den Web Services bekannten

Standards WSDL, SOAP und UDDI. Allerdings unterscheiden sich Grid Services von Web Services durch

- das zweistufige Namensschema, das aus Grid Service Handles und konkreten Grid Service References besteht.
- obligatorische Servcie Data, die für jeden Dienst Zustandsinformation und Metadaten enthält.
- standardisierte Instanzierung und Lifetime Management

Die Entwicklungen im Bereich Web Services und der OGSI Konzepte werden im Entwurf des *Web Service Ressource Framework* zusammengeführt [114]. Das führt dazu, dass die Umsetzung im WSRF in einigen Details anders aussieht, aber generell ist verbesserte Transparenz und Interoperabilität der Dienste nach wie vor das Ziel.

3.4 Proprietäre versus offene Technologien

Da im Rahmen dieser Arbeit Wert darauf gelegt wurde, das SimASP Framework möglichst mit Hilfe offener Technologien oder Standards zu realisiern, folgt ein kurzer Abschnitt über Vor- und Nachteile der offenen Technologien im Vergleich zu proprietären Technologien.

Offene Systeme können auf folgende Weise definiert werden:

"Open Source software development is software development that produces products where binary versions as well as complete sources of the product are available (usually on the Internet). Moreover open source software is published under a license, that allows any user to do arbitrary modifications to the sources and create new versions of the

software."[2] [86]

Entgegen der dem allgemeinen Verständnis bezeichnen offene Technologien nicht nur gratis erhältliche Software, sondern die Akzeptanz und der Gebrauch offener Protokolle[3] scheint im Rahmens des Arbeitens quer über Systemgrenzen hinweg (dem klassischen Umfeld, wenn verschiedene Firmen und Forschungseinrichtungen zusammenarbeiten) zumindest ebenso wichtig.
Beispiele offener Protokolle sind die im Kapitel Internettechnologien unter XML, Web Services und Grid Services beschriebenen Standards XML, SOAP, UDDI, OGSA, WSRF, ...

Die wesentlichen Fragestellungen bei kooperativer Arbeit zu einer Entscheidung offener versus proprietärer Systeme sind folgende [86]:

- Ist die Software für alle Projektpartner verfügbar und benutzbar?
- Ist die Persistenz der Projektdaten garantiert. Ist das Dokumentformat offen und ausreichend dokumentiert?
- Sind Veränderungen und Modifikationen an der Software erlaubt? Wenn ja, unter welchen Umständen?
- Ist die Softwarefirma ein verlässlicher Partner? Wird die Software auch noch in Zukunft unterstützt?

Ohne auf einzelne proprietäre Software eingehen zu wollen, kann man festhalten, dass offene Software und Protokolle in allen Punkten den Anforderungen entsprechen (am ehesten wurde früher noch der letzte Punkt

[2] Beispiele für solche Lizenzen sind die Gnu Public License [37] oder die Apache License [6].

[3] Protokolle sind Spezifikationen, die die Kommunikation verschiedener Systeme definieren.

kritisiert. Mittlerweile stehen in Open Source Projekten ein große Zahl von Entwicklern (fast) rund um die Uhr unentgeltlich zur Verfügung) und ihnen daher wenn möglich der Vorzug vor proprietären Systemen gegeben werden sollte [86].

Teil III

Das SimASP Framework

Kapitel 4

Allgemeine Beschreibung

Abbildung 4.1 zeigt den Aufbau des SimASP Frameworks. Clients sind mit dem Server Environment über das Internet verbunden. Die Verbindung über das Internet mittles Web Service Technologie bedeutet, dass der Austausch von Nachrichten mittels SOAP abgewickelt wird. Die Verwendung von SOAP als Kommunikationsprotokoll, welches unabhängig von der Netzwerkschicht MIME Attachments, RPC (Remote Precedure Call) und Messaging unterstützt, ermöglicht die Kommunikation mit dem Simulationsserver auf Basis von HTTP und damit den einfachen Zugang zu Netzwerken, die heutzutage im Allgemeinen durch Firewalls geschützt sind.

4.1 Modulare Services

Um dem Client möglichst maßgeschneiderte Dienstleistung anzubieten, wurden die möglichen Teilbereiche einer Simulationsstudie in Module zu zerlegt, die der Client je nach Bedarf wählen kann. Folgende Module sollen in einer ausgebauten Version implementiert sein:

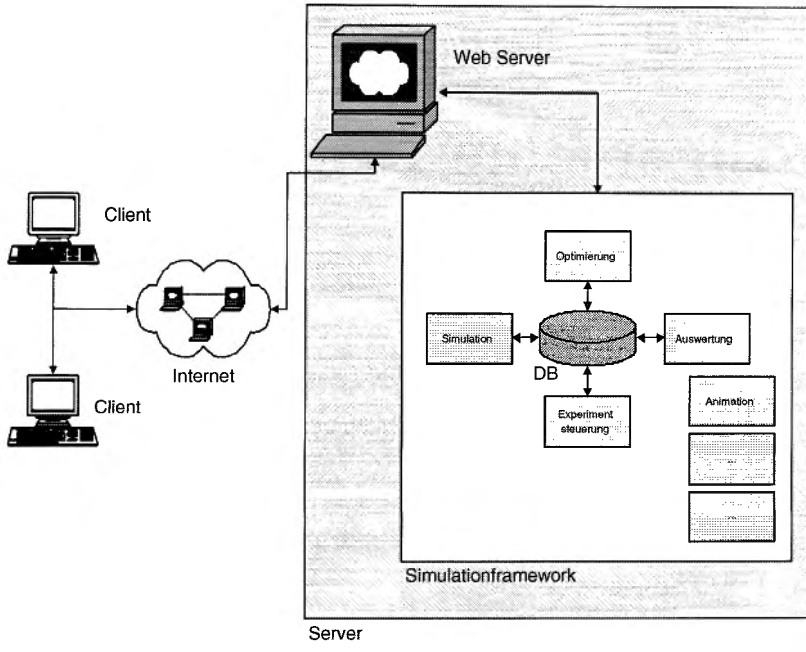


Abbildung 4.1: Der Aufbau des SimASP Frameworks

- *Simulatoren*: Je nach Bedarf und Modellierungsansatz des Benutzers sollen verschiedene Simulationsengines wählbar sein. In dem Prototypen des SimASP sind der prozessorientierte Simualtor JSIM und der ereignisorientierte Simualtor Simkit wählbar.

- *Experimentsteuerung*: Die Experimentbeschreibung des Prototypen umfasst momentan die Parameter

  - *Replications* definiert die Anzahl der Simulationsläufe pro Simulationsexperiment
  - *Initialseed* definiert den Startwert, der zur Berechnung von Zufallszahlfolgen verwendet wird.
  - *Stop Criteria* gibt an, nach welcher Bedingung ein Simulationsexperiment beendet wird. Es kann entweder die Endzeit oder die maximale Anzahl simulierter Entities angegeben werden. Per Default ist eine Endzeit eingestellt.
  - *Simulator* gibt an mit welchem Simulationsmodul die Simulation gestratet werden soll.

  Da die Modelldarstellung erweiterbar angelegt ist, kann dieses Modul beliebig ausgebaut werden.

- *Optimierung*: Ein eigenes Optimierungsmodul mit heuristischen Optimierungsalgorithmen ist geplant, aber noch nicht implementiert.

- *Auswertungen*: Momentan werden nach einem Simulationslauf grundlegende statistische Auswertungen sowohl der gesamten Simulation als auch der einzelnen Objekte der Simulation ausgegeben. Grundlage der Auswertugen sind die von den Simulatoren zur Verfügung gestellten Werkzeuge der Statistik. Mit diesen Daten ist die weitere Auswertung der Daten mit Programmen wie z.B. Excel

möglich.
In einer nächsten Version sollen Daten zu den Objekten einer Simulation gesammelt und nach Vorstellung des Benutzers gegenübergestellt und ausgewertet bzw. Spreadsheetfunktionen automatisiert werden.

- *Animationen*: Da Simulation heutzutage oft als Entscheidungsgrundlage für Managemententscheidung und die erarbeitet Information auch Nichtexperten zugänglich gemacht werden muss, ist Animation eine wesentliche Funktionalität von modernen Simualtonstools. Dieses Modul wird erst in einer zukünftigen Ausbaustufe des SimASP Frameworks umgesetzt.

Durch die Flexibilität des Frameworks ist die Erweiterung je nach Stand der Technik und dem Wunsch der Betreiber bzw. der Klienten jederzeit und unabhängig von den anderen Modulen möglich.

4.2 Datenhaltung mit XML

Da die modulare Funktionalität auch im Simulationsmodell repräsentiert werden soll, ist die Verwendung einer strukturierten Beschreibung Nahe liegend. Dadurch können die Inhalte der einzelnen Module unabhängig von einander verwendet und verwaltet werden. Der klassische ASP Ansatz sieht dafür eine (in den meisten Anwendungen relationale) Datenbank vor. Neben der Alternative der objektorientierten Datenbanken haben sich in den letzten Jahren vor allem im Bereich der Speicherung von strukturierten Textdaten XML-Datenbanken etabliert.
Da die Simulationsmodelle bereits in ein generisches XML Modell übersetzt werden, entsteht bei der Verwendung von XML als Speicherformat nur ein geringer bis kein Overhead.
Im Prototyp des SimASP werden die Modelle direkt als XML-Files, die

der Syntax des XML-Modellschemas entsprechen, gespeichert. Bei der Verwaltung großer Datenmengen und veschiedener Versionen der gleichen Modelle wird das Framework um eine Datenbank erweitert.

4.3 Kommunikation über Web Services

Die Kommunikation über das Internet wird mittels des *Vienna Grid Environment* (VGE) des Instituts für Softwarescience der Universität Wien umgesetzt.
Das VGE stellt eine serviceorientierte Grid Infrastruktur dar, die auf standardisierter Web Service Technologie basiert. Die Architektur für Dienste im Rahmen eines Grid Systems ist vom GGF im der *Open Grid Service Architecture* (OGSA) spezifiziert und ist so ausgelegt, dass alle Grid Dienste auch gleichzeitig Web Services sind. Sie werden als Grid Services bezeichnet.

4.4 Ablaufdarstellung

Abbildung 4.2 zeigt den Ablauf eines Simulation Service Providing Prozesses.

4.4.1 Erstellen eines DES Modells

Beim Client wird ein Simulationsmodell erstellt. Je nach Modellierungsansatz erfolgt die Modellerstellung entweder mittels eines GUI oder durch Erstellen eines Programmcodes.

4.4.2 Modellübersetzung in ein XML-File

Im nächsten Schritt wird das DES Modell in eine generische XML-Darstellung übersetzt. Dieser Übersetzungsprozess verwendet eine aus-

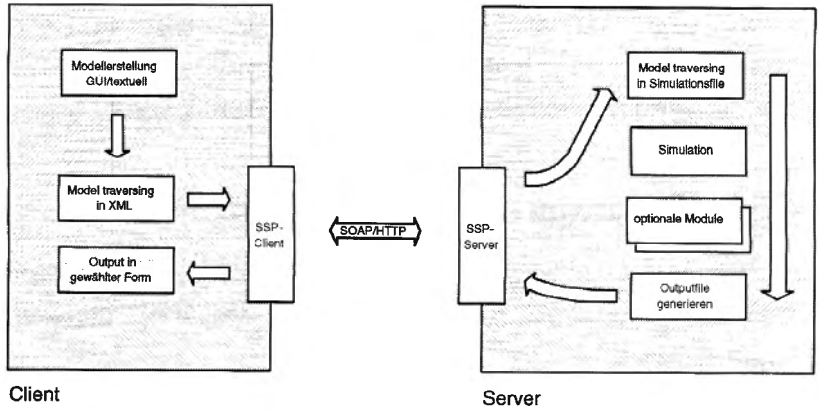


Abbildung 4.2: Der Ablauf eines Simulation Service Providing Prozesses

programmierte Schnittstelle zwischen dem ursprünglichen Modell und der Schemadefinition der XML-Modelle.
Der Prototyp des SimASP Frameworks arbeitet mit den XML-Files der DES-Modelle. Ein (noch zu entwerfendes) GUI braucht außer der Speicherung grafisch erzeugter Daten und der Übersetzung in ein XML-Modell keine weitere Funktionalität.

4.4.3 Übermitteln des Modells zum Service Provider

Im nächsten Schritt wird das generische Modell an den entsprechenden Service eines Service Providers geschickt. Optional kann diesem Schritt noch das Suchen eines Services in einer UDDI-Registry vorangehen.
Zum Suchen, Ansprechen des Services und Uploaden des Modellfiles wird die API des VGE-Client verwendet. Im Prototyp des Frameworks wird ein Webclient verwendet (siehe Abbildungen 4.3 bis 4.7)
Als erstes muss sich ein (menschlicher) Client authentifizieren.
Ist der Client berechtigt, den Grid Client zu benutzen, hat er auf der

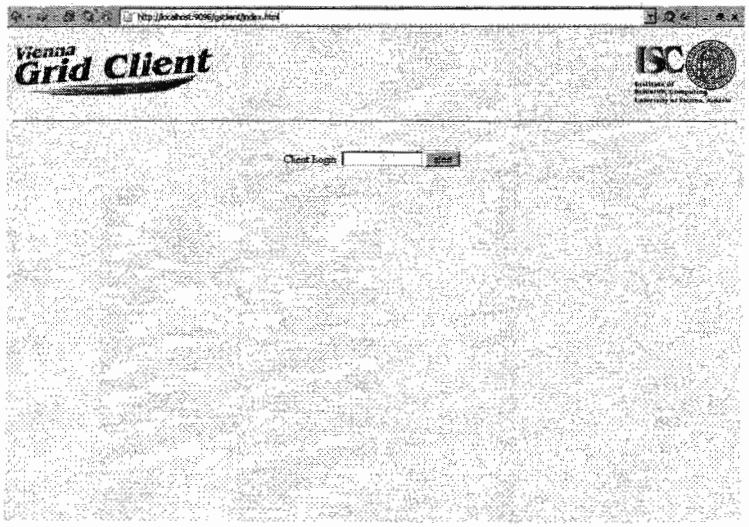


Abbildung 4.3: Identifizierung eines Client am VGE

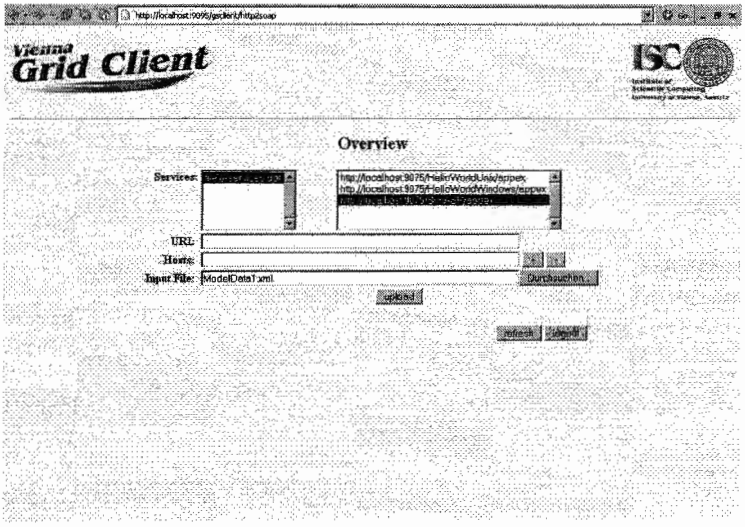


Abbildung 4.4: Auswahl eines Services des VGE

folgenden Maske die Möglichkeit, entweder die URI des Services anzugeben oder aus bereits vordefinierten Services den entsprechenden auszuwählen.
Im nächsten Schritt wird das Modellfile auf den Server geladen und der Service gestartet (siehe Abbildung 4.5).

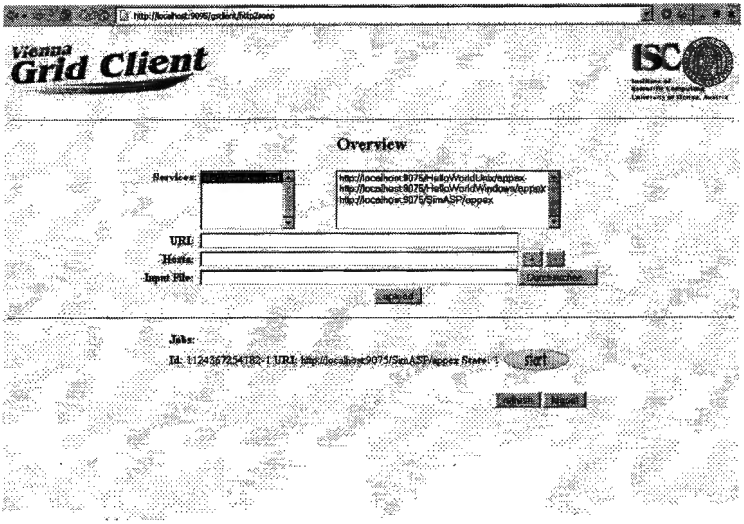


Abbildung 4.5: Start des Services

Mit dem Start wird dem Client eine Session-ID zugeteilt und dem Client zurückgeliefert.

4.4.4 Modellübersetzung in ein simulatorspezifisches File

Je nach gewählten Modulen und damit auch gewählten Simulatoren wird das XML-File in ein simulatorspezifisches File umgewandelt.

4.4.5 Simulation und Erzeugen des Outputs am Server

Nach erfolgreich durchgeführter Simulation bzw. Abarbeiten der gewünschten Funktionalität wird ein XML-Outputfile generiert, das dem XML Outputschema des SimASP entspricht.

4.4.6 Übermitteln des Outputs zum Client

Der Client hat während der Bearbeitung am Server die Möglichkeit, Statusinformation über den Fortschritt der Bearbeitung einzuholen oder den laufenden Service zu beenden (siehe Abbildung 4.6).

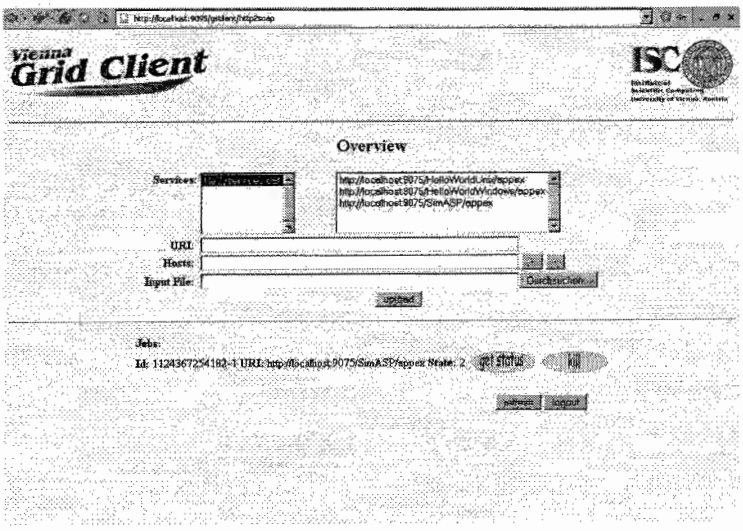


Abbildung 4.6: Mögliche Aktionen des Client während der Simulation am Server

Nach Beendigung des Simulationslaufes wird das Outputfile zum Dow-

nload angeboten (siehe Abbildung 4.7).

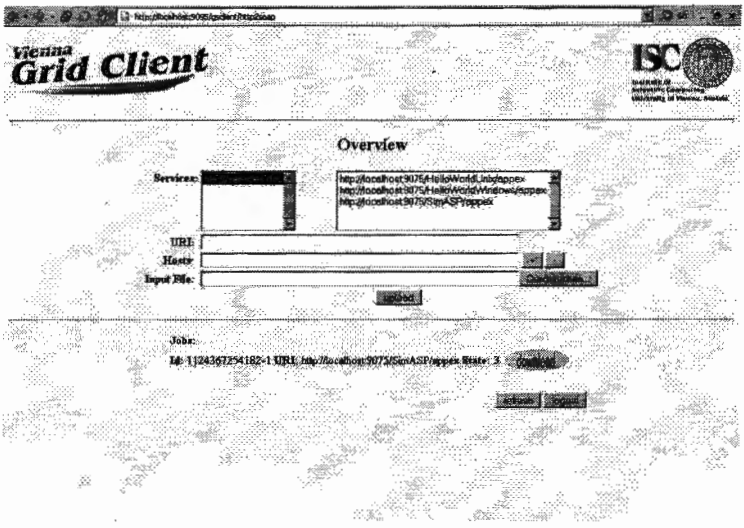


Abbildung 4.7: Downloaden des Outputfiles

Da das XML-Outputfile momentan nur statistische Auswertungen enthält, kann bzw. muss das File am Client je nach Wunsch weiterverarbeitet werden (das kann die Umwandlung in eine Spreadsheet- oder in grafische Darstellung bedeuten).

Kapitel 5

Die XML-Darstellung von DES Modellen

5.1 Das Input Schema

Das Simulationsmodell besteht aus einer Abfolge von Objekten - sogenannten *simEntities* (wie zum Beispiel Server, Queue, ...), die *Items* verarbeiten und mittels Connections verbunden werden. Dabei definieren die Connections die Flüsse der Items (siehe Abbildung 5.1).

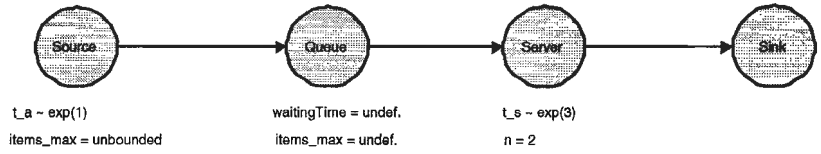


Abbildung 5.1: Job Shop Modell

Die Syntax sowohl der SimEntities als auch der Simulation sind in einem

XML Schema definiert. Dieses Input Schema definiert die Struktur, die eine XML-Instanz eines Simulationsmodells annehmen darf und damit implizit auch die erlaubte Struktur der Modelle selbst. Die und nur die im Inputschema definierten Objektinstanzen werden bei der weiteren Verarbeitung erkannt. Abbildung 5.2 zeigt die Baumstruktur des Inputschemas.

5.1.1 Benutzerdefinierte Datentypen

Es wurden die folgenden benutzerdefinierten Datentypen definiert. Die Beschreibung enthält jeweils die Elemente des Datentyps. Zur genauen Definition der Datentypen inklusive aller Attribute, Datentypen und Restriktionen wird auf den Anhang vewiesen.

5.1.1.1 LocationType

Der LocationType enthält die Koordinaten einer grafische Repräsentation von Simulationsobjekten. Diese sind als Attribute

- *x*: Die x-Koordinate einer grafischen Repräsentation eines Simulationsobjektes
- *y*: Die y-Koordinate einer grafischen Repräsentation eines Simulationsobjektes

definiert.

5.1.1.2 ConnectorType

Der ConnectorType definiert die Datenstruktur, die eine Verbindung zweier Simulationsobjekte erfüllen muss. Er besitzt folgende Elemente:

- *startObject*: Der logische Beginn einer Connection.

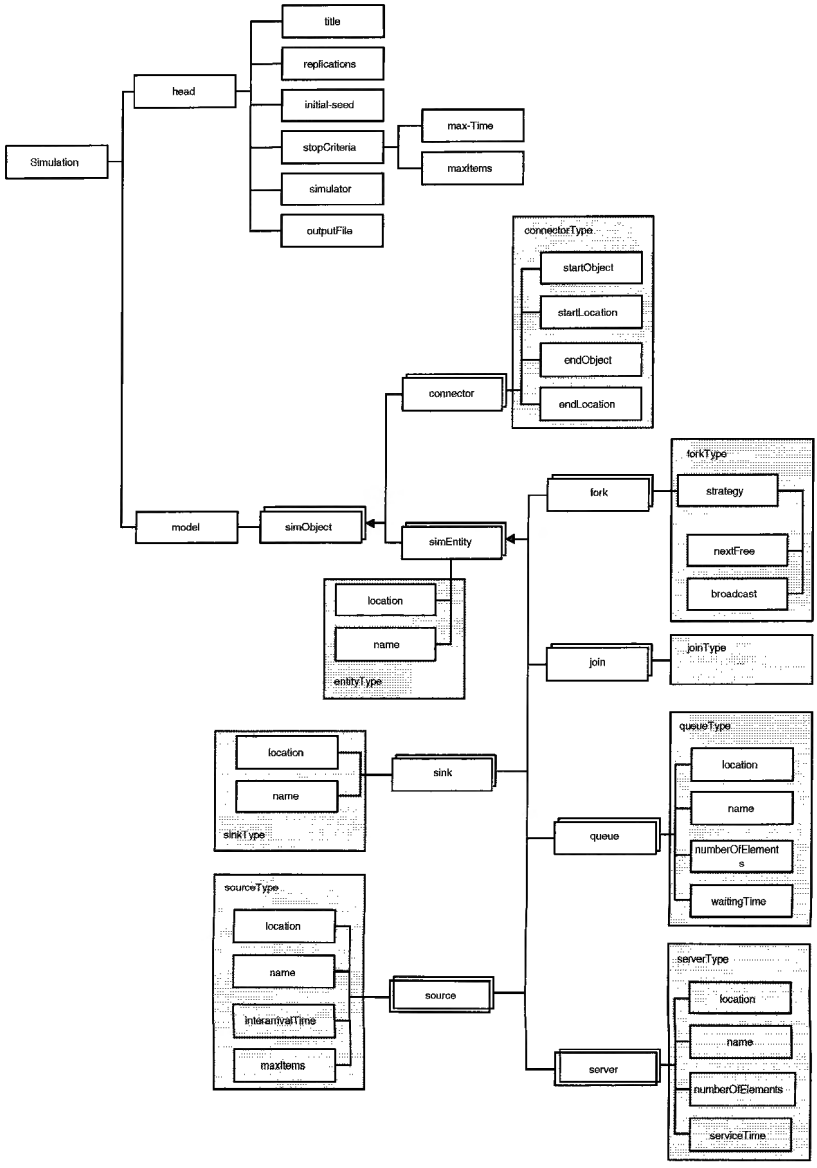


Abbildung 5.2: XML Schema eines DES Modells

- *startLocation*: Der grafische Beginn einer Connection. startLocation ist vom Datentyp locationType und enthält die Koordinaten einer grafischen Repräsentation eines Objektes.

- *endObject*: Das logische Ende einer Connection

- *endLocation*: Das grafische Ende einer Connection. endLocation ist vom Datentyp locationType und enthält die Koordinaten einer grafischen Repräsentation eines Objektes.

5.1.1.3 EntityType

Der EntityType definiert die grundlegende Datenstruktur aller grafischen Simulationsobjekte und besteht aus folgenden Elementen:

- *location*: Dieses Element ist vom Datentyp locationType und enthält die Koordinaten einer grafischen Repräsentation eines Objektes.

- *name*: Der Name des Simulationsobjektes.

5.1.1.4 SourceType

Der SourceType definiert die Datenstruktur einer Source und enthält folgende Elemente

- *location*: Dieses Element ist vom Datentyp locationType und enthält die Koordinaten einer grafischen Repräsentation eines Objektes.

- *name*: Der Name des Simulationsobjektes.

- *interarrivalTime*: Die Verteilung der Zwischenankunftszeiten. Wenn nicht anders definiert ist das eine Exponentialverteilung.

- *maxItems*: Die (optionale) maximale Zahl ankommender Items. Wenn kein Wert angegeben ist, wird der Wert unendlich angenommen.

5.1.1.5 QueueType

Der Datentyp einer Queue. Folgende Elemente sind definiert:

- *location*: Dieses Element ist vom Datentyp locationType und enthält die Koordinaten einer grafischen Repräsentation eines Objektes.
- *name*: Der Name des Simulationsobjektes.
- *numberOfElements*: Die maximale Nummer von Items, die eine Queue aufnehmen kann.
- *waitingTime*: Die Zeit, die ein Item in der Queue warten muss, bevor es weiterbefördert wird.

5.1.1.6 ServerType

Der Datentyp eines Servers. Folgende Elemente sind definiert:

- *location*: Dieses Element ist vom Datentyp locationType und enthält die Koordinaten einer grafischen Repräsentation eines Objektes.
- *name*: Der Name des Simulationsobjektes.
- *numberOfServer*: Die maximale Anzahl der gleichzeitig tätigen Server.
- *serviceTime*: Die Zeit, die für die Bearbeitung eines Items verwendet wird.

5.1.1.7 SinkType

Der Datentyp einer Sink. Da eine Sink Items gegenüber nicht agiert, sind nur folgende Elemente definiert:

- *location*: Dieses Element ist vom Datentyp locationType und enthält die Koordinaten einer grafischen Repräsentation eines Objektes.

- *name*: Der Name des Simulationsobjektes.

5.1.1.8 ForkType

Eine Fork splittet den Strom von Items in mehrere Ströme auf. Die Anzahl der ausgehenden Itemströme wird durch die ausgehenden Connections bestimmt und kann zwischen 1 und n liegen. Folgende Elemente sind definiert:

- *location*: Dieses Element ist vom Datentyp locationType und enthält die Koordinaten einer grafischen Repräsentation eines Objektes.

- *name*: Der Name des Simulationsobjektes.

- *strategy*: Eine Fork kann Items nach verschiedenen Strategien weiterleiten. Diese Strategien sind wie folgt definiert:

  - *nextFree*: Die Items werden der zufällligen nächsten freien Connection zugeordnet.

  - *broadcast*: Die Items werden vervielfältigt und jeder Connection wird ein Item zugeordnet.

5.1.1.9 JoinType

joinType ist der Datentyp eines Join-Elements. Dort werden die Elemente mehrerer Itemströme zusammengefasst und in einen Itemstrom eingespeist. Folgende Elemente sind definiert:

- *location*: Dieses Element ist vom Datentyp locationType und enthält die Koordinaten einer grafischen Repräsentation eines Objektes.
- *name*: Der Name des Simulationsobjektes.

5.1.2 Die Elemente des Input Schemas

Die Elemente beschreiben die Simulation und die SimEntities und haben jeweils einen definierten Datentyp, der ihre wesentlichen Ausprägungen definiert. Diesbezügliche Ausnahmen sind die Elemente *Head* und *Simulation*.

5.1.2.1 Simulation-Element

Das Simulation-Element stellt den Root-Knoten der hierarchischen Baumstruktur dar und repräsentiert das Simulatonsmodell. Das Simulation-Element hat die folgenden Kindelemente:

- *Head*
- *Model*

5.1.2.2 Head-Element

Das Head-Element enthält die Metainformation des Simulationsmodells. Diese besteht einerseits aus der Information zum Namen und den

gewünschten Modulen des Simulationsmodells, andererseits aus der Information, die zur Experimentsteuerung gedacht ist. Die Information ist in Form folgender Kindelemente repräsentiert:

- *Title* gibt den Namen des Simulationsmodells an.
- *Replications* definiert die Anzahl der Simulationsläufe pro Simulationsexperiment.
- *Initialseed* definiert den Startwert, der zur Berechnung von Zufallszahlfolgen verwendet wird.
- *Stop Criteria* gibt an, nach welcher Bedingung ein Simulationsexperiment beendet wird. Es kann entweder die Endzeit oder die maximale Anzahl simulierter Items angegeben werden. Per Default ist eine Endzeit eingestellt.
- *Simulator* gibt an mit welchem Simulationsmodul die Simulation gestratet werden soll.
- *Outputfile* definiert den Namen des Ausgabefiles, das vom Service Provider zurückgeliefert wird.

5.1.2.3 Model-Element

Das Model-Element definiert die Syntax des eigentlichen Simulationsmodells, das heißt die Objekte und Datentypen, die in einem Simulationsmodell definiert werden können.

5.1.2.4 SimObject-Element

Das SimObject-Element stellt das Elternobjekt für alle Objekte des Simulationsmodells dar, die Items nicht verändern. Alle Objekte eines Simulationsobjekts sind vom Typ SimObject.

5.1.2.5 Connector-Element

Konnektoren verbinden die Objekte des Simulationsmodells und geben die möglichen Routen in einem Modell für die Items vor. Sie erben ihre Parameter vom ConnectorType.

5.1.2.6 SimEntity-Element

Die "Klasse" SimEntitiy stellt die Elternklasse der Objekte eines Simulationsmodells dar, in denen die (passiven) Items bearbeitet werden und hat den Typ EntityType.

5.1.2.7 Source-Element

Das Source Element ist vom Typ SourceType und definiert die Eigenschaften einer Source eines DES-Modells.

5.1.2.8 Queue-Element

Queues können Items in Warteschlangen speichern und Items weiterleiten und sind vom Typ QueueType.

5.1.2.9 Server-Element

Server-Elemente stellen multiple Server dar und haben eine maximale Anzahl gleichzeitig tätiger Serviceprozesse. Sie stammen vom ServerType ab.

5.1.2.10 Sink-Element

Das Sink-Element dient zum Auffangen der Items im Simulationssystem und stammt vom SinkType ab.

5.1.2.11 Fork-Element

Das Fork-Element dient dem Aufsplitten von Item-Strömen und hat den benutzerdefinerten Typ ForkType. In einem Fork-Element können verschiedene Strategien gewählt werden. Entweder kann eine Fork alle Items broadcasten, dh. entlang aller Connections weiterleiten oder es wird die Strategie *nextfree* gewählt. Mit dieser Strategie wird von allen Connections die nächste zufällig freie gewählt und das Element dorthin geroutet.

5.1.2.12 Join-Element

Das Join-Element wird verwendet, um mehrere Ströme von Items zu einem Itemstrom zu vereinigen. Das Join-Element ist vom Datentyp JoinType.

5.2 Das XML Output Schema des statistischen Outputs

Als Defaultoutput wird beim Ausführen einer Simulation statistischer Output generiert und an den Client zurückgeliefert. Dieser Output enthält Statistiken zur gesamten Simulation und für jede Replikation eines Simulationslaufes Statistiken zu den einzelnen Simulationsobjekten (siehe Abbildung 5.3).

5.2.1 Benutzerdefinierte Datentypen

Es wurde der Datentyp *statCollectionType*, der die Sammlung statistischer Daten definiert, entworfen. Er enthält folgende Elemente:

- *ID*: Das ID-Element ist eine Referenz auf die Objekt-ID eines Simulationsobjekts. Damit ist die statCollection eindeutig einem Objekt einer Replikation eines Simulationsmodells zugeordnet.

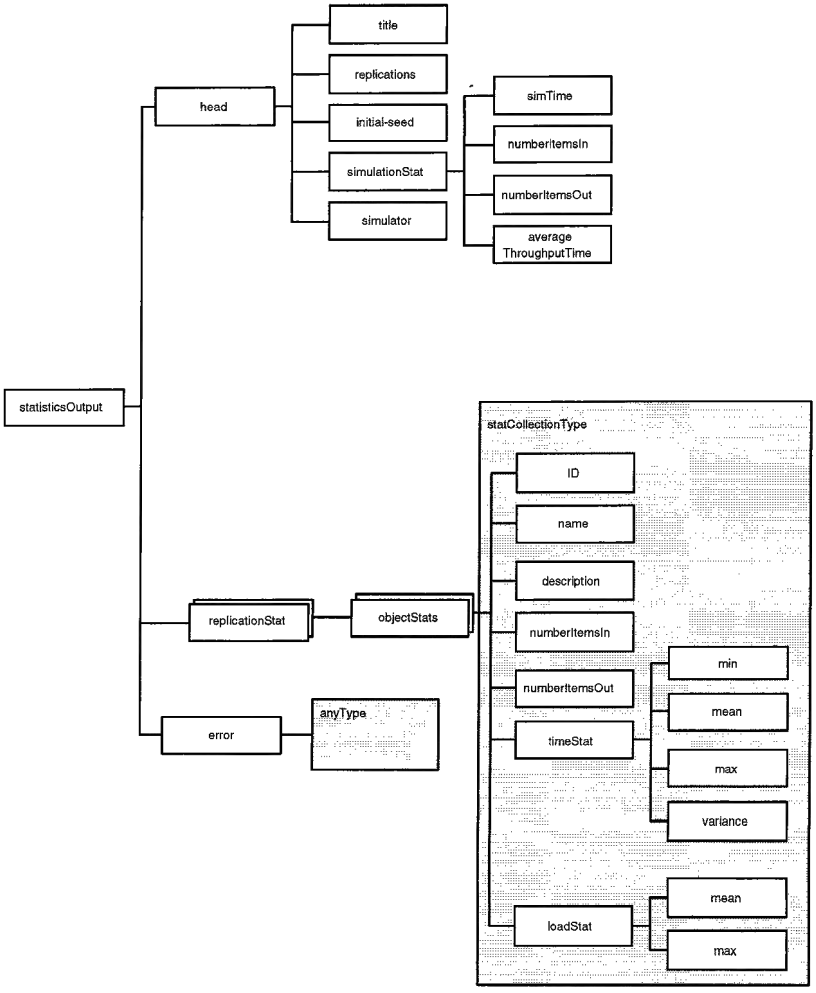


Abbildung 5.3: Das XML Schema des statistischen Outputs

- *name*: Der Name des Simulationsobjektes.
- *description*: Eine Beschreibung des Kontext' eines Simulationsobjektes.
- *numbersItemsIn*: Die Anzahl der während eines Simulationslaufes eingegangener Items.
- *numbersItemOut*: Optional die Anzahl der Items, die das Simulationsobjekt verlassen haben.
- *timeStat*: Jedem Simulationsobjekt können keine, eine oder auch mehrere Zeitstatistiken zugeordnet werden. Jede Zeitstatistik besteht aus den Elementen:
  - *min*: Dem Minimum der aufgetretenen Zeitwerte
  - *mean*: Dem Mittelwert der aufgetretenen Zeitwerte
  - *max*: Dem Maximum der aufgetretenen Zeitwerte
  - *variance*: Der Varianz der aufgetretenen Zeitwerte
- *loadStat*: Jedem Simulationsobjekt können keine, eine oder zwei Loadstatistiken zugeordnet werden. Diese bestehen jeweils aus den Elementen:
  - *mean*: Dem Mittelwert der Anzahl der gleichzeitig bearbeiteten Items
  - *max*: Dem Maximum der Anzahl der gleichzeitig bearbeiteten Items

5.2.2 Die Elemente des Output Schemas

Das Output Schema definiert die Möglichkeiten, die eine Instanz eines statistischen Outputs annehmen kann.

5.2.2.1 Error-Element

Wenn während der Simulation beim Service Provider ein Fehler auftritt, wird eine Fehlermeldung zurückgegeben. Da zu diesem Zeitpunkt nicht sicher ist, ob eine statistiche Auswertung Sinn macht, kann es passieren, dass der Output nur aus der Fehlermeldung besteht. Momentan werden als Fehlermeldungen nur Strings ausgegeben. Um das SimASP Framework für zukünftige Entwicklung offen zu halten, können Fehlermeldungen aber jeden im Schema zulässigen Datentyp annehmen.
Wenn kein Fehler auftritt, wird der eigentliche statistische Output generiert.

5.2.2.2 Head-Element

Das Head Element enthält die Metadaten zum generierten Output, die Statistik der gesamten Simulation und besteht aus folgenden Elementen:

- *title*: Der Titel des zugehörigen Simulationsmodells.
- *replications*: Die Anzahl der Simulationsläufe.
- *initial-seed*: Der dem Simulationsmodell mitgegebene Startwert zur Erzeugung von Zufallszahlen.
- *simulationStat*: Die Statistik des gesamten Simulationsexperiments. Diese Statistik besteht aus folgenden Elementen:
  - *simTime*: Die gesamte Simulationszeit.
  - *numbersItemsIn*: Die Anzahl aller in das Modell eingegangener Items. Das entspricht dem Wert aller von Sources erzeugten Items.
  - *numbersItemsOut*: Die Anzahl aller aus dem Modell ausgehenden Items. Das entspricht dem Wert aller in Sinks aufgefangener Items.

  - *averageThroughputTime*: Der durchschnittliche Wert, den ein Item im modellierten System verbracht hat.

- *simulator*: Der gewählte Simulator.

5.2.2.3 ReplicationStat-Element

Das ReplicationStat-Elementwird für jede Replikation eines Simulationslaufes angelegt und muss daher mindestens einmal vorkommen. Es enthält für alle Objekte eines Simulationslaufes die entsprechende Statistik. Daher bestet es nur aus einer Sammlung von *simObjectStats*. Diese enthalten für jedes Objekt einer Replikation eines Simulationslaufes die im Datentyp statsCollection definierten Werte.

Kapitel 6

Die Schnittstelle Client-Server

Als Schnittstellenumgebung für die Kommunikation zwischen Client und Server über das Internet wurde das Vienna Grid Environment (VGE) gewählt. Es wurde vom Institut für Softwarescience der Universität Wien im Rahmen des EU Projektes GEMSS (Grid-enabled Medical Simulation Services) entwickelt und ist als Prototyp Teil des GEMSS Grid Systems (GGS). Das GGS ist freie Software und wird unter der GNU Lesser Public License Version 2.1 oder höher zur Verfügung gestellt.

Das VGE stellt eine serviceorientierte Grid-Infrastruktur dar, die auf standardisierter Web Service Technologie basiert. Mittels des VGEs können High Performance Computing (HPC) Applikationen als Grid Services für on-demand Supercomputing zur Verfügung gestellt werden. Außerdem wird ein Application Programming Interface (API) für die Entwicklung von clientseitigen Applikationen angeboten.

Als eine Schlüsseleigenschaft stellt das VGE ein flexibles Quality of

Service (QoS) Verhandlungsmodell, mittels dessen Clients dynamisch und auf einzelnen Serviceanforderungen basierend, QoS Garantien bezüglich Ausführungszeit, Preis und weitere Eigenschaften mit potentiellen Service Providern aushandeln können. Das VGE ist im Wesentlichen in JAVA implementiert, basiert auf standardisierten Web Service Technologien wie WSDL, SOAP, WS-Security oder WSLA und verwendet die Open-Source Frameworks Tomcat und Axis für Service Hosting und Service Deployment.

6.1 VGE Architektur und Infrastruktur

6.1.1 Architektur

Das VGE ist auf einer serviceorientierten Architektur aufgebaut und ermöglicht die Integration mehrere Grid Clients und mehrerer Grid Services, einer oder mehr Service Registries und einer Certifikations Autorität (siehe Abbildung 6.1).

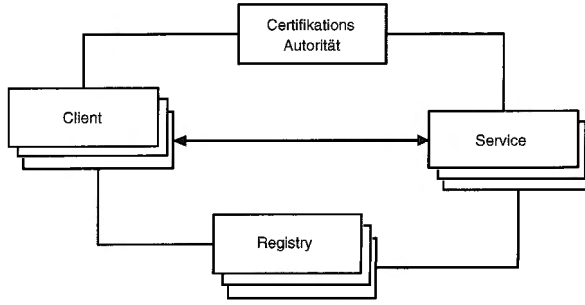


Abbildung 6.1: High-level Architektur des Frameworks

Die VGE Services kapseln ursprüngliche HPC Applikationen, die auf Clustern oder anderen parallellen Rechnerstrukturen zur Verfügung stehen und bieten Operationen für die Ausführung, Job Monitoring, Fehlerbehebung und QoS-Unterstützung auf Applikationsebene.
Die Services werden mittels WSDL angeboten und können mittels SOAP unter Einhaltung von gebräuchlichen Sicherheitsstandards in Anspruch genommen werden. Das VGE Framwork ermöglicht Service Providern das Anbieten bereits bestehender Applikationen als Services, auf die über das Internet zugegriffen werden kann.
Alle Interaktionen eines Clients mit einem Service werden vom Client initiiert. Callbacks oder andere Mechanismen zur Notifikation werden nicht verwendet.
Die VGE Client Applikationen laufen normalerweise auf einem PC oder einer Workstation und verwenden die VGE Client API für die Kommunikation mit den Services über die VGE Middleware. Die clientseitige Applikation steuert die Erzeugung von Inputdaten, den QoS Verhandlungsprozess und das Postprocessing der Outputdaten des Services.
Die Registries verwalten eine Liste der Service Provider sowie der von ihnen unterstützten Services und werden während des QoS Verhandlungsprozesses verwendet.
Die Zertifikationsautorität sichert die Identität jedes Clients als auch jedes Service Providers. Außerdem verwalten sie die Sicherheitsmechanismen auf Transport- und Messagelayerebene.

6.1.2 Der Zugang zu den Services

Das VGE unterstützt ein flexibles Zugriffsmodell zu den Services. Es werden drei Phasen unterschieden:

- *Die Businessphase*: In dieser Phase wird das Preismodell gewählt. Momentan wird nur ein Preismodell unterstützt, das das Zahlen pro Benutzung vorsieht. Dadurch kann es allerdings passieren, dass für jede Serviceanfrage ein anderer Preis, je nach aktuell verwendeten Ressourcen, zur Anwedung kommt.

- *Die QoS - Verhandlungsphase*: Während dieser Phase werden von einem Client mit den Service Providern bestimmte QoS Garantien ausgehandelt und der Client wählt dann den Provider mit dem besten Angebot. Als Eigenschaften, über die QoS Garantien ausgehandelt werden können, werden momentan Dauer der Ausführung und Preis unterstützt. Die QoS Garantien werden als XML Dokument nach Web Service Level Agreement (WSLA) Spezifikation gespeichert und wenn ein Client mit einem Service Provider eine Einigung erzielt hat, wird ein QoS Vertrag ausgetauscht.

- *Die Applikationsausführungsphase*: Diese Phase beteht üblicherweise aus dem Upload von Eingangsdaten, Starten der Remote Ausführung des Applikationsservices und dem Download der Resultate.

6.1.3 Client Infrastruktur

Die Hauptkomponenten des Client Environments werden in Abbildung 6.2 gezeigt.

- *Service Discovery*: Diese Komponente enthält verschiedene Methoden, um Services zu entdecken, UDDI-Registries aufzusuchen und Service Proxies zurückzuliefern.

- *State Management*: Diese Komponente verwaltet die Zustände auf der Client Seite und stellt Mechanismen zur Verfügung, um bei

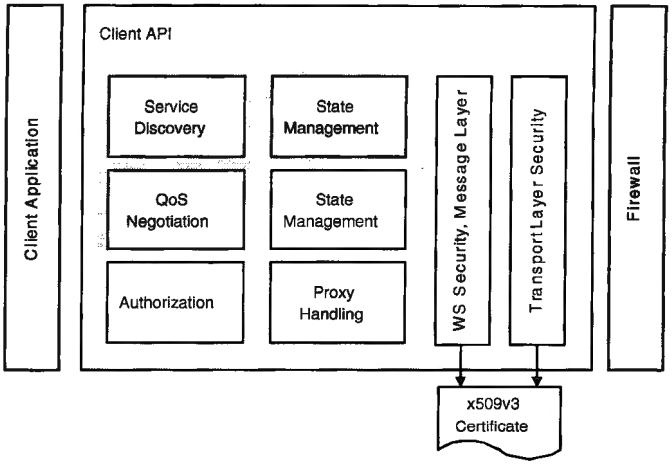


Abbildung 6.2: Client Infrastructure

lange laufenden Service Requests den Zustand des Clients zu speichern und zur späteren Weiterarbeit wiederherzustellen.

- *Desriptor Handling*: Diese Komponente dient dem Erzeugen und Verwalten von Request Descriptors und QoS Descriptors.

- *QoS Negotiation*: Diese Komponente ermöglicht Quality of Service-Verhandlungen mit mehreren potenziellen Serviceanbietern.

- *Proxy Handling*: Diese Komponente wickelt die Kommunikation mit einem spezifischen Service ab.

- *Authorization*: Die Komponente verwaltet die Authentifizierung des Clients.

6.1.3.1 Die Client API

Das VGE bietet dem Client eine High Level API, die die zugrundeliegende Interaktion mit den Services versteckt und dem Client ermöglicht, remote Grid Services auf einem hohem Abstraktionsniveau zu entdecken und zu bedienen (siehe Abbildung 6.3).

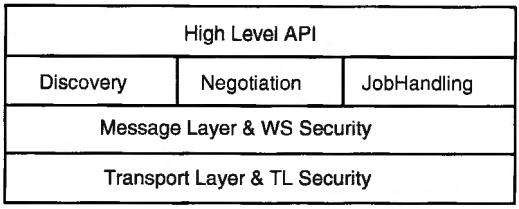


Abbildung 6.3: Die Abstraktionsschichten der API

Die API besteht aus einer Ansammlung von Klassen, deren Methoden es erlauben, Grid Services zu suchen, zu entdecken und auszuwählen. Diese Methoden liefern jeweils eine Service Proxy zurück, die zur weiteren Interaktion mit dem Service verwendet wird.

Zum Suchen von Services stehen folgende Varianten zur Verfügung:

- *ServiceProxy getService(String endpoint)*: Als einfachste Methode wird ein Service mittels dessen URI ausgewählt.
- *ServiceProxy getServices(String[] attrib)*: Ein Service wird auf Basis einer Menge vordefinierter Attribute in den VGE Registries gesucht und ausgewählt.
- *ServiceProxy getService(String[] attrib, QoSDsc q, ReqDsc r)*: Zusätzlich können noch bestimmte QoS-Anforderungen bestehen.

Die Erzeugung der clientseitigen Stubs wird dem Benutzer vom VGE abgenommen. Aus Sicht des Benutzers sind allein die Service Proxies für das Jobhandling, den Upload und Download von Daten und das Status- und Errorhandling zuständig.
Security Aspekte werden auf der untersten Abstraktionsebene behandelt und sind vom Benutzer vollständig verborgen.

6.1.3.2 Das Client Environment

6.1.3.2.1 Systemvoraussetzungen

- Plattformen: Es werden die Plattformen Solaris 9, Win2000, XP, NT und Linux unterstützt.
- Browser: Netscape 7.0, IE 6.0
- Java Virtual Machine (JVM): Die JVM muss Version >= 1.4.0 haben

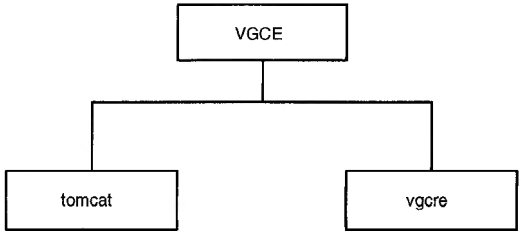


Abbildung 6.4: Die Ordner Struktur des Vienna Grid Client Environments

6.1.3.2.2 Ordnerstruktur: Abbildung 6.4 zeigt die Ordnerstruktur des VGCE. Das Verzeichnis VGCE enthält zwei Unterverzeichnisse.

- *tomcat*: Das Verzeichnis eines Jakarta-Tomcat V.4.1.30

- *vgcre (Das Vienna Grid Client Runtime Environment)*: Dieses Verzeichnis enthält die Build-Files, .jsp-Files, Class Files sowie Libraries.

6.1.4 Service Providing Infrastruktur

Applikationen werden als Web Service mittels eines Webservers und eines Servicecontainers zur Verfügung gestellt (Apache Tomcat/Axis) [5, 7, 98].

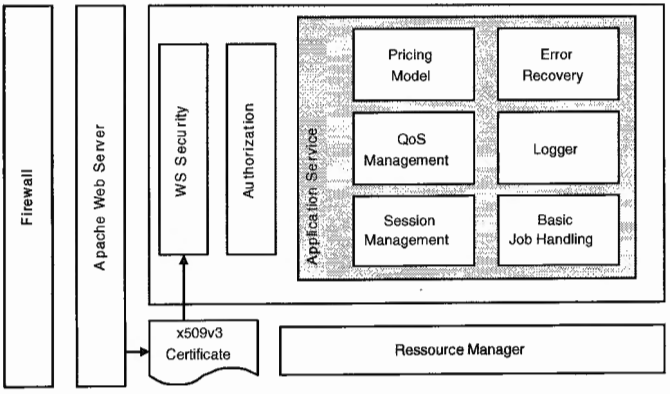


Abbildung 6.5: Service Providing Infrastructure

Abbildung 6.5 zeigt die wesentlichen Komponenten der Infrastruktur des Service Providers (vergleiche [9]):

- *Quality of Service Management*: Das QoS Management offeriert dem Client während des Verhandlungsprozesses Angebote und kommuniziert mit dem Ressourcen Manager um die Verfügbarkeit von Ressourcen zu sichern.

- *Ressource Manager*: Der Ressourcen Manger ist ein abstraktes Interface zu Scheduling Systemen, die die Reservierung von Ressourcen im Vorhinein unterstützen.

- *Error Recovery*: Das Modul steuert Check Pointing und Restarting, wenn es von der Applikation unterstützt wird.

- *Pricing Model*: Das Preismodell dient der Festlegung von Preisen der CPU-Zeit und des Arbeitsspeichers.

- *Session Management*: Das Session Management vergibt für jeden Client eine eigene ID und verwendet diese um eine Session auch über mehrere Aufrufe zu verwalten bzw. Statusinformation auszutauschen.

- *Logger*: Das Logger Modul ist für die Aufzeichnung von Beobachtungsdaten und Events im System des Service Providers zuständig.

6.1.4.1 Generische Applikationsservices

Die Transformation einer nativen Applikation in ein Grid Service basiert auf dem Prinzip des generischen Applikationsservices [9]. Abbildung 6.6 zeigt den Aufbau eines solchen Applikationsservices.

Das generischen Applikationsservice ist eine konfigurierbare Softwarekomponente, die einem Client eine native Applikation als allgemeines Service mit standardisierter Funktionalität anbietet. Diese wird durch drei Interfaces realisiert:

- *JobHandling-Interface*: Dieses Interface wird für die Basisfunktionalität einer Grid Service Applikation verwendet. Es bietet einem Client folgende Methoden an:

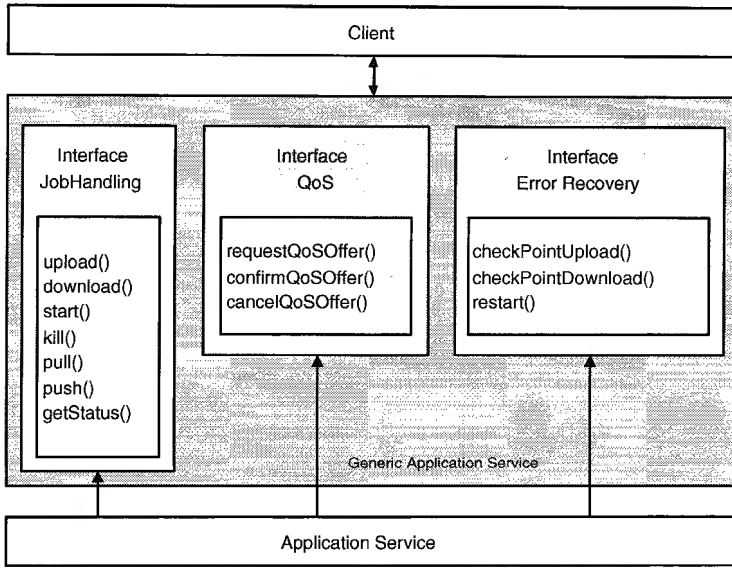


Abbildung 6.6: Aufbau des Generischen Applikationsservices

  - *upload()*: Uploading eines Simulationsmodells zum Serviceprovider
  - *download()*: Downloading von Output-Data
  - *start()*: Starten eines remoten Applikationsservices
  - *kill()*: Beenden eines remoten Applikationsservices
  - *push()* und *pull()*: Diese Methoden werden für den Datenaustausch zwischen verschiedenen Applikationen verwendet.
  - *getStatus()*: Download eines applikationsspezifischen Downloadfiles, das Statusinformation zu einem Applikationsservice enthält

- *QoS-Interface*: Dieses Interface wird im QualityOfService-Verhandlungsprozess verwendet und bietet folgende Methoden an:
  - *requestQoSOffer()*
  - *confirmQoSOffer()*
  - *cancelQoSOffer()*

- *ErrorRecovery-Interface*: Dieses Interface bietet im Falle eines aufgetretenen Fehlers folgende Methoden zum Resumen und Restarten eines Services an:
  - *checkPointUpload()*
  - *checkPointDownload()*
  - *restart()*

Das Verhalten des Services bei einem Aufruf dieser Methoden ist in einem zugehörigen XML-Application Descriptor definiert.

6.1.4.2 Application Descriptor

Ein Application Descriptor ist ein XML Dokument, das Metadaten zu einem Service enthält und die Semantik eines generischen Applikationsservices definiert [9].
Es werden vom Service Provider das Inputfile, das Outputfile, ein Script zum Starten des Applikationsservices, Skripts zur Bereitstellung von Statusinformation und ein Finishfile spezifiziert. Das bedeutet, das nur der Service Provider entscheidet, welche Skripts auf seinem Server ausgeführt werden.
Wenn QoS Unterstützung gefordert wird, können Anfrageparameter, Anfrage-Metadaten und Maschinenparameter spezifiziert werden. Während eines QoS-Verhandlungsprozesses muss der Client dann aktuelle Werte für die Anfrageparameter in einem Request Descriptor zur Verfügung stellen.
Der Application Descriptor wird vom Service Provider veröffentlicht, um dem Client die benötigten Parameter zum Aufruf eines Services bzw. für das Erstellen eines Request Descriptors zur Verfügung zu stellen.

6.1.4.3 Deployen und Betreiben der Services

Um ein Application Service zu erzeugen, muss die Applikation im VGE Hosting Environment, das heißt Apache Tomcat/Axis, installiert werden und ein entsprechender Application Descriptor erzeugt werden.

Zur Erstellund dieses Application Descriptors und zum Verteilen eines Services wird ein Deploymenttool zur Verfügung gestellt, das dem Provider die Spezifiaktion der benötigten Parameter und das Deployen des Services mittels eines GUIs ermöglicht. Das Deploymenttool erzeugt den XML Application Descriptor und generiert einen entsprechenden Web

Sevice der die Applikation kapselt, veröffentlicht den Web Service in einer UDDI-Registry und verteilt den Service im Hosting Environment.

Damit kann ein Applikationsservice remote von einem Client aufgerufen werden. Das VGE stellt Services multihreaded zur Verfügung. Für jeden aufrufenden Client wird eine eigene Session erzeut und intern verwaltet.

6.1.4.4 Das Service Environment

6.1.4.4.1 Systemvoraussetzungen

- Plattformen: Es werden die Plattformen Solaris 9, Win2000, XP, NT und Linux unterstützt.
- Browser: Netscape 7.0, IE 6.0
- Java Virtual Machine (JVM): Die JVM muss Version >= 1.4.0 haben

6.1.4.4.2 Ordnerstruktur: Abbildung 6.7 zeigt die Ordnerstruktur des Vienna Grid Service Environments (VGSE). Das Verzeichnis VGSE enthält drei Unterverzeichnisse.

- *tomcat*: Das Verzeichnis eines Jakarta-Tomcat V.4.1.30
- *vgsre (Vienna Grid Service Runtime Environment)*: Dieses Verzeichnis enthält die Build-Files, .jsp-Files, Class Files, sowie Libraries und die Service Descriptoren.
- *services*: Das Service Verzeichnis enthält die Arbeitsverzeichnisse aller installierten Services. Wird ein Applikationsservice neu verteilt, erzeugt das System automatisch ein zugehöriges Verzeichnis. In dem darunter liegenden *config*-Verzeichnis sind der Service Descriptor und ein startup-File gespeichert. Der Service Descriptor

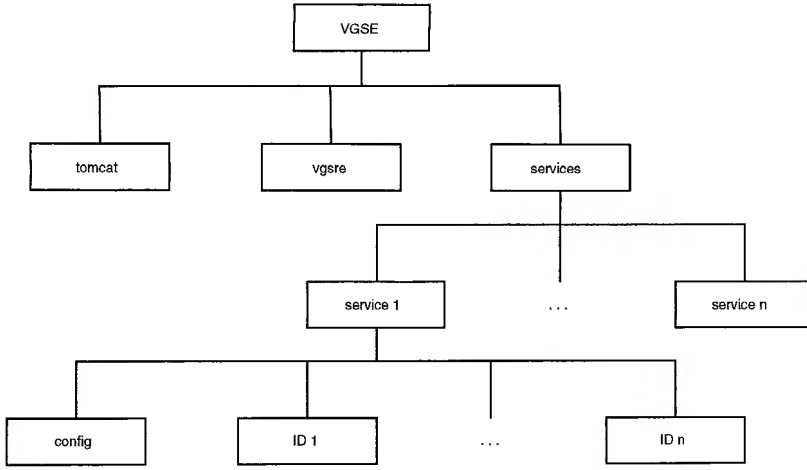


Abbildung 6.7: Die Ordner Struktur des Vienna Grid Client Environments

enthält den Pfad des Arbeitsverzeichnisses des Service und die Pfadnamen des Startscriptes, des Input- , Output- und Finish-Files.
Inerhalb des Arbeitsverzeichnisses wird für jede Clientsession ein Verzeichnis erstellt, in dem Input- und Outputfiles abgelegt werden.

6.1.4.4.3 Das Deploymenttool: Im vgsre-Verzeichnis wir zum Erzeugen des Servicedescriptors und zum Verteilen des Services ein grafisches Deploymenttool zur Verfügung gestellt. Abbildung 6.8 zeigt einen Screenshot des Deploymenttools.

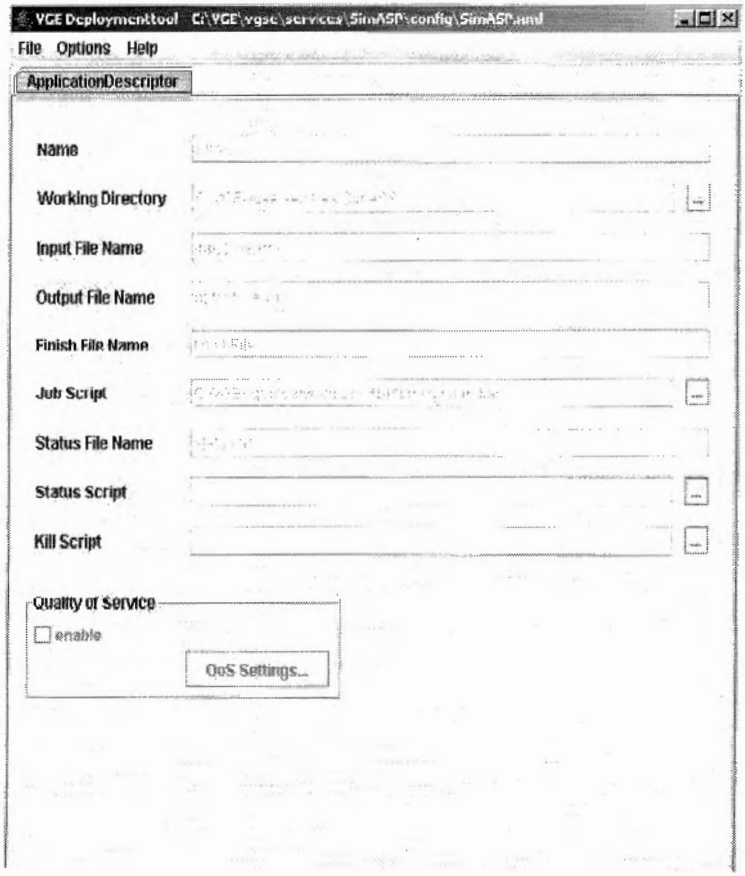


Abbildung 6.8: Erzeugen eines Applikationsdescriptors mit dem Deploymenttool

6.2 Security

Die Certificate Authority stellt eine *Public Key Infrastructure* (PKI) auf der Basis von X.509-Zertifikaten zu Verfügung. Es werden für den Client und für den Server entsprechend der Internet X.509 PKI Certificate Policy and Certification Practices Zertifikate erstellt, die der Identifikation von Client und Server dienen [9, 20].

Die Zertifikate verwenden ein zweischichtiges Sicherheitskonzept:

- *Transport Layer Security*: Es wird SSL via https verwendet.
- *Message Layer Security*: Es wird Web Service Security entsprechend WSS4J verwendet [116, 117].

6.3 Registry Services

Mittels dem VGE können mehrere Service Registries verwendet und betrieben werden. Die Registries sind als eigene Web Services konzipiert, entsprechen dem UDDI-Standard und werden mittels einer High Level API sowohl von den Service Providern als auch von den Clients verwendet.

Der Service Provider veröffentlicht dort seine angebotenen Services und spezifiziert eine beliebige Anzahl von zu einer Applikation gehörender Attribute in Form von name/value-Paaren. Diese Attribute können dann vom Client verwendet werden, um Services auszuwählen, die seinen spezifischen Anforderungen entsprechen.

Kapitel 7

Simulationsmodule

Als erste Umsetzung des Ansatzes, Simulationsdienste in mehrere Module zu zerlegen und getrennt anzubieten, wurden zwei Simualtoren gewählt, die vom methodischen Ansatz prinzipiell verschieden sind. Damit wurde gezeigt, dass das Konzept des generischen Simulationsmodells, das erst zur Laufzeit der Simulation in ein simulatorspezifisches File übersetzt wird, zumindest für einfache Modelle funktioniert.
Im Folgenden wird eine Einführung in die Simulationsmodule präsentiert.

7.1 Simkit

Simkit ist ein auf JAVA basierendes Softwarepaket, das Discrete Event Simulation (DES)-Modelle in purer Form implementiert. Das heißt, dass außer der DES-Sicht der Realität keine Konzepte wie z.B. Prozesse oder Ressourcen unterstützt werden. Dieser Ansatz macht zwar manche einfache Modelle komplizierter, bietet dafür aber mehr Flexibilität als ein prozessorientierter Ansatz. Alle prozessorientierten Modelle können in

einen ereignisorientierten Ansatz übersetzt werden - das Gegenteil ist nicht richtig [16].
Die Modellierung von Simkit-Modellen ist eng mit der Modellierungstechnik der Eventgraphen verknüpft.

7.1.1 Konzepte der DES

Die Basiskonzepte der DES sind die Zustandsvariablen und die Ereignisse. Das Modell emuliert das modellierte System, indem Trajektorien der Zustandsvariablen erzeugt werden. Modelle der DES sind dadurch charakterisiert, daß diese Trajektorien stückweise konstant verlaufen. Ereignisse treten an den Zeitpunkten auf, an denen sich zumindest eine Zustandsvariable ändert. Wenn ein Ereignis auftritt, vergeht keine Simulationszeit. Die Simulationszeit vergeht nur zwischen den Ereignissen. Das Abarbeiten der Ereignisse wird durch die Future Event List (FEL) kontrolliert, die nur aus einer Abfolge der zukünftig auftretenden Ereignisse mit je einem entsprechenden Zeitstempel besteht. Wenn ein neues Ereignis auftreten soll, wird eine Eventnotitz an die FEL geschickt und dort das entsprechende Ereignis eingetragen. Diese Notitz besteht erstens aus der Information welches Ereignis eingetragen werden soll und zweitens aus dem Zeitpunkt, wann das Ereignis stattfinden soll. Die FEL ordnet die Ereignisse nach den Zeitpunkten an denen sie auftreten sollen.
Wenn zwei Ereignisse zur gleichen Zeit auftreten sollen, müssen sie priorisiert werden. Das kann entweder manuell angegeben oder zufällig gewählt werden (vergleiche [61]).

Die FEL wird von einem Discrete Event Flow Algorithmus kontrolliert, der den Ablauf der Zeit steuert. Bei jeder Iteration sucht der Algorithmus in der FEL nach eingeplanten Ereignissen. Wenn nichts zu tun ist und die FEL leer ist, wird die Simulation beendet. Ist nichts zu tun

und die FEL ist nicht leer, wird die Simulationszeit auf den Wert des als nächstes auftretenden Ereignisses gesetzt und die mit dem Ereignis assozierte Zustandsveränderung wird ausgeführt. Das Abbruchkriterium der leeren FEL bedeutet, dass zu Beginn der Simulaiton zumindest ein Ereignis in die FEL gestellt werden muss. Dieses Ereignis wird gemäß Übereinkunft mit *Run* bezeichnet [89]. Dieses *Run*-Event ist für das Initialisieren der ersten Simulationsereignisse zuständig.

Wenn ein Ereignis eintritt, werden folgende Schritte abgearbeitet:

1. Per Konvention werden alle Zustandsveränderungen, die mit diesem Ereignis verknüpft sind, durchgeführt.

2. Alle durch das Ereignis spezifizierten Ereignisse werden (eventuell neu) geplant.

3. Eine neue Eventnotitz wird an die FEL geschickt

Diese Reihenfolge kann prinzipiell auch geändert werden, sollte aber innerhalb eines Modells unverändert bleiben.

7.1.2 Eventgraphen

Eventgraphen sind eine Möglichkeit der grafische Repräsentation von DES-Modellen und sind die einzige Möglichkeit die Logik einer Future Event List (FEL) direkt grafisch zu modellieren [15].

Jeder Eventgraph besteht aus Knoten und gerichteten Kanten.
Jeder Knoten eines Modells korrespondiert mit einem Ereignis beziehungsweise mit einem Zustandsübergang und jeder Kante entspricht das Schedulen von Ereignissen in der FEL. Jeder Kante kann entweder eine boolsche Bedingung oder ein Timedelay zugeordnet sein. Abbildung 7.1

zeigt das fundamentale Konstrukt eines Eventgraphen:

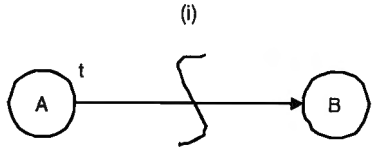


Abbildung 7.1: Basiselement eines Eventgraphen

Das Ereignis A scheduled ein Ereignis B nach t Zeiteinheiten unter der Vorraussetzung, dass die Bedingung (i) den Wert *true* annimmt. Die Bedingung (i) wird erst nach den Veränderungen, die das Ereignis A auslöst, ausgewertet. Wenn keine zeitliche Verzögerung zwischen dem Auslösen des Ereignisses B durch A und der Ausführung von B liegt, fällt die Angabe von t weg. Analog kann man die Bedingung (i) weglassen, wenn das Ereignis A immer das Ereignis B auslöst. Damit sind bereits alle Konstruktionselemente eines Eventgraphen definiert:

- Knoten
- Kanten mit den Varianten
  - Kante ohne Zeitverzögerung und ohne Bedingung
  - Kante mit Zeitverzögerung
  - Kante mit Bedingung
  - Kante mit Zeitverzögerung und Bedingung

Mit diesen Komponenten können bereits alle DES Modelle repräsentiert werden [15, 88, 89], wobei der rein ereignisorientierte Ansatz vom Modellentwickler bei komplexeren Aufgabenstellungen einen höheren

Abstraktionslevel erfordert.

Mit dem nach Konvention geforderten *Run*-Event könnte ein Arrivalprozess folgendermaßen definiert werden (siehe Abbildung 7.2):

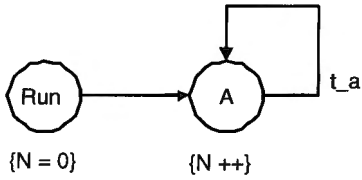


Abbildung 7.2: Arrivalprozess ohne Abbruchsbedingung

Mit dem Start der Simulation wird das *Run*-Ereignis abgearbeitet, das in diesem Fall die Zustandsvariable N initialisiert und außerdem das Ereignis A anstößt. Das Ereignis A erhöht einerseits jedesmal wenn es aufgerufen wird N um eins, andererseits ruft es sich selbst immer wieder mit der Zeitverzögerung t_a auf. Diese Simulation hat kein Abbruchkriterium, da immer ein Ereignis A in der FEL wartet und müsste daher händisch beendet werden. Eine ander Variante wäre das Einfügen einer Bedingung $N < 10$ (siehe Abbildung 7.3).

7.2 JSIM

JSIM ist ein JAVA-basiertes Simulations- und Animationsenvironment, das sowohl in einer prozessorientierten als auch einer ereignisorientierten-den Variante existiert und den *Query Driven Simulation* (QDS)-Ansatz unterstützt [70, 73]. JSIM besteht aus folgenden Packages:

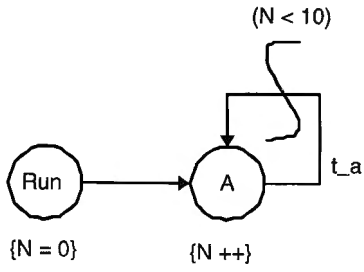


Abbildung 7.3: Arrivalprozess mit Abbruchsbedingung

- *Process Interaction Paradigm Package*: Dieser Teil der Library dient dem Entwurf prozessorientierter Simulationssysteme und wird im SimASP Framework verwendet.
  Simulationsmodelle können als Java Bean gekapselt werden und besitzen sogenannte *Dynamic Nodes*. Momentan stehen die Dynamic Nodes Server, Facility, Signal, Source und Sink zur Verfügung, um Modelle aufzubauen. Die Dynamic Nodes werden durch Kanten mit *SimEntities* verbunden. Jedes Simulationsmodell enthält außerdem ein *Model*-Element, das die Simulation kontrolliert, indem es alle Sources startet und die Simulation terminiert.

- *Event Scheduling Paradigm Package*: Alternativ dazu bietet JSIM auch die Möglichkeit, den ereignisorientierten Simulationsansatz zu verwenden. Es werden folgende Klassen implementiert:

  - *Event*: Darin können Eventroutinen, das heißt Abläufe, die bei dem Auftreten eines Ereignisses passieren, angegeben werden.

  - *Entitie*: Es können Eigenschaften der in der Simulation vorkommenden Entities spezifiziert und verwaltet werden.

  - *Scheduler*: Diese Klasse wird verwendet, um zukünftige Ereignisse in der FEL zu einzuplanen.

- *Support Components*: Die Supportkomponenten umfassen unter anderem folgende Packages:

  - *Queue-Package*: Dieses Package implementiert verschiedene Typen von Queues: FIFO-Queues, LIFO-Queues, Priority-Queues und temporäre Queues.

  - *Variate Package*: Es werden verschiedene diskrete und kontinuierliche Zufallszahlengeneratoren implementiert, die alle von der Klasse *Variate* abstammen und den JSIM-eigenen linearen Kongruenzgenerator verwenden. Alternativ dazu könenn auch externe Zufallszahlengeneratoren eingebunden werden.

  - *Statistc Package*: Es implementiert die Sammlung und Auswertung der statistischen Daten, die durch einen Simulationslauf generiert werden. Alle statistischen Auswertungen werden von der Klasse *Statistic* abgeleitet und können Minima, Maxima, Mittelwerte, Varianz, Standardabweichung, mittlere quadratische Abweichung und Konfidenzintervalle berechnen.

- *QDS Component*: Diese Komponente unterstützt das QDS-Environment. Der QDS-Ansatz besteht darin, dass sowohl der Simulationsexperte als auch der einfache Benutzer ein Simulationsmodell als ein Informationssystem verstehen, dass Information über das zu untersuchende System generiert und speichert [72, 73]. Dementsprechend enthält die QDS-Komponente Klassen zur Benutzerinteraktion und zur Datenbankanbindung.

- *Graphical Designer*: Das Graphical Designer Package enthält ein GUI, das es erlaubt, Simulationsmodelle mittels Drag and Drop zu

erstellen.

7.2.1 Prozessorientierte Simulation mit JSIM

Prozessorientierte Simulation implementieren typischerweise den Lebenszyklus eines Simulationsentities als Thread. Der Zeitfortschritt und die Kontrolle, welcher Thread ausgeführt wird, kann straight forward implementiert werden. Eine Uhr steuert den Fortschritt der Simulationszeit und eine FEL wird verwendet, um die Ordnung der zukünftigen Threadaktivitäten zu steuern. In einer JSIM-Simulation sind im Wesentlichen folgende drei Threads gleichzeitig aktiv:

- *Scheduler Thread*: Dieser virtuelle Thread ist ständig aktiv und steuert die Abfolge der Threadaktivitäten. Er durchläuft so lange eine Schleife, in der er die nächste Threadaktivität anstößt, bis die FEL leer ist.
  Da der Scheduler Thread mit niedrigster Priorität läuft, übernimmt immer der gerade aktive Simulationsthread die Kontrolle bis er beendet wird.

- *Simulation Thread*: Es kann immer nur ein Simulation Thread gleichzeitig aktiv sein und ein Simulation Thread kann entweder ein Simulation Entity, eine Entitie Source oder ein Signal sein.

- *Animation Thread*: Da JSIM von Beginn an auf den Einsatz von Animationen als Darstellung von Simulationsläufen gesetzt hat, sind Animationen und deren grafische Darstellung integraler Bestandteil des prozessorientierten Packages von JSIM. Der Animation Thread wird in einem fixen Abstand von ein paar Milisekunden aktiv, erfasst den Zustand der Simulation und stellt sie grafisch dar. Dieser Thread läuft unabhängig von den anderen beiden Threads und beeinflusst die Simulation nicht. Dadurch kann die Animation

einer Simulation bei Bedarf auch abgestellt werden, um zum Beispiel die Performance des Simulationslaufes zu erhöhen.
Da die Simulationsgeschwindigkeit normalerweise zu schnell für das menschliche Auge ist, muss bzw. kann die Animation mittels der Methode *Thread.sleep()* in der *main()*-Methode des virtuellen Schedulers auf die gewünschte Geschwindigkeit gebracht werden.

Auf der Basis der virtuellen Scheduler-Klasse und der Java *Thread*-Klasse sind die folgenden Klassen *SimObject* und *DynamicNode* implementiert. Abbildung 7.4 gibt eine Übersicht über die Klassenhierarchie des prozessorientierten JSIM-Packages:

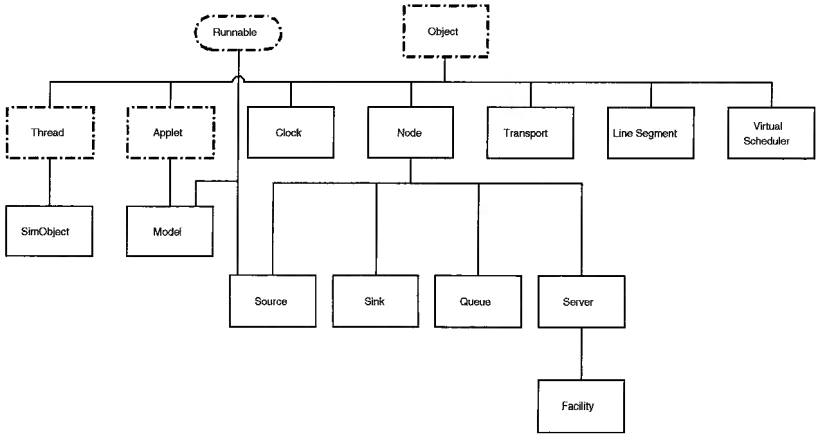


Abbildung 7.4: Die Klassenhierarchie des prozessorientierten JSIM-Packages

7.2.1.1 SimObjekt Klasse

Ein prozessorientiertes Simulationsmodell muss die Simulationsentities und ihren Lebenszyklus innerhalb des Simulationsmodells definieren. Eine Instanz von SimObject repräsentiert ein einzelnes Simulation Entity beziehungsweise einen Simulationsprozess.
SimObject erweitert die von Java bereitgestelte Threadklasse. Der Modellbilder wiederum kann die SimObject-Klasse erweitern, um Simulation Entities mit dem gewünschten Verhalten zu entwickeln.

7.2.1.2 DynamicNode Klasse

Dies ist eine abstrakte Klasse, die die gemeinsamen Eigenschaften aller Nodes in einem JSIM-Modell kapselt. JSIM stellt folgende Basis-Nodes zur Verfügung:

- *Server*: Ein Server stellt eine bestimmte Anzahl von Serviceeinheiten zur Verfügung, die einem Simulation Entity, das Serviceleistung anfordert, zur Verfügung stehen. Simulation Entities können als SimObjekts Server anfordern und verwenden. Wenn alle Server belegt sind, wird das Client Entity gedroppt.
  Server sammeln Statistiken über ihre Auslastung und die Servicezeiten der Clients

- *Facility*: Diese Klasse ist von der Server-Klasse abgeleitet und repräsentiert einen Server mit vorgeschalteter Queue. Das bewirkt, dass im Falle eines ausgelasteten Servers ein Client Entity nicht gedroppt wird, sondern in die Warteschlange des Servers eingereiht wird.

- *Signal*: Ein Signal beeinflusst das Verhalten eines Severs, indem es die Anzahl der Serviceeinheiten entweder vermindert oder erhöht.

- *Source*: Eine Source ist ein Generator von Simulation Entities. Es werden SimObjects nach dem Ablauf einer Zwischenankunftszeit erzeugt. Die Anzahl der gleichzeitig erzeugten SimObjects kann angegeben werden.
  Da jede Source nur Instanzen eines bestimmten SimObjects erstellen kann, muss für jeden in der Simulation verwedeten Entity-Typ eine eigene Source erzeugt werden.

- *Sink*: Eine Sink zerstört die von einer Source erzeugten Entities. SimObjects werden am Ende ihres Lebenszykluses zerstört.

7.2.1.3 Transport Klasse

Die Transportklasse repräsentiert die Kanten eines Simulationsgraphen und verbinden jeweils zwei Knoten. SimObjects wandern mit konstanter Geschwindigkeit entlang solcher Kanten (= Transport eines Entities von einem Knoten zum nächsten). Die Geschwindigkeit ist ein änderbarer Parameter eines Transports.
Transports sind als *quad curves* realisiert, die Bestandteil der Java-2D API sind und Kurven als quadratische Funktion von x und y Koordinaten spezifiziert.

7.2.1.4 Model Klasse

Diese Klasse ist so konzipiert, dass mehrere Simulationsmodelle simultan in je einem Windowframe ablaufen können.
Die Model Klasse startet alle Sources und ist außerdem für das periodische Neuzeichnen der Simulationsanimation zuständig. Nach Ende der Simulation werden statistische Resultate ausgegeben.

7.2.1.5 Animation von JSIM Modellen

In JSIM kann ein Simulationsmodell als ein gerichteter Graph aufgefasst werden, wo ausgehend von Sources Knoten durch gerichtete Kanten verbunden werden. Da in JAVA reichhaltige Bibliotheken und APIs zur Darstellung von 2D und 3D Animationen zur Verfügung stehen, funktioniert die Erweiterung eines JSIM-Simulationsmodells um eine Animation nahtlos.
Jeder Knoten und jede Kante hat fixe Koordinaten, die entweder händisch eingegeben werden oder mit dem JMODEL Visual Designer mittels Drag and Drop definiert werden.
Die Entities dagegen fließen durch das Sytem entlang der Graphen und haben dadurch ständig wechselnde Koordinaten. Die Model Klasse verursacht periodisch ein Neuzeichnen aller Nodes und aller SimObjects. Die gleitende Bewegung veränderlicher Objekte wird durch eine entsprechende Anzahl an von der Model Klasse verursachten Updates erreicht. Außerdem wird *Double Buffering* eingesetzt um das Rucken einer Animation zu reduzieren [73].

Kapitel 8

Die Schnittstelle Modell-Simulator

Die Schnittstelle Modell-Simulator besteht aus vier logischen Modulen:

- *SimASP-Kern*: Die XML-Schnittstelle und Ablauflogik der Durchführung eines Simulationslaufes.
- *Normalisierte Darstellung eines Simulationsmodells*
- *Einheitliche Schnittstelle zur Simulator-Schicht*
- *Simulator-Schicht*

Mit diesem Aufbau ist es möglich, beliebige Simulatoren zu nutzen (es müssen lediglich die Mapping-Layer dafür entwickelt werden). Der zu benutzende Simulator wird in den Eingabedaten definiert, die ersten beiden Module müssen nicht verändert werden.

Beim Durchführen einer Simulation wird folgende Ablauflogik umgesetzt:

- Der SimASP-Kern wird aufgerufen, die Eingabe-, Ausgabe- und (optional) Statusdateinamen werden angegeben.
- Die Eingabedatei wird geöffnet und die enthaltenen XML-Daten werden eingelesen.
- Aus den XML-Daten wird das normalisierte (d.h. Simulator-unabhängige) Simulationsmodell erstellt.
- Gemäß den Header-Daten der XML-Datei wird der Simulator ausgewählt und initialisiert.
- Das normalisierte Simulationsmodell wird in die simulatorspezifische Form übersetzt.
- Der Simulationslauf wird durchgeführt, dabei werden statistische Daten protokolliert.
- Nach Abschluss des Simulationslaufes werden die statistischen Daten ausgewertet und in die Ausgabedatei geschrieben.

8.1 Die normalisierte (kanonische) Darstellung eines Modells

Der SimASP-Kern erzeugt aus den Eingabedaten eine normalisierte (auch als kanonische), vom Simulator unabhängige Darstellung des Simulationsmodells. Dieses Modell entspricht dem in der XML-Eingabedatei angegebenem und ist (entsprechend dem üblichen Vorgehen bei objektorientierter Softwareentwicklung) aus Objekten aufgebaut, die die logischen (statischen) Entitäten im Modell repräsentieren (Source, Queue, Server, Fork, Join, Sink).
Durch diese Darstellung kann eine Kapselung der Modelllogik von simulatorspezifischen Details umgesetzt werden.

Die Implementierung dieses kanonischen Modells folgt der Struktur der XML-Datei, wobei jedes relevante Element (sofern es keine Eigenschaft des darüberliegenden Elements beschreibt, sondern einen funktionalen Inhalt darstellt) in ein entsprechendes Objekt übersetzt wird.

8.1.1 Das Simulations-Objekt und Metadaten

Die oberste Hierarchieebene bildet das Simulation-Objekt. Es entspricht dem <Simulation>-Element und hält die Metadaten des Simulationslaufes und -modells (im <head>-Element bzw. SimulationHead-Objekt) und das eigentliche Modell (im <model>-Element bzw. SimulationModel-Objekt).
Das SimulationHead-Objekt ist denkbar einfach aufgebaut, es dient lediglich als Datenstruktur (mit den Modell-Metadaten als Objektdaten).
Auch das SimulationModel-Objekt dient als Datenstruktur, wobei die (statischen) Entitäten des Modells und die Objekte (= Connectors) in Form eines Vectors (dynamisches Array) gehalten werden.
Diese Objekte enthalten keine Logik (die über das Setzen und Abrufen der Objektdaten hinausgeht). Sie dienen also nur als Datenstrukturen zum Halten und Übergeben des Modells an andere Schichten.

8.1.2 Die Entities des Modells

Die Klasse model.canonical.SimEntity repräsentiert den entityType im XML-Eingabeschema. Sie erweitert model.canonical.SimObject und übernimmt damit das ID-Attribut und erweitert es um eine beliebige Anzahl an Locations ((x,y)-Wertepaare).
Die Funktionalität der Entities wird erst durch die Subklassen festgelegt. Diese sind als Source, Sink, Queue, Server, Fork und Join implementiert (in model.canonical.SimSource, etc.).

8.1.3 Zufallszahlenverteilungen

Zufallszahlenverteilungen spielen eine wesentliche Rolle in der Simulation, und um die Auswahl der Verteilungen nicht auf der normalisierten Ebene einzuschränken, werden Verteilungen in sehr allgemeiner Form definiert. Die Klasse model.canonical.RandomVariateStream repräsentiert eine beliebige Verteilung, deren Typ durch einen (beliebigen) String, in der Regel der Name der Verteilung (z.B. "Exponential" oder "Uniform"), festgelegt wird. Außerdem können mehrere (momentan max. 4) numerische Parameter angegeben werden. Der simulator-spezifische Mapping-Layer ist dann für die entsprechende Umsetzung der Verteilung verantwortlich.

8.2 Der Mapping Layer für den Simulator Simkit

Kernstücke von Simkit sind die Eventlist und der Scheduler, die das Management der Events übernehmen. Ein Modell wird aus Entities aufgebaut, die von einer Klasse simkit.SimEntityBase abgeleitet sind; dadurch werden sie vom Scheduler erkannt und in die Simulation eingebunden. Im Detail sieht der Ablauf einer Simulation folgendermaßen aus:

- Erstellen der Simulationsobjekte.
- Konfigurieren des Schedulers mit Simulationsparametern.
- Starten der Simulation durch Aufruf von Schedule.startSimulation().
- Aufrufen aller doRun()-Methoden der Simulationsobjekte, zur Initialisierung und zum schedulen weiterer Events.

- Solange sich Events in der Eventlist befinden und kein Abbruchkriterium erfüllt ist (z.B. max. Simulationszeit oder Aufruf von Schedule.stopSimulation()), ist der Scheduler aktiv.

- Nach Simulationsende kann ein Statistikbericht erstellt werden, die aufgenommenen Werte werden von den Simulationsobjekten während der Simulation selbst gesendet (über die JAVA-übliche PropertyChangeListener Systematik).

8.2.1 Modellierung in Simkit

Um einen Mapping-Layer zwischen SimKit und dem generischen Modell zu entwickeln, muss analysiert werden, wie ein allgemeines Modell in SimKit entworfen werden kann. In dieser Fallstudie wird die Modellierung im Paket model.simkit realisiert.
Um vom prozess-basierten zum ereignis-basierten Paradigma überzugehen, werden die Grundkomponenten der Prozessmodellierung, also Source, Sink, Queue, Server, usw. als SimKit-Objekte realisiert. Intern wird deren Logik ereignisbasiert verarbeitet, die Schnittstellen zu anderen Objekten werden von SimKit unabhängig mit einem JAVA-Interface, genannt model.simkit.SimKitChainEntity, realisiert. Dieses Interface muss nachfolgende Entitäten anstoßen können, was als Weitergabe eines Items (ohne weitere Eigenschaften) angesehen werden kann. Der Nachfolger muss das Item zurückweisen können (wenn z.B. eine Queue voll ist), aber auch bei seinem Vorgänger anfragen, ob noch Items folgen (nicht unbedingt sofort) bzw. den Vorgänger informieren, dass nun Items akzeptiert werden können.
Zum Setzen von Ereignissen in einem Simulationsobjekt, bei deren Eintritt eine bestimmte Funktion aufgerufen wird, wird von SimKit die Funktion simkit.SimEntityBase.waitDelay(name, delayTime) angeboten. "name" ist dabei ein String, der den Namen der aufzurufenden

Funktion ohne einleitendes "do" enthält (z.B. "Arrival", um die Funktion doArrival() aufzurufen). "delayTime" ist die Zeitspanne, gemessen in Simulationszeit, bis das Ereignis ausgelöst wird. Die angegebene Funktion wird dann über die Java Reflection API aufgerufen.

Das Source-Objekt model.simkit.SimKitSource implementiert den Standardfall des Ankunftsprozesses. Die Methode "doArrival" wird immer aufgerufen, wenn ein Item ankommt und gibt das Item dem Nachfolger der Source weiter, wenn möglich. Ist dies nicht möglich, so wird das Item in dieser Implementierung verworfen, jedoch eine Warnung erzeugt. Auch in der Statistik ist dies durch eine Diskrepanz zwischen Anzahl der Arrivals und Anzahl der hinausgehenden Items ersichtlich. Bei der Source ist kein Vorgänger definiert, aber ein Nachfolger kann bei der Source anfragen, ob noch Items kommen werden.

Das Sink-Objekt (model.simkit.SimKitSink) ist noch einfacher aufgebaut, es wird nur die Interface-Methode inItemArrival() implementiert und bei Ankunft eines Items wird der Statistik-Counter aktualisiert.

Die Funktionalität von Queue und Server können in eleganter Weise in einem einzigen Objekt zusammengefasst werden, dem model.simkit.SimKitGenericProcessor. Als Kapazität wird im Falle einer Queue dabei die maximale Länge der Queue, im Falle des Servers die Anzahl gleichzeitig bearbeitbarer Items gesehen. Oft ist bei einer Queue keine Verzögerung der Items vorgesehen, was durch Vorgabe einer Nullzeitverteilung zu erreichen ist. Kommt ein Item in einem Prozessor an, und ist Kapazität dafür frei, wird das Ereignis "StartService" ausgelöst. In der aufgerufenen Methode "doStartService" wird die Bearbeitungszeit für dieses Item festgelegt und dann ein "EndService"-Ereignis eingeplant. Sobald die Funktion aufgerufen wird, versucht

sie, das Item an den Nachfolger weiterzugeben. Ist dies möglich, wird natürlich wieder Kapazität frei, anderenfalls wird das Item gestapelt ("gestacked") und auf Abruf dem Nachfolger weitergegeben.
Fragt der Nachfolger an, ob noch Items kommen, kann entweder sofort geantwortet werden (wenn gerade Items in Bearbeitung oder gestapelt sind), oder die Anfrage wird an den Vorgänger weitergegeben. Hier ist die Anfrage-Methode successorNotify() mit doppelter Funktion behaftet: einerseits gibt sie die Information, ob noch Items kommen, zurück, andererseits wird die Methode auch zur Benachrichtigung, dass nun Kapazität beim Nachfolger frei geworden ist, benutzt. Wenn die Funktion aufgerufen wird und gestapelte Items existieren, wird daher versucht, ein gestapeltes Item weiterzugeben.

Die Objekte Fork (model.simkit.SimKitFork) und Join (model.simkit.SimKitJoin) können beliebig viele Nachfolger (Fork) bzw. Vorgänger (Join) haben. Die andere Seite ist jeweils einfach belegt. Um einen Item-Fluss aufzuteilen (Fork), muss die Strategie, nach der diese Aufteilung stattfindet, festgelegt werden. Derzeit sind die 2 Strategien "Broadcast" und "NextFree" implementiert. Im Broadcast-Modus werden die Items vervielfältigt und der Reihe nach an alle Nachfolger geschickt. Sobald mindestens ein Nachfolger das Item angenommen hat, gilt das Item als angenommen, ansonsten wird es zurückgewiesen. Die Strategie "NextFree" jedoch behält die Anzahl der Items bei, es wird genau ein Nachfolger ausgewählt (zufällig, gepuffert). Ist dieser Nachfolger belegt, werden in einer zufälligen Reihenfolge alle anderen Nachfolger probiert, bis das Item entweder weitergeht, oder alle Nachfolger das Item zurückgewiesen haben-dann weist auch die Fork das Item zurück.

Umgekehrt bei der Vereinigung (Join), hier ist keine "Join-Strategie"

nötig, da nur ein Ausgang existiert und die Items keine Eigenschaften (z.B. zur Priorisierung) haben.

Wichtig ist der sorgfältige Entwurf der Interface-Methoden, da durch die Aufteilung der Ströme (Kanten) Kreise im Graphen entstehen können. Dies lässt sich nicht verhindern, die Benachrichtigungen müssen aber robust implementiert werden, um Endlosschleifen zu verhindern. Die Überprüfung wurde in die Benachrichtigung der Join-Entität vom Nachfolger eingebaut, der Itemstatus wird nur dann von den Vorgängern geholt, wenn die Methode zu diesem Simulationszeitpunkt noch nicht aufgerufen wurde. Ansonsten wird das zuletzt erhaltene Ergebnis (noch Itemankunft oder nicht) zurückgegeben. Damit kann keine Endlosschleife entstehen, ein Kreis wird höchstens ein Mal durchlaufen. Es zeigt sich auch, dass diese Maßnahme hinreichend ist, denn das beschriebene Verhalten kann nur auftreten, wenn überhaupt eine Join eingebaut ist. Alternativ könnte man eine analoge Maßnahme stattdessen auch in der Fork realisieren.

Weiters findet sich im Paket model.simkit die Klasse SimKitSimulationRun, die als globaler Item-Zähler fungiert und ein Abbruchkriterium auf Basis der insgesamt eingegangenen Items realisiert. Dabei wird von jeder Source angefragt, ob eine Itemankunft möglich bzw. erlaubt ist (dies ist der Fall, wenn keine Obergrenze der Itemanzahl definiert ist, oder wenn diese noch nicht erreicht ist).Bei erfolgter Itemankunft in einer Source bzw. Sink werden mit den Methoden increaseItemIn() bzw. increaseItemOut() die Itemzähler aktualisiert.

8.2.2 Der Mapping-Layer: Die Klasse SIMKITInterface

Die Aufgabe des SIMKITInterface (des eigentlichen Mapping-Layers zwischen dem generischen Modell und SimKit) ist es nun, aus einem gegebenen generischen Modell eine SimKit-Objektstruktur aufzubauen, die Simulation initiieren / starten und stoppen zu können und Statistikdaten in Form eines XML-Berichtes zurückgeben zu können.
Die Benennung des Interfaces als model.SIMKITInterface ist notwendig, damit die Klasse in dieser Schreibweise von SimASP zur Laufzeit gefunden werden kann. Die Klasse muss dazu auch das Interface model.SimInterface implmentieren, über das die gesamte Kommunikation zu SimASP läuft. So ist es möglich, ohne Änderung von SimASP neue Mapping-Layer in das System zu integrieren. Die Modell-Metadaten enthalten den Simulatornamen, wie etwa "simkit", der dann zur Instanzierung (wieder mit Java Reflection) des entsprechenden Mapping Layers herangezogen wird.
Als Datenstruktur für das Modell wird eine HashMap gewählt, mit der Integer-ID der Entität als Schlüssel und dem entsprechenden SimKit-Objekt als gespeichertem Wert. Damit ist ein schneller, ID-basierter Zugriff möglich, die Traversierung des Modells ist ohnehin durch die verkettete Struktur der Objekte gesichert.
Kern des Mapping Layers ist die Methode generateModel(sim) mit "sim" als generisches Simulationsobjekt. Sie erstellt und konfiguriert die SimKit-Objekte, die allen Entitäten im generischen Modell entsprechen (spezifisch für jeden Objekttyp) und erstellt alle Verkettungen aus der Liste der Konnektoren des generischen Modells. Modellspezifische Zufallsverteilungen werden dabei über die Methode createFromRandomVariateStream(rndVarStream) erzeugt. In SimKit gibt es zahlreiche "Variate"-Klassen (Verteilungen), die Instanzierung wird aus

praktischen Gründen wieder per Java Reflection erledigt.
Der Mapping-Layer implementiert dann die verbleibenden Interface-Methoden initSimulation(), startSimulation(), stopSimulation() und generateXMLStatistics(doc). Sie sind trivial aufgebaut, die XML-Statistik wird generisch durch erzeugen der XML-Statistikelemente für jedes SimKit-Objekt gebildet. Schließlich folgen die Methoden getSimTime(), getNumberItemsIn() und getNumberItemsOut().

8.3 Mapping Layer für den Simulator JSIM

JSIM ist als eigenständiges Produkt konzipiert und stark auf ein grafisches Interface zum Anwender aufgebaut. Dies erweist sich für reine Simulationszwecke als unnötig und behindernd, da dieser Ballast nie ganz abgeworfen werden kann. Andererseits sind durch die Prozessorientierung alle benötigten Klassen für die Entities bereits vorhanden und können instanziert oder erweitert werden . Dazu sind im Paket JSIM.process die Klassen Source, Sink, Server / Facility, Split und And-Join hilfreich (alle erweitern JSIM.process.DynamicNode). Items werden nicht als Messages, sondern als physikalische Objekte realisiert. Vorteil dabei ist die Möglichkeit, den Items Eigenschaften zuzuweisen (die in JSIM aber nicht unterstützt wird), dagegen spricht der Speicher- und Rechenzeitbedarf bei hoher Itemanzahl.
Um ein Modell zu repräsentieren, steht JSIM.process.Model zur Verfügung, das aber als Unterklasse von java.awt.Frame den genannten grafischen Overhead mitschleppt. Der Ablauf einer Simulation sieht dann in JSIM so aus:

- Erstellen eines eigenen Model-Objektes, abgeleitet von JSIM.process.Model.

- Erstellen der Simulationsobjekte, hinzufügen zum Model-Objekt.
- Initialisierung des Modells mit Simulationsdaten.
- Starten der Simulation durch Aufruf von model.beginSim() und model.start().
- Die Simulation ist eigentlich ein Animationsprozess, der bei dem Ausführen einer Simulation "ohne Animation" nicht dargestellt wird. Wenn model.isDone() wahr wird, ist die Simulation abgeschlossen.
- Nach Simulationsende kann ein Statistikbericht erstellt werden, die Werte werden von den Simulationsobjekten selbst verwaltet, deren Bedeutung und Benennungen sind aber nicht zuletzt aufgrund fehlender Dokumentation unklar.

8.3.1 Modellierung in JSIM

Da die statischen Entitäten schon in JSIM implementiert sind, enthält das Mapping-Paket lediglich eine speziell adaptierte
Modell-Implementierung und die Definition der dynamischen (durch das System bewegten) Objekte.
In model.JSIM.ModelImpl befindet sich diese Modellimplementierung, sie erweitert die JSIM-Klasse JSIM.process.Model um die Logik zum Halten der SimASP-Struktur. Eine Kette von statischen Entitäten und Konnektoren (Transports) wird gespeichert, zur Zeit sind nur einfache Modellstrukturen mit einem Item-Fluss unterstützt.
Die dynamischen Items werden von einer Source erstellt und mit einer Referenz auf die Modellimplementierung initialisiert, leider steht aber kein anderer Weg, Daten zu übermitteln, zur Verfügung. Die Item-Bewegung wird vom Item selbst verwaltet, es holt sich vom Modell das erste und dann immer die nächsten Entitäten und arbeitet sie

selbstständig ab.
Durch die mangelnde Kontrolle über die Item-Initialisierung ist es ohne Adaption des JSIM-Sourcecodes nicht möglich, Forks, Joins und multiple Itempfade sinnvoll zu realisieren.
Schließlich befindet sich in diesem Paket noch die Klasse JSIMStatisticTools, die aus den JSIM-Statistikobjekten die zum SimASP Schema konformen XML-Daten erzeugt. Leider wäre auch hier aufgrund mangelnder Dokumentation und fragwürdiger Implementierung seitens JSIM sehr viel Entwicklungs- und Testaufwand nötig, derzeit sind die von JSIM hier exportierten Daten unter weitgehend unbekannten Randbedingungen zu sehen.

8.3.2 Der Mapping Layer: Die Klasse JSIMInterface

Der Mapping-Layer folgt im Aufbau (gefordert auch vom Aufbau des generischen Interfaces) dem bei SimKit. Die Methode generateModel() baut auch hier das JSIM-spezifische Modell auf. Die Realisierung ist allerdings weniger klar strukturiert, da den JSIM-Objekten schon zum Zeitpunkt der Erstellung z.B. alle Daten der ausgehenden Kanten vorliegen müssen.
Die anderen Methoden sind einfach gehalten, wie auch die Entsprechungen im SimKit Interface.

8.4 Beurteilung der verwendeten Simulatoren

SimKit erwies sich in der Integration als sauber entwickeltes System, mit klar ersichtlicher Struktur und Funktion. Die Dokumentation ist zwar spärlich, aber Dank JavaDoc-API ausreichend. Insgesamt funktionieren die beiliegenden Beispiele alle gut, ohne übermäßigen Modellierungsauf-

wand. In der derzeitigen Version des Mapping Layers und von SimASP ist es allerdings nicht möglich, Items dezidiert als wandernde Objekte mit frei definierbaren Eigenschaften zu realisieren, dazu müssten sowohl der XML-Input, das generische Modell und der Mapping Layer adaptiert und erweitert werden.
Im direkten Vergleich mit einem sauber entwickeltem Produkt wie SimKit ist JSIM in vielen Aspekten mangelhaft, zusätzlich dazu kommt, dass JSIM grundsätzlich für einen anderen Einsatzzweck entwickelt wurde (End-User-Simulation).
Mit dem JSIM-Interface demonstriert das SimASP Framework aber, dass trotz den angesprochenen Komplikationen der SimASP-Ansatz sehr flexibel ist. Es zeigt sich aber auch, dass für eine generische Lösung wie SimASP die Qualität der benutzten Komponenten wesentlich ist.

Kapitel 9

Die Entwicklungsumgebung

Da bei der Entwicklung des SimASP Frameworks Wert darauf gelegt wurde, offene Standards und Open Source Programme zu verwenden, wurden folgende integrierten Entwicklungsumgebungen (IDEs von *Integrated Development Environmets*) verwendet.

9.1 Eclipse

Die Eclipse-IDE ist ein Framework, das seit Version 3.0 nur aus einem Kern besteht, der einzelne Plugins lädt, die dann die eigentliche Funktionalität zur Verfügung stellen. Dieser Ansatz nennt sich Rich Client Platform (kurz RCP) und basiert auf dem OSGi-Standard [76]. Diese Spezifikation definiert eine Java-basierte Laufzeitumgebung samt Basisdiensten.
Sowohl Eclipse als auch die Plugins sind vollständig in Java implementiert und als GUI-Framework zur Erstellung der grafischen Oberfläche

wurde das *Standard Widget Toolkit* (SWT) verwendet. Zur Darstellung der GUI-Komponenten basiert SWT auf den nativen GUI-Komponenten des jeweiligen Betriebssystems [26].

9.2 Jakarta Tomcat

Jakarta Tomcat ist ein Servletcontainer, der im Jakarta-Projekt entwickelt wurde. Als Servlet bezeichnet man im Rahmen der Java 2 Platform Enterprise Edition (J2EE) ein Java-Objekt, an das ein Webserver Anfragen seiner Clients delegiert, um die Antwort an den Client zu erzeugen. Der Inhalt der Antwort ist nicht statisch (etwa in Form einer HTML-Seite) am Webserver hinterlegt, sondern wird erst im Moment der Anfrage generiert.
Das Jakarta-Projekt ist ein Projekt der Apache Software Foundation. Es beherbergt, entwickelt und unterstützt freie Software, die in der Programmiersprache Java geschrieben wurde. Zur Zeit besteht es aus 19 Unterprojekten.
Tomcat ist in Java geschrieben und läuft somit auf jedem Betriebssystem, für welches es eine JVM gibt. Tomcat läuft in der Regel als Plugin im Kontext eines Web Servers, um die Unterstützung für Servlets herzustellen und hat, vor allem für Entwicklungszwecke, einen eigenen HTTP-Server, läuft aber meistens zusammen mit dem Apache Web-Server [98].

9.3 Apache Axis

Apache Axis (*Apache eXtensible Interaction System*) ist eine SOAP Engine zur Konstruktion von darauf basierenden Webservices und Client-Anwendungen. Es existiert eine Implementation in C++ und Java.
Apache Axis ist eine Neuentwicklung und der Nachfolger von Apache SOAP, das auf dem IBM-Framework SOAP4J basierte. Ziel dieser Neu-

entwicklung war eine höhere Geschwindigkeit, Flexibilität, Komponentenorientierung und Abstraktion des Transportframeworks, sowie die Unterstützung von WSDL.
Die höhere Geschwindigkeit erreicht AXIS durch Verwendung des SAX-Parsers im Gegensatz zum langsameren DOM-Parser von Apache SOAP. Axis wird häufig als Java-Servlet innerhalb eines Servlet-Containers (beispielsweise Jakarta Tomcat) betrieben, der Webservices für Java-Klassen anbietet [7].

9.4 XMLSpy

XMLSpy ist eine umfassende IDE der Firma Altova zur Entwicklung von XML-Projekten und steht in drei Versionen zur Verfügung [124]. Die Home Edition ist als freie Software (allerdings mit eingeschränktem Funktionsumfang) verfügbar und wurde für die Entwicklung der XML-Schemata und XML-Dateien verwendet.
Zur Bearbeitung des XML-Teils des SimASP Frameworks wurden vor allem folgende Funktionen verwendet:

- *Schema/WSDL Editoren*: Neben der Bearbeitung des Textes kann für XML-Schemata auch eine eigene Schema/WSDL-Ansicht verwendet werden - eine grafische Benutzeroberfläche, die das Erstellen von komplexen Schemata erleichtert.

- *Wohlgeformtheitsprüfung und integrierter Validator* : Alle XML-Dokumente werden auf ihre Wohlgeformtheit überprüft, wenn die Ansicht gewechselt wird oder die Datei gespeichert wird. Außerdem können XML-Dokumente validiert werden, wenn ein Schema (DTD oder XML-Schema) mit dem XML-Dokument verknüpft wurde. Andere Dokumentarten wie z.B. DTDs werden außerdem auch auf Fehler in der Syntax und Struktur überprüft.

Teil IV

Validierung des Frameworks

Kapitel 10

Beispielmodelle der DES

In diesem Kapitel werden exemplarisch drei Beispiele verschiedener Komplexität präsentiert, die als Testfälle für das SimASP Framework dienen.
Sie werden jeweils in einer resourcenorientierten Darstellung und in einer Ereignisgraphendarstellung formuliert und die wichtigsten Modellparameter werden erklärt. Die XML-Modellfiles sind im Anhang einzusehen.

10.1 Job Shop Modell

Das erste Beispiel verwendet die Basiskomponenten eines resourcenorientierten Modells und besteht aus einer einfachen Kette *Source-Queue-Server-Sink* (siehe Abbildung 10.1).

Items werden in einer Source mit exponentialverteilter Zwischenankunftszeit erzeugt, in einer Queue zu einem Server geführt, in diesem Server abgearbeitet und dann in einer Sink vernichtet.
Die Abbildung 10.2 zeigt die entsprechende Ereignisgraphendarstellung.

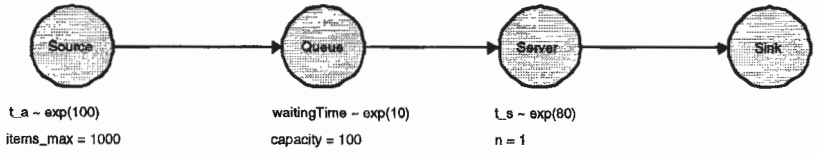


Abbildung 10.1: Job Shop Modell

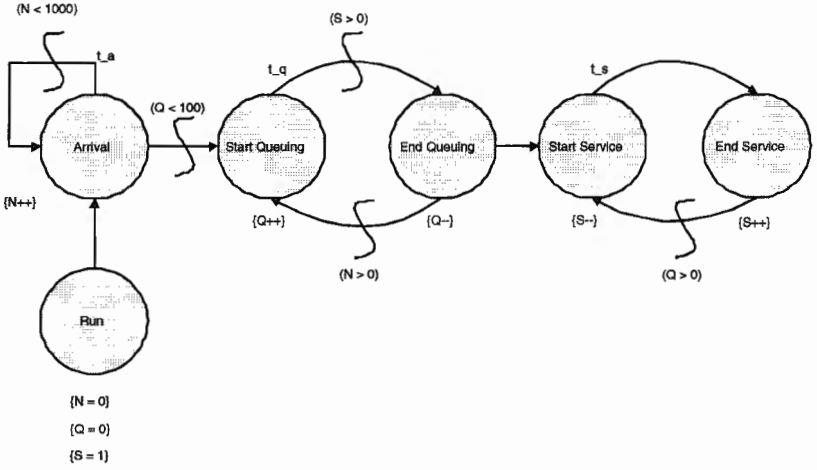


Abbildung 10.2: Ereignisgraph des Job Shop Modells

10.1.1 Die Parameter des Job Shop Modells

10.1.1.0.1 Die experimentsteuernden Modellparameter

- *Replications*: 3
- *Initial Seed*: 1
- *Stop Kriterium*: MaxItems = 1000
- *Simulator*: JSIM

10.1.1.0.2 Source

- *interArrivalTime*: Exponentialverteilt mit Parameter 100

10.1.1.0.3 Queue

- *WaitingTime*: Exponentialverteilt mit Parameter 10
- *Kapazität*: 100

10.1.1.0.4 Server

- *ServiceTime*: Exponentialverteilt mit Parameter 80
- *Anzahl gleichzeitiger Serviceprozesse*: 1

10.1.1.0.5 Sink: Die Sink dient dem Zweck, die im System erzeugten Items einzusammeln und zur statistischen Auswertung Items zu zählen.

10.2 Multiple Server Lines

Das zweite Beispiel erweitert das Job Shop Modell um die Ressourcen *Fork* und *Join*, mit denen Verzweigungen der Itemströme realisiert

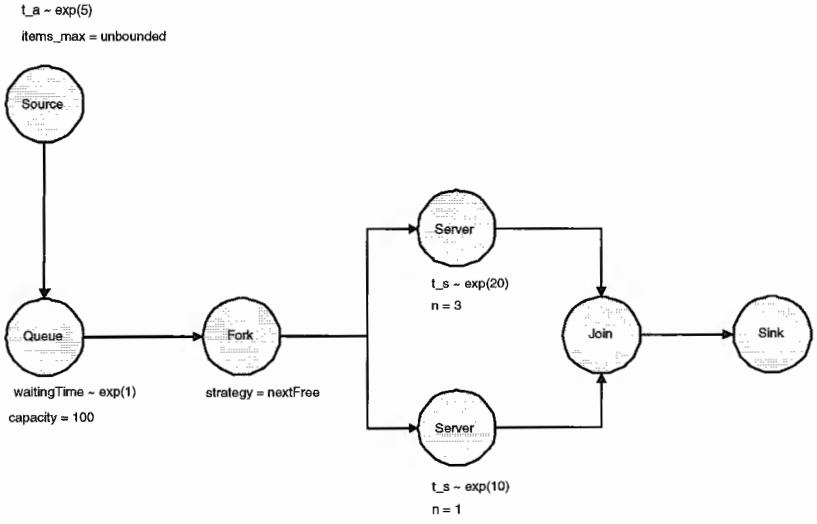


Abbildung 10.3: Das Modell einer Multiserver Line

werden können (siehe Abbildung 10.3).

Die Items werden in einer Source mit exponentialverteilter Zwischenankunftszeit erzeugt, in einer Queue zu einer Fork geführt, die aus den verfügbaren ausgehenden Connections zufällig die nächste freie Connection ermittelt. Ein Item wird dann zu einem der beiden Server geroutet. Die Server sind verschieden parametrisiert: Ein Server mit langer Bearbeitungszeit und mehreren gleichzeitig arbeitenden Serviceprozessen und ein schnellerer Server mit einer Serviceline. So ergeben sich verschiedene Durchsatzraten. Nach der Servicebearbeitung werden die Itemströme wieder zusammengeführt und in einer Sink gesammelt.
Die Abbildung 10.4 zeigt die entsprechende Ereignisgraphendarstellung.

10.2.1 Die Parameter des Multi Server Lines-Modells

10.2.1.0.6 Die experimentsteuernden Modellparameter

- *Replications*: 2
- *Initial Seed*: 1
- *Stop Kriterium*: MaxTime = 10000
- *Simulator*: Simkit

10.2.1.0.7 Source

- *interArrivalTime*: Exponentialverteilt mit Parameter 5

10.2.1.0.8 Queue

- *WaitingTime*: Exponentialverteilt mit Parameter 1
- *Kapazität*: 100

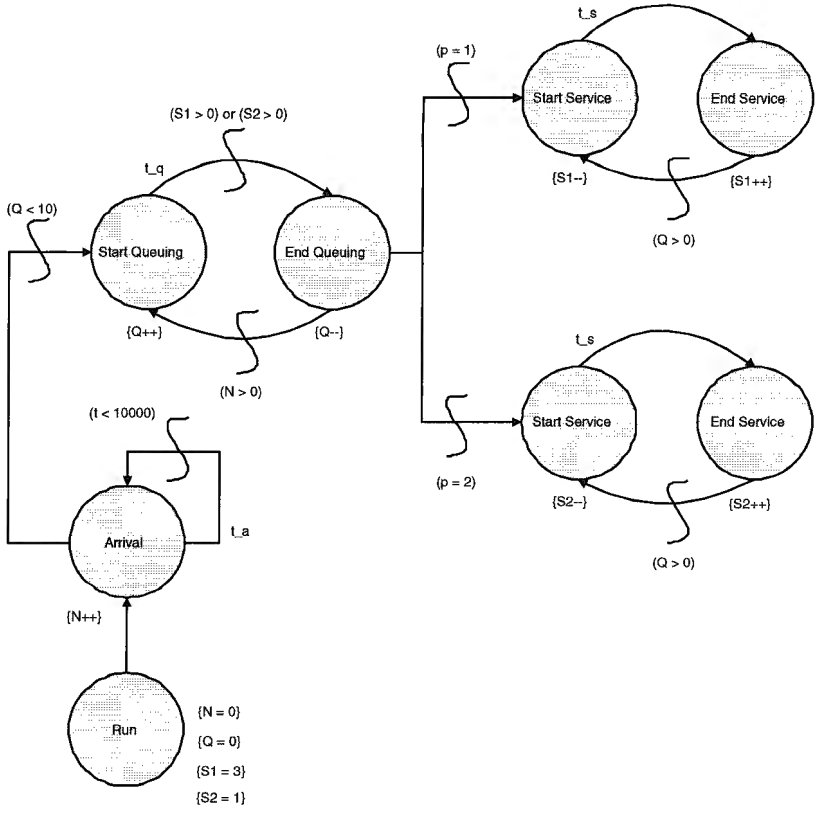


Abbildung 10.4: Ereignisgraph des Multi Server Line-Modells

10.2.1.0.9 Fork

- *Verzweigungsstrategie*: NextFree

10.2.1.0.10 Server1

- *ServiceTime*: Exponentialverteilt mit Parameter 20
- *Anzahl gleichzeitiger Serviceprozesse*: 3

10.2.1.0.11 Server2

- *ServiceTime*: Exponentialverteilt mit Parameter 10
- *Anzahl gleichzeitiger Serviceprozesse*: 1

10.2.1.0.12 Join: Ein Join-Element dient dem Zusammenführen mehrerer Itemströme und hat keine innere Logik implementiert.

10.2.1.0.13 Sink: Die Sink dient dem Zweck, die im System erzeugten Items einzusammeln und zur statistischen Auswertung Items zu zählen.

10.3 Multiflow Modell mit Feedback

Mit dem dritten Modell, das alle verfügbaren Typen von Ressourcen und rückgekoppelte Itemflows verwendet, wird die Stabilität des Frameworks auch bei komplexen Modellen getestet (siehe Abbildung 10.5).
Zwei Sources erzeugen mit verschiedenen Zwischenankunftsverteilungen Items. Nachdem die Items der zweiten Source durch einen Serviceprozess gegangen sind, werden sie mit den (noch unverarbeiteten) Items der andern Source zusammengeführt. In der Fork werden die Items an den zufällig nächsten freien Connector übergeben. Das kann dazu führen,

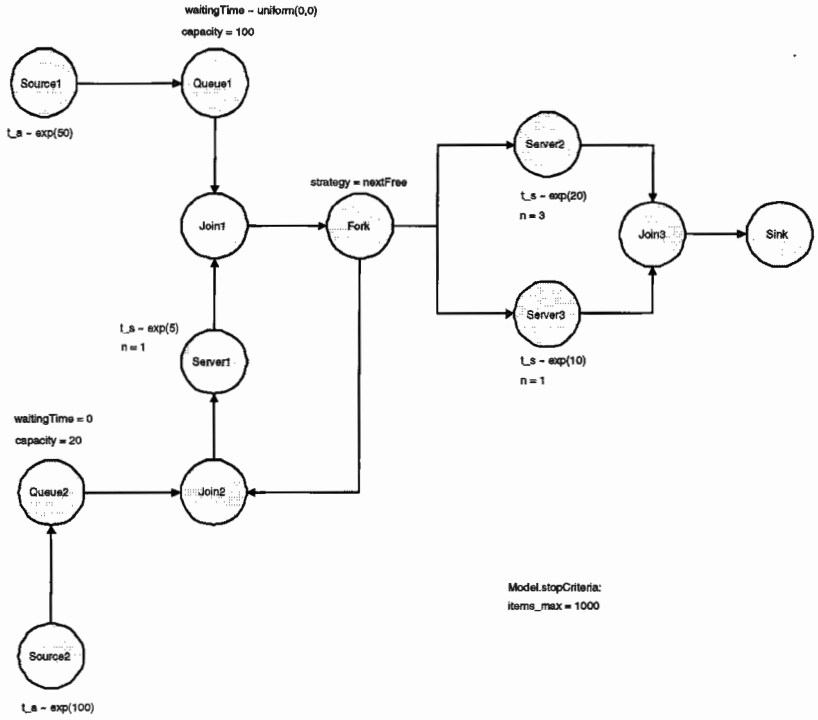


Abbildung 10.5: Das Modell des Multiflow Modells mit Feedbackschleife

dass die in der zweiten Source erzeugten Items wieder zum ersten Server geroutet werden. Nachdem Items im zweiten oder dritten Server bearbeitet wurden, fließen sie in eine Sink und werden dort gesammelt. Als Abbruchkriterium *StopCriteria* ist *maxItems = 1000* angegeben. Das bedeutet, dass die Sources nach dem 1000sten erzeugten Item keine neuen Items in das System einbringen. Dadurch steigt mit Fortschritt der Simulation die Wahrscheinlichkeit, dass alle Items in der Sink landen.
Die Abbildung 10.6 zeigt die entsprechende Ereignisgraphendarstellung.

10.3.1 Die Parameter des Multiflow-Modells

10.3.1.0.14 Die experimentsteuernden Modellparameter

- *Replications*: 1
- *Initial Seed*: 1
- *Stop Kriterium*: MaxItems = 1000
- *Simulator*: Simkit

10.3.1.0.15 Source 1

- *interArrivalTime*: Exponentialverteilt mit Parameter 50

10.3.1.0.16 Queue 1

- *WaitingTime*: Gleichverteilt im Intervall [0,0]. Das ist gleichbedeutend mit $WaitingTime = 0$.
- *Kapazität*: 100

10.3.1.0.17 Source 2

- *interArrivalTime*: Exponentialverteilt mit Parameter 100

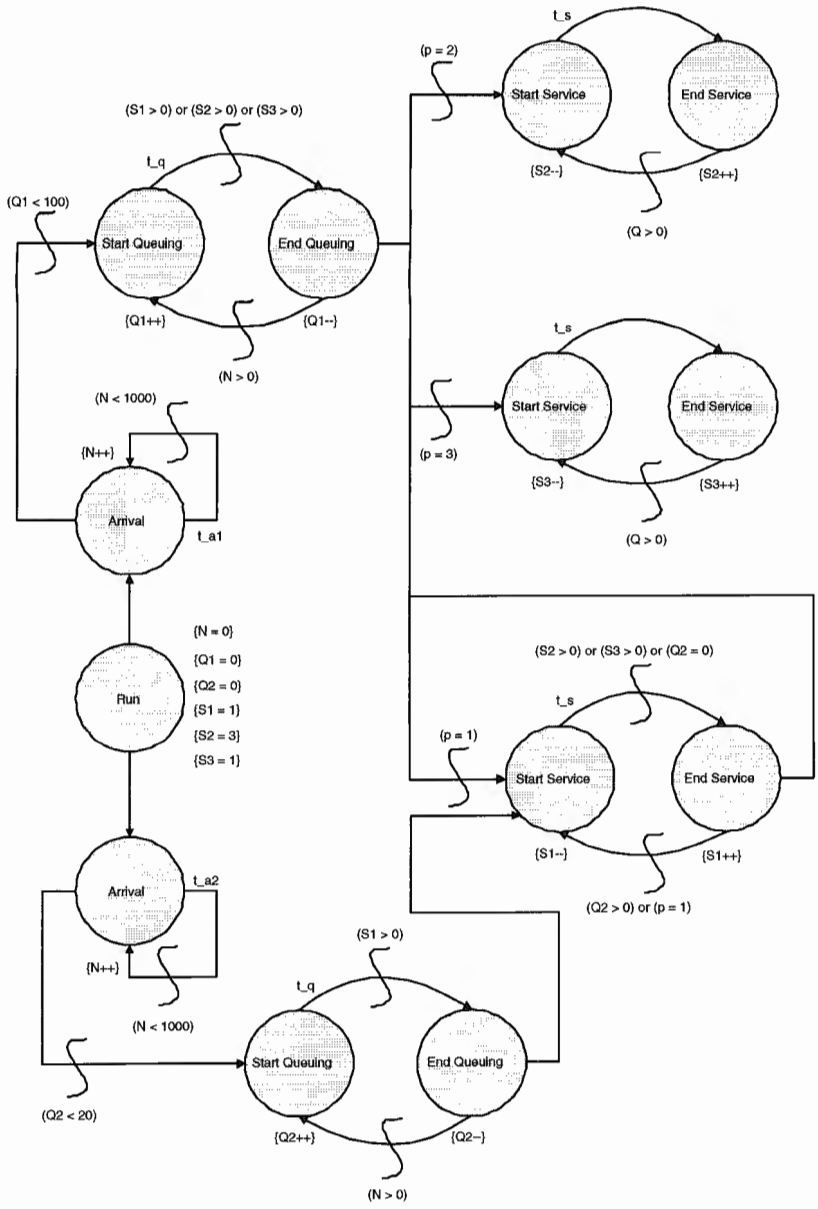


Abbildung 10.6: Ereignisgraph des Multiflow-Modells mit Feedback

10.3.1.0.18 Queue 2

- *WaitingTime*: 0
- *Kapazität*: 20

10.3.1.0.19 Join 1: Das Join-Element dient dem Zusammenführen mehrerer Itemströme und hat keine innere Logik implementiert.

10.3.1.0.20 Server 1

- *ServiceTime*: Exponentialverteilt mit Parameter 5
- *Anzahl gleichzeitiger Serviceprozesse*: 1

10.3.1.0.21 Join 2: Das Join-Element dient dem Zusammenführen mehrerer Itemströme und hat keine innere Logik implementiert.

10.3.1.0.22 Fork

- *Verzweigungsstrategie*: NextFree

10.3.1.0.23 Server 2

- *ServiceTime*: Exponentialverteilt mit Parameter 20
- *Anzahl gleichzeitiger Serviceprozesse*: 3

10.3.1.0.24 Server 3

- *ServiceTime*: Exponentialverteilt mit Parameter 10
- *Anzahl gleichzeitiger Serviceprozesse*: 1

10.3.1.0.25 Join 3: Das Join-Element dient dem Zusammenführen mehrerer itemströme und hat keine innere logik implementiert.

10.3.1.0.26 Sink: Die Sink dient dem Zweck, die im System erzeugten Items einzusammeln und zur statistischen Auswertung Items zu zählen.

Kapitel 11

Zusammenfassung und Ausblick

In diesem abschließenden Kapitel soll noch einmal kurz zusammengefasst werden, was mit der vorliegenden Arbeit erreicht wurde. Der Ausblick soll zeigen, in welchen Bereichen noch Verbesserungen vorzunehmen sind und welche zukünftigen Entwicklungen in dem behandelten Gebiet zu erwarten sind.

11.1 Zusammenfassung

Diese Arbeit zeigt die Möglichkeit der Erweiterung der Modellbildungs- und Simulationstechnik unter Einbeziehung von State of the Art-Internettechnolgie. Ein spezielles Augenmerk wurde dabei auf die Erweiterbarkeit des entwickelten Frameworks und auf die Verwendung offener Softwaresysteme und offener Standards gelegt.
Die Resultate können in einer allgemeinen Weise in den diversen Anwendungsgebieten der Discrete Event Simulation und in eingeschränkter

Form auch in der kontinuierlichen Simulation genutzt werden.

Folgende Ergebnisse wurden präsentiert:

1. Die in der Literatur beschriebene Konvergenz von Simulationstechnik und Webtechnologie kann durch die Verwendung von XML und darauf aufbauender Technologien wie Web Services erreicht werden.

2. Es wurde ein Ansatz eines generischen DES-Modells entwickelt, der es erlaubt, ein Simulationsmodell unabhängig vom gewählten Simulator zu erstellen und weiters die verschiedenen Teilbereiche einer Simulationsstudie modular zu definieren.

3. Es wurde als Web Service Framework ein Grid Service Environment präsentiert, dass es einem Benutzer von Simulationsdienstleistungen erlaubt, die benötigten Dienste im Web dynamisch zu suchen und sie dann auf einfache Weise auszuführen und zu steuern.

11.2 Ausblick

11.2.1 Erweiterung der Modellierungsmöglichkeiten des SimASP Frameworks

Das SimASP Framework hat momentan den Status eines Prototyps, der (zum Teil aus pragmatischen Gründen) noch nicht die ganze geplante Funktionalität implementiert hat. Im Folgenden werden die verschiedenen Bereiche des Frameworks präsentiert, die in einer weiteren Ausbaustufe geplant sind:

- Die Anbindung an weitere Simulatoren: Es sollen weitere Simulatoren verfügbar gemacht werden. Dieser Entwicklungsschritt wird

aber zu einem großen Teil von den Entwicklern der Simulationssoftware abhängen, da zur Entwicklung der Mappinglayer jeweils simulatorspezifisches Know How erforderlich ist.

- Weitere funktionale Module: Es sollen Module für erweiterte statistische Auswertung, Optimierung und Animation entwickelt werden.

- Entwicklung von Modell-Bibliotheken: Um dem Benutzer die Entwicklung von Simulationsmodellen zu erleichtern, sollen Bibliotheken für spezielle Teilbereiche der Modellierung entwickelt werden.

- Erweiterung der Implementierung logischer Zusammenhänge: Momentan können logische Bedingungen nur bedingt modelliert werden. Dieser Teilbereich ist für die Verwendung des Frameworks in komplexen Aufgabenstellungen unumgänglich.

- Anreichern der Items mit Objekteigenschaften: Momentan können Items keine individuellen Eigenschaften und Parameter zugewiesen werden. Dies ist ebenfalls eine grundlegende Erweiterung, die zum Bearbeiten komplexer Aufgabenstellungen erforderlich ist.

11.2.2 Erweiterung der genutzten gridservicespezifischen Funktionalität

Das Vienna Grid Environment ist selbst nur ein Prototyp, es bietet aber bereits jetzt mehr Funktionalität als momentan im SimASP Framework genutzt wird. Der Enwurf eines Simulation Grids ist allerdings ein langfristiges Ziel.

11.2.3 Anschluss an bestehende Initiativen im Simulationsbereich

Das SimASP Framework ist nicht die einzige Initiative, die sich mit der Anwendung von XML im Simulationsbereich beschäftigt. Als Ergebnisse anderer Arbeitsgruppen können Modelica, SRML oder OpenSML genannt werden. Welche dieser Entwicklungen die größten Chancen am "Simulationsmarkt" hat muss noch evaluiert werden. Eine Zusammenarbeit mit solchen Initiativen ist denkbar.

11.2.4 Semantic Web/Ontologien

Die Entwicklung der Webtechnologien macht nicht Halt und nachdem der Durchbruch von Web Services als zentraler Kommunikationsschnittstelle nur als eine Frage der Zeit erscheint, wird bereits über Folgetechnologien nachgedacht.
Das neue Schlagwort ist das *Semantic Web*. Darunter versteht man die Erweiterung des WWWs um maschinenlesbare Daten, welche die Semantik der Inhalte formal festlegen. Informationen sollen zusätzlich zu der für Menschen lesbaren Form auch formal, in einer für Maschinen verarbeitbaren Form repräsentiert werden, damit Programme darauf operieren können. Die Annotation der Information im Web geschieht z.B. mittels Wissens- bzw. Ontologie-Repräsentationssprachen.
Diese Entwicklung hat bereits zu ersten Ansätzen von Ontologien für DES-Modelle geführt und es bleibt abzuwarten, wie sich diese in den nächsten Jahren entwickeln.

Teil V

Anhang

Anhang A

Glossar und Abkürzungen

Apache - Kurzform für Apache Software Foundation, eine ehrenamtlich arbeitende Organisation für Open Source Software

API - Application Programming Interface, eine Programmschnittstelle

ASP - Application Service Providing, eine Dienstleistung, die anderen Firmen Softwareanwendungen über das World Wide Web zur Verfügung stellt

Axis - Apache eXtensible Interaction System, eine SOAP Engine zur Konstruktion von darauf basierenden Webservices und Client-Anwendungen

Client - eine Anwendung, die in einem Netzwerk den Dienst eines Servers in Anspruch nimmt

Deployment - die Verteilung und Installation von Software auf Zielsystemen, einschließlich deren Konfiguration

DES - Discrete Event Simulation, es werden im Gegensatz zur kontinuierlichen Simulation nur endlich viele Zustandsveränderungen pro Zeitintervall durchgeführt

DOM - Document Object Model, eine Programmierschnittstelle für den Zugriff auf HTML- oder XML-Dokumente

DTD - Document Type Definition, eine Deklaration in SGML- und XML-Dokumenten, die die Struktur eines solchen Dokuments festlegt

EAI - Enterprise Application Integration, Integration von verschiedenen Applikationen auf unterschiedlichen Plattformen zur Abwicklung von Geschäftsprozessen

Eclipse - eine freie Software-Entwicklungsumgebung

GGF - Global Grid Forum, Standardisierungsgremium der Grid Technologie

Grid Service - Ein Web Service, der nach OGSA spezifiziert ist

HTML - Hypertext Markup Language, ein Dokumentenformat zur Auszeichnung von Hypertext im World Wide Web

Jakarta Project - ein Projekt der Apache Software Foundation, das freie Software beherbergt, entwickelt und unterstützt, die in der Programmiersprache Java geschrieben wurde

JAVA - eine objektorientierte, plattformunabhängige Programmiersprache

JSIM - einer der im SimASP Framework verwendeten Simualtoren

Namespace - ein Namensraum, der zur eindeutigen Identifizierung eines Objektes benötigt wird und dazu verwendet wird, Konflikte bei der Namensvergabe zu verhindern

OASIS - Organization for the Advancement of Structured Information Standards, eine internationale, nicht-gewinnorientierte Organisation, die sich mit der Weiterentwicklung von Web Service - Standards beschäftigt

OGSA - Open Grid Service Architecture, eine vom GGF verabschiedete Spezifikation für Grid Services

OGSI - Open Grid Service Infrastructure, eine vom GGF verabschiedete Spezifikation für Grid Services

Ontologie - in der Informatik im Bereich der Wissensrepräsentation ein formal definiertes System von Begriffen und/oder Konzepten und Relationen zwischen diesen Begriffen

Open Protocol - das Konzept der Quelloffenheit angewandt auf Kommunikationsprotokolle

Open Source - in der Informatik das Konzept der Quelloffenheit - jedem wird ermöglicht, Einblick in den Quelltext eines Programms zu haben

OWL - Web Ontology Language, eine Spezifikation des W3C, um Ontologien anhand einer formalen Beschreibungssprache erstellen, publizieren und verteilen zu können

PKI - Public-Key-Infrastruktur, ein System, welches es ermöglicht, digitale Zertifikate auszustellen, zu verteilen und zu prüfen

Proxy - ein Computerprogramm, das im Datenverkehr zwischen Client und Server vermittelt

RDF - Resource Description Framework, eine Spezifikation für ein Modell zur Repräsentation von Metadaten

Recommendation - Ein vom W3C empfohlener Standard

RPC - Remote Procedure Call, ein Protokoll für verteilte Anwendungen

SAX - Simple API for XML, eine Bibliothek zur Interpretation und Auswertung von XML-Daten

Semantic Web - eine Erweiterung des World Wide Web (WWW) um maschinenlesbare Daten, welche die Semantik der Inhalte formal festlegen

Server - ein Programm, welches auf die Kontaktaufnahme eines Client-Programmes wartet und nach Kontaktaufnahme mit diesem Nachrichten austauscht

Service - Eine auf einem Server laufende Softwareapplikation, die einem Client über ein Netzwerk zur Verfügung steht

SGML - Standard Generalized Markup Language, eine Metasprache, mit deren Hilfe man verschiedene Auszeichnungssprachen (engl. markup languages) für Dokumente definieren kann.

SimASP - das in dieser Arbeit beschriebene Framework, das Simulation Service Providing mittels Web Servic Technologie ermöglicht

Simkit - einer der im SimASP Framework verwendeten Simualtoren

SOA - Service Oriented Architecture, ein Ansatz zum Entwurf verteilter Systeme im Softwaredesign

SOAP - früher das Simple Object Access Protocol, ein Protokoll, mit dessen Hilfe Daten zwischen Systemen ausgetauscht und Remote Procedure Calls durchgeführt werden können.

SSP - Simulation Service Providing, Simulation als Application Service Providing

Stub - ein lokaler Anknüpfungspunkt für Software, der benutzt werden kann, um ansonsten nur über komplexe Protokolle erreichbare Softwarekomponenten einfach anzusprechen und die Komplexität zu verbergen

Stylesheet - eine Formatvorlage für die Darstellung von Information

Three-Tier-Architektur - eine Client-Server-Architektur, die softwareseitig als dreischichtiges System aufgebaut ist

Tomcat - ein Servletcontainer, der im Jakarta-Projekt der Apache Software Foundation entwickelt wurde

UDDI - Universal Description, Discovery and Integration, einen Verzeichnisdienst für dynamische Webservices

UML - Unified Modeling Language, eine standardisierte Beschreibungssprache für Strukturen und Abläufe in objektorientierten Programmsystemen

URI - Uniform Resource Identifier, ist eine Zeichenfolge, die zur Identifizierung einer abstrakten oder physikalischen Ressource im Internet und dort vor allem im WWW dient

VGE - Vienna Grid Environment, ein Framework auf Basis von Grid Services, das im SimASP die Client - Server Schnittstelle realisiert

W3C - World Wide Web Consortium, das bedeutendste Gremium zur Standardisierung der das World Wide Web betreffenden Techniken

Web Service - eine Softwareanwendung, die per Uniform Resource Identifier eindeutig identifizierbar ist und deren Schnittstellen als XML - Artefakte definiert, beschrieben und gefunden werden können

WSDL - Web Services Description Language, ein plattform-, programmiersprachen- und protokollunabhängiger XML-Standard zur Beschreibung von Webservices

WSRF - Web Services Resource Framework, definiert ein generisches und offenes Framework für die Modellierung und den Zugriff auf zustandsbehaftete Ressourcen über Web Services

WSS4J - Web Service Security for Java, ein Unterprojekt des Apache Web Services Projekt, das sich mit Security - Aspekten für Web Services beschäftigt.

WWW - World Wide Web, ein über das Internet abrufbares Hypertext-System

XML - Extensible Markup Language, ein Standard zur Erstellung maschinen- und menschenlesbarer Dokumente in Form einer Baumstruktur

XML Encryption - eine Spezifikation, die eine Reihe von Möglichkeiten, wie XML-Dokumente ver- und entschlüsselt werden können, definiert

XML Schema - eine Empfehlung des W3C zum Definieren von XML-Dokumentstrukturen. Anders als bei den klassischen XML DTDs wird die Struktur in Form eines XML-Dokuments beschrieben.

XML Signature - eine Spezifikation, die eine XML-Schreibweise für digitale Signaturen definiert

XSL - Extensible Stylesheet Language, eine Familie von Sprachen zur Erzeugung von Layouts für XML-Dokumente

XSLT - Extensible Stylesheet Language Transformation, eine Programmiersprache zur Transformation von XML-Dokumenten

X.509 - ein Standard für eine Public Key-Infrastruktur und derzeit der wichtigste Standard für digitale Zertifikate

Anhang B

XML Schemata des SimASP Frameworks

B.1 XML Schema Definition eines DES Modells

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <xsd:complexType name="locationType">
  <xsd:attribute name="x" type="xsd:float" use="required"/>
  <xsd:attribute name="y" type="xsd:float" use="required"/>
 </xsd:complexType>
 <xsd:complexType name="objectType" abstract="true">
  <xsd:attribute name="ID" type="xsd:ID" use="required"/>
 </xsd:complexType>
 <xsd:complexType name="entityType">
  <xsd:complexContent>
```

```
    <xsd:extension base="objectType">
     <xsd:sequence>
      <xsd:element name="location"
                                type="locationType"/>
      <xsd:element name="name" type="xsd:string"/>
     </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="ConnectorType">
 <xsd:complexContent>
  <xsd:extension base="objectType">
   <xsd:sequence>
    <xsd:element name="startObject">
     <xsd:complexType>
      <xsd:attribute name="IDRef" type="xsd:IDREF"/>
     </xsd:complexType>
    </xsd:element>
    <xsd:element name="startLocation"
                                type="locationType"/>
    <xsd:element name="endObject">
     <xsd:complexType>
      <xsd:attribute name="IDRef" type="xsd:IDREF"/>
     </xsd:complexType>
    </xsd:element>
    <xsd:element name="endLocation"
                                type="locationType"/>
   </xsd:sequence>
  </xsd:extension>
 </xsd:complexContent>
```

```
</xsd:complexType>
<xsd:complexType name="ForkType">
 <xsd:complexContent>
  <xsd:extension base="objectType">
   <xsd:sequence>
    <xsd:element name="strategy">
     <xsd:complexType>
      <xsd:choice>
       <xsd:element name="nextFree"/>
       <xsd:element name="broadcast"/>
      </xsd:choice>
     </xsd:complexType>
    </xsd:element>
   </xsd:sequence>
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="JoinType">
 <xsd:complexContent>
  <xsd:extension base="objectType">
   <xsd:sequence/>
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="QueueType">
 <xsd:complexContent>
  <xsd:extension base="entityType">
   <xsd:sequence>
    <xsd:element name="numberOfElements"
                              type="xsd:integer"/>
```

```
    <xsd:element name="waitingTime"
                 type="RandomVariateStream"/>
   </xsd:sequence>
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="ServerType">
 <xsd:complexContent>
  <xsd:extension base="entityType">
   <xsd:sequence>
    <xsd:element name="numberOfServer"
                                  type="xsd:int"/>
    <xsd:element name="serviceTime"
                 type="RandomVariateStream"/>
   </xsd:sequence>
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="SinkType">
 <xsd:complexContent>
  <xsd:extension base="entityType"/>
 </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="SourceType">
 <xsd:complexContent>
  <xsd:extension base="entityType">
   <xsd:sequence>
    <xsd:element name="interarrivalTime"
                 type="RandomVariateStream"/>
    <xsd:element name="maxItems" type="xsd:int"/>
```

```
    </xsd:sequence>
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>
 <xsd:complexType name="statCollectionType">
  <xsd:sequence>
   <xsd:element name="ID" type="xsd:ID"
                                        minOccurs="0">
    <xsd:annotation>
     <xsd:documentation>Referenz auf Object ID
                                  </xsd:documentation>
    </xsd:annotation>
   </xsd:element>
   <xsd:element name="displayName" type="xsd:string">
    <xsd:annotation>
     <xsd:documentation>Anzeige- oder Objektname
                                  </xsd:documentation>
    </xsd:annotation>
   </xsd:element>
   <xsd:element name="filename" type="xsd:string">
    <xsd:annotation>
     <xsd:documentation>FileressourceName
                                  </xsd:documentation>
    </xsd:annotation>
   </xsd:element>
   <xsd:element name="numberItems" type="xsd:int">
    <xsd:annotation>
     <xsd:documentation>Anzahl entities
                                  </xsd:documentation>
    </xsd:annotation>
```

```
    </xsd:element>
    <xsd:element name="minTime" type="xsd:double"/>
    <xsd:element name="averageTime" type="xsd:double"/>
    <xsd:element name="maxTime" type="xsd:double"/>
    <xsd:element name="stdDev"/>
   </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="RandomVariateStream"/>
 <xsd:element name="Simulation">
  <xsd:complexType>
   <xsd:sequence>
    <xsd:element name="head">
     <xsd:complexType>
      <xsd:all>
       <xsd:element name="title" type="xsd:string"/>
       <xsd:element name="replications"
                                    type="xsd:integer"/>
       <xsd:element name="initial-seed" type="xsd:long"/>
       <xsd:element name="stopCriteria">
        <xsd:complexType>
         <xsd:choice>
          <xsd:element name="max-time" type="xsd:double"/>
          <xsd:element name="max-items" type="xsd:long"/>
         </xsd:choice>
        </xsd:complexType>
       </xsd:element>
       <xsd:element name="simulator"/>
       <xsd:element name="outputFile"/>
      </xsd:all>
     </xsd:complexType>
```

```
  </xsd:element>
  <xsd:element name="model">
     <xsd:complexType>
      <xsd:sequence>
       <xsd:element ref="simObject" minOccurs="0"
                                    maxOccurs="unbounded"/>
       <xsd:element ref="simEntity" minOccurs="0"
                                    maxOccurs="unbounded"/>
      </xsd:sequence>
     </xsd:complexType>
    </xsd:element>
   </xsd:sequence>
  </xsd:complexType>
 </xsd:element>
 <xsd:element name="simObject" type="objectType"
                                        abstract="true">
</xsd:element>
<xsd:element name="simEntity" type="entityType">
</xsd:element>
<xsd:element name="connector" type="ConnectorType"
                      substitutionGroup="simObject"/>
<xsd:element name="queue" type="QueueType"
                      substitutionGroup="simEntity"/>
<xsd:element name="server" type="ServerType"
                      substitutionGroup="simEntity"/>
<xsd:element name="sink" type="SinkType"
                      substitutionGroup="simEntity"/>
<xsd:element name="source" type="SourceType"
                      substitutionGroup="simEntity"/>
<xsd:element name="fork" type="ForkType"
```

```
                    substitutionGroup="simEntity"/>
 <xsd:element name="join" type="JoinType"
                    substitutionGroup="simEntity"/>
</xsd:schema>
```

B.2 XML Schema Definition des statistischen Outputs

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:complexType name="statCollectionType">
    <xs:sequence>
      <xs:element name="ID" type="xs:ID">
        <xs:annotation>
          <xs:documentation>Referenz auf Object
                              ID</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="description" type="xs:string"/>
      <xs:element name="numberItemsIn" type="xs:int"/>
      <xs:element name="numberItemsOut" type="xs:int"
                                    minOccurs="0"/>
      <xs:element name="timeStat" minOccurs="0"
                              maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="min" type="xs:double"/>
            <xs:element name="mean" type="xs:double"/>
```

```
        <xs:element name="max" type="xs:double"/>
        <xs:element name="variance"
                                type="xs:double"/>
      </xs:sequence>
       <xs:attribute name="typ" use="required">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="arrival"/>
            <xs:enumeration value="delay"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
      <xs:attribute name="name" type="xs:string"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="loadStat" minOccurs="0"
                                    maxOccurs="2">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="mean" type="xs:double"/>
        <xs:element name="max" type="xs:int"/>
      </xs:sequence>
      <xs:attribute name="name">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="in process"/>
            <xs:enumeration value="stacked"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
```

```
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>
<xs:element name="statisticsOutput">
  <xs:complexType>
    <xs:choice>
      <xs:element name="error" type="xs:anyType"/>
      <xs:sequence>
        <xs:element name="head">
          <xs:complexType>
            <xs:all>
              <xs:element name="title"
                                type="xs:string"/>
              <xs:element name="replications"
                               type="xs:integer"/>
              <xs:element name="initial-seed"
                                 type="xs:long"/>
              <xs:element name="simulationStat">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="simTime"
                                type="xs:double"/>
                    <xs:element name="numberItemsIn"
                                  type="xs:long"/>
                    <xs:element name="numberItemsOut"/>
                    <xs:element
                    name="averageThroughputTime"/>
                  </xs:sequence>
                </xs:complexType>
```

```
            </xs:element>
            <xs:element name="simulator">
              <xs:complexType/>
            </xs:element>
          </xs:all>
        </xs:complexType>
      </xs:element>
      <xs:element name="replicationStat"
                          maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="objectStats"
              type="statCollectionType" minOccurs="0"
                              maxOccurs="unbounded"/>
          </xs:sequence>
          <xs:attribute name="number" type="xs:int"
                                      use="required"/>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:choice>
 </xs:complexType>
</xs:element>
</xs:schema>
```

Anhang C

XML Darstellung der Beispielfiles

C.1 Job Shop Modell

```
<?xml version="1.0"?>
<Simulation
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:noNamespaceSchemaLocation="Input_SimASP.xsd">
  <head>
    <title>Beispiel 1: Job Shop Model</title>
    <replications>3</replications>
    <initial-seed>1</initial-seed>
    <stopCriteria>
      <max-items>1000</max-items>
    </stopCriteria>
    <simulator>jsim</simulator>
  </head>
```

```
<model>
  <source ID="10">
    <location x="1.0" y="1.0"/>
    <name>Source</name>
    <interarrivalTime variate="Exponential">
      <parameter>100.0</parameter>
    </interarrivalTime>
  </source>
  <queue ID="20">
    <location x="2.0" y="1.0"/>
    <name>First Queue</name>
    <waitingTime variate="Exponential">
      <parameter>10.0</parameter>
    </waitingTime>
    <capacity>100</capacity>
  </queue>
  <server ID="30">
    <location x="3.0" y="1.0"/>
    <name>First Server</name>
    <serviceTime variate="Exponential">
      <parameter>80.0</parameter>
    </serviceTime>
    <capacity>1</capacity>
  </server>
  <sink ID="40">
    <location x="4.0" y="1.0"/>
    <name>Sink</name>
  </sink>
  <connector ID="100">
    <startObject IDRef="10"/>
```

```
      <startLocation x="1.0" y="1.0"/>
      <endLocation x="2.0" y="1.0"/>
      <endObject IDRef="20"/>
    </connector>
    <connector ID="110">
      <startObject IDRef="20"/>
      <startLocation x="2.0" y="1.0"/>
      <endLocation x="3.0" y="1.0"/>
      <endObject IDRef="30"/>
    </connector>
    <connector ID="120">
      <startObject IDRef="30"/>
      <startLocation x="3.0" y="1.0"/>
      <endLocation x="4.0" y="1.0"/>
      <endObject IDRef="40"/>
    </connector>
  </model>
</Simulation>
```

C.2 Multiple Server Lines Modell

```
<?xml version="1.0"?>
<Simulation
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:noNamespaceSchemaLocation="Input_SimASP.xsd">
  <head>
    <title>Beispiel 2: Multiple Server Lines</title>
    <replications>2</replications>
    <initial-seed>1</initial-seed>
    <stopCriteria>
```

```
    <max-time>10000</max-time>
  </stopCriteria>
  <simulator>simkit</simulator>
</head>
<model>
  <source ID="10">
    <location x="1.0" y="1.0"/>
    <name>Source</name>
    <interarrivalTime variate="Exponential">
      <parameter>5.0</parameter>
    </interarrivalTime>
  </source>
  <queue ID="20">
    <location x="2.0" y="1.0"/>
    <name>First Queue</name>
    <waitingTime variate="Exponential">
      <parameter>1.0</parameter>
    </waitingTime>
    <capacity>100</capacity>
  </queue>
  <fork ID="30">
    <location x="3.0" y="1.0"/>
    <name>Distribution fork</name>
    <strategy>nextFree</strategy>
  </fork>
  <server ID="40">
    <location x="4.0" y="0.5"/>
    <name>Fast Lane Server</name>
    <serviceTime variate="Exponential">
      <parameter>20.0</parameter>
```

```
    </serviceTime>
    <capacity>3</capacity>
  </server>
  <server ID="50">
    <location x="4.0" y="1.5"/>
    <name>Slow Lane Server</name>
    <serviceTime variate="Exponential">
      <parameter>10.0</parameter>
    </serviceTime>
    <capacity>1</capacity>
  </server>
  <join ID="60">
    <location x="5.0" y="1.0"/>
    <name>Collector</name>
  </join>
  <sink ID="70">
    <location x="6.0" y="1.0"/>
    <name>Sink</name>
  </sink>
  <connector ID="100">
    <startObject IDRef="10"/>
    <startLocation x="1.0" y="1.0"/>
    <endLocation x="2.0" y="1.0"/>
    <endObject IDRef="20"/>
  </connector>
  <connector ID="110">
    <startObject IDRef="20"/>
    <startLocation x="2.0" y="1.0"/>
    <endLocation x="3.0" y="1.0"/>
    <endObject IDRef="30"/>
```

```
</connector>
<connector ID="120">
  <startObject IDRef="30"/>
  <startLocation x="3.0" y="1.0"/>
    <endLocation x="4.0" y="0.5"/>
    <endObject IDRef="40"/>
</connector>
<connector ID="130">
  <startObject IDRef="30"/>
  <startLocation x="3.0" y="1.0"/>
  <endLocation x="4.0" y="1.5"/>
  <endObject IDRef="50"/>
</connector>
<connector ID="140">
  <startObject IDRef="40"/>
  <startLocation x="4.0" y="0.5"/>
  <endLocation x="5.0" y="1.0"/>
  <endObject IDRef="60"/>
</connector>
<connector ID="150">
  <startObject IDRef="50"/>
  <startLocation x="4.0" y="1.5"/>
  <endLocation x="5.0" y="1.0"/>
  <endObject IDRef="60"/>
</connector>
<connector ID="160">
  <startObject IDRef="60"/>
  <startLocation x="5.0" y="1.0"/>
  <endLocation x="6.0" y="1.0"/>
  <endObject IDRef="70"/>
```

```
    </connector>
  </model>
</Simulation>
```

C.3 Multiflow Modell mit Feedback

```
<?xml version="1.0"?>
<Simulation
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:noNamespaceSchemaLocation="Input_SimASP.xsd">
  <head>
    <title>Beispiel 3: Multiflow Model mit Feedback</title>
    <replications>1</replications>
    <initial-seed>1</initial-seed>
    <stopCriteria>
      <max-items>1000</max-items>
    </stopCriteria>
    <simulator>simkit</simulator>
  </head>
  <model>
    <source ID="5">
      <location x="1.0" y="1.0"/>
      <name>North Source</name>
      <interarrivalTime variate="Exponential">
        <parameter>50.0</parameter>
      </interarrivalTime>
    </source>
    <queue ID="10">
      <location x="2.0" y="1.0"/>
      <name>North Queue</name>
```

```
    <waitingTime variate="Uniform">
      <parameter>0.0</parameter>
      <parameter>0.0</parameter>
    </waitingTime>
    <capacity>100</capacity>
  </queue>
  <source ID="15">
    <location x="1.0" y="4.0"/>
    <name>South Source</name>
    <interarrivalTime variate="Exponential">
      <parameter>100.0</parameter>
    </interarrivalTime>
  </source>
  <queue ID="20">
    <location x="2.0" y="4.0"/>
    <name>South Queue</name>
    <capacity>20</capacity>
  </queue>
  <join ID="25">
    <location x="3.0" y="4.0"/>
    <name>Feedback Collector</name>
  </join>
  <server ID="30">
    <location x="4.0" y="3.0"/>
    <name>South Controller</name>
    <serviceTime variate="Exponential">
      <parameter>5.0</parameter>
    </serviceTime>
    <capacity>1</capacity>
  </server>
```

```
<join ID="35">
  <location x="3.0" y="2.0"/>
  <name>North-South Collector</name>
</join>
<fork ID="40">
  <location x="4.0" y="2.0"/>
  <name>Feedback Distributor</name>
  <strategy>nextFree</strategy>
</fork>
<server ID="45">
  <location x="5.0" y="1.0"/>
  <name>Fast Lane Server</name>
  <serviceTime variate="Exponential">
    <parameter>20.0</parameter>
  </serviceTime>
  <capacity>3</capacity>
</server>
<server ID="50">
  <location x="5.0" y="3.0"/>
  <name>Slow Lane Server</name>
  <serviceTime variate="Exponential">
    <parameter>10.0</parameter>
  </serviceTime>
  <capacity>1</capacity>
</server>
<join ID="55">
  <location x="6.0" y="2.0"/>
  <name>End Join</name>
</join>
<sink ID="60">
```

```
  <location x="7.0" y="2.0"/>
  <name>Sink</name>
</sink>
<connector ID="10510">
  <startObject IDRef="5"/>
  <startLocation x="1.0" y="1.0"/>
  <endLocation x="2.0" y="1.0"/>
  <endObject IDRef="10"/>
</connector>
<connector ID="11035">
  <startObject IDRef="10"/>
  <startLocation x="2.0" y="1.0"/>
  <endLocation x="3.0" y="2.0"/>
  <endObject IDRef="35"/>
</connector>
<connector ID="11520">
  <startObject IDRef="15"/>
  <startLocation x="1.0" y="4.0"/>
  <endLocation x="2.0" y="4.0"/>
  <endObject IDRef="20"/>
</connector>
<connector ID="12025">
  <startObject IDRef="20"/>
  <startLocation x="2.0" y="4.0"/>
  <endLocation x="3.0" y="4.0"/>
  <endObject IDRef="25"/>
</connector>
<connector ID="12530">
  <startObject IDRef="25"/>
  <startLocation x="3.0" y="4.0"/>
```

```
    <endLocation x="3.0" y="3.0"/>
    <endObject IDRef="30"/>
  </connector>
  <connector ID="13035">
    <startObject IDRef="30"/>
    <startLocation x="3.0" y="3.0"/>
    <endLocation x="3.0" y="2.0"/>
    <endObject IDRef="35"/>
  </connector>
  <connector ID="13540">
    <startObject IDRef="35"/>
    <startLocation x="3.0" y="2.0"/>
    <endLocation x="4.0" y="2.0"/>
    <endObject IDRef="40"/>
  </connector>
  <connector ID="14025">
    <startObject IDRef="40"/>
    <startLocation x="4.0" y="2.0"/>
    <endLocation x="3.0" y="4.0"/>
    <endObject IDRef="25"/>
  </connector>
  <connector ID="14045">
    <startObject IDRef="40"/>
    <startLocation x="4.0" y="2.0"/>
    <endLocation x="5.0" y="1.0"/>
    <endObject IDRef="45"/>
  </connector>
  <connector ID="14555">
    <startObject IDRef="45"/>
    <startLocation x="5.0" y="1.0"/>
```

```
      <endLocation x="6.0" y="2.0"/>
      <endObject IDRef="55"/>
    </connector>
    <connector ID="14050">
      <startObject IDRef="40"/>
      <startLocation x="4.0" y="2.0"/>
      <endLocation x="5.0" y="3.0"/>
      <endObject IDRef="50"/>
    </connector>
    <connector ID="15055">
      <startObject IDRef="50"/>
      <startLocation x="5.0" y="3.0"/>
      <endLocation x="6.0" y="2.0"/>
      <endObject IDRef="55"/>
    </connector>
    <connector ID="15560">
      <startObject IDRef="55"/>
      <startLocation x="6.0" y="2.0"/>
      <endLocation x="7.0" y="2.0"/>
      <endObject IDRef="60"/>
    </connector>
  </model>
</Simulation>
```

Literaturverzeichnis

[1] Serge Abiteboul; Peter Buneman; Dan Suciu: *Data on the Web: from relations to semistructured data and XML.* - San Francisco, Calif.: Morgan Kaufmann, 2000

[2] Alves-Foss, Jim [Hrsg.]: *Formal syntax and semantics of Java.* - Berlin [u.a.]: Springer, 1999

[3] Ammelburger, Dirk: *XML mit Java: eine XML-Einführung für Java-Programmierer; die APIs, Three-Tier-Anwendungen, Servlets, XML-RPC u.v.a.m..* - München: Markt-und-Technik-Verl., 2002

[4] ANSA: *The Advanced Network Systems Architecture (ANSA) Reference Manual.* Architecture Project Management - Castle Hill, Cambridge England, 1989

[5] Apache Project: http://www.apache.org, 08.2005

[6] Apache software license: http://www.apache.org/LICENSE.txt, 03.2003

[7] WebServices - Axis: http://ws.apache.org/axis/, 08.2005

[8] Chris Bates: *XML in theory and practice: [features Java, DTD, navigation, schema, CSS, XSLT, XSL-FO, Sax, Dom, DocBook, web services, SOAP, XUL].* - Chichester [u.a.]: Wiley, 2003

[9] Siegfried Benkner, Ivona Brandic, Gerhard Engelbrecht, Rainer Schmidt: *VGE - A Service-Oriented Grid Environment for On-Demand Supercomputing.* - Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing (GRID04), 2004

[10] Berman, F.; Hey, A.; Fox, G.: *Grid computing: Making the global infrastructure a reality..* - Wiley, 2003

[11] Wendy Boggs; Michael Boggs: *UML mit Rational Rose: [Softwaredesign für professionelle Entwickler; Reverse Engineering; automatische Codegenerierung; professionelles Anwendungsdesign].* - Bonn: mitp, 2003

[12] Jacques Boy; Christian Dudek; Sabine Kuschel: *Projektmanagement: Grundlagen, Methoden und Techniken, Zusammenhänge.* - Bremen: GABAL, 1994

[13] Francois Bry, Wolfgang E. Nagel, Michael Schroeder: *Grid-Computing.* - Informatik Spektrum 13, 552 - 554, 2004

[14] Buss, A.: *Component-Based Simulation Modeling.* - Proceedings of the 2000 Winter Simulation Conference, J. A. Joines, R. R. Barton, K. Kang, and P. A. Fishwick, eds., 2000

[15] Arnold Buss: *Basic Event Graph Modeling.* - Technical Notes, Simulation News Europe 31, 2001

[16] Arnold Buss: *Discrete Event Programming with Simkit.* - Technical Notes, Simulation News Europe 32, Vienna, 2001

[17] Senthilanand Chandrasekaran, Gregory Silver, John A. Miller, Jorge Cardoso, and Amit P. Sheth: *Web Service Technologies and their Synergy with Simulation.* - In Proceedings of the 2002 Winter Simulation Conference, 2002

[18] David A. Chappell; Tyler Jewell: *Java Web Services.* - Köln [u.a.]: O'Reilly, 2003

[19] Dean C. Chatfield (Virginia Tech) and Terry P. Harrison and Jack C. Hayya: *SISCO: A Supply Chain Simulation Tool Utilizing Silk and XML.* - In Proceedings of the 2001 Winter Simulation Conference, 2001

[20] S. Chokhani, W. Ford, R. Sabett, C. Merrill, S. Wu: *Internet X.509 Public Key Infrastructure Certificate Policy and Certification Practices Framework,.* - http://www.ietf.org/rfc/rfc3647.txt, The Internet Society, 2003

[21] Thorsten S. Daum, and Robert G. Sargent: *A Web-Ready HiMASS: Facilitating Collaborative, Reusable, and Distributed Modeling and Execution of Simulation Models with XML.* - In Proceedings of the 2002 Winter Simulation Conference, 2002

[22] DTD: http://www.w3.org/TR/2004/REC-xml-20040204, 08.2005

[23] Reiner Dumke: *Web Engineering.* - München: Pearson Studium, 2003

[24] Schahram Dustdar; Harald Gall; Manfred Hauswirth: *Software-Architekturen für verteilte Systeme: Prinzipien, Bausteine und Standardarchitekturen für moderne Software.* - Berlin [u.a.]: Springer, 2003

[25] Rainer Eckstein, Silke Eckstein: *XML und Datenmodellierung.* - Heidelberg, dPunkt.Verlag, 2004

[26] Eclipse: http://eclipse.org/

[27] Gerald Ehmayer; Siegfried Reich: *Java in der Anwendungsentwicklung: Objektorientierung, Verteilung, Datenbanken.* - Heidelberg: dPunkt, Verl. für digitale Technologie, 1998

[28] Endlich Ulrich: *An evaluation of enterprise application integration solutions and the role of Web services* . - Wien, Dissertation der TU Wien, 2004

[29] Fishwick, P.A: *Web-Based Simulation: Some Personal Observations.* - In Proceedings of the 1996 Winter Simulation Conference, 772-779, 1996

[30] Paul A. Fishwick: *Using XML for Simulation Modeling.* - In Proceedings of the 2002 Winter Simulation Conference, 2002

[31] 5. Foster I., Kesselman C.,Nick J.M.,Tuecke S.: *The physiology of the grid An open grid services architecture for distributed systems integration.* - http://www.globus.org/research/papers/ogsa.pdf, 2002

[32] Foster, I.; Kesselman, C.: *The Grid: Blueprint for a new computing infrastructure.* - Morgan Kaufmann, 2004

[33] Martin Fowler with Kendall Scott: *UML distilled: a brief guide to the standard object modeling language.* - Boston [u.a.]: Addison-Wesley, 2000

[34] Elmar Geese; Markus Heiliger: *Internetlösungen für VB- und Web-Entwickler.* - Bonn: Galileo Press, 2000

[35] Global Grid Forum: http://www.ggf.org/, 08.2005

[36] Randall Gibson, D. J. Medeiros, Andrew Sudar, Bill Waite and Matthew W. Rohrer: *Increasing Return on Investment from Simulation.* - In Proceedings of the 2003 Winter Simulation Conference, 2003

[37] Gnu general public license: http://www.gnu.org/copyleft/gpl.html, 03.2003

[38] Tom-David Graupner, Hendrik Richter, and Wilfried Sihn: *Configuration, Simulation and Animation of Manufacturing Systems via the Internet.* - Proceedings of the 2002 Winter Simulation Conference, 2002

[39] Gutierrez, Dan: *Web-Datenbanken für Windows-Plattformen: ADO, OLE DB, ODBC, JDBC und mehr.* - München: Markt-und-Technik-Verl., 2000

[40] M. Gyimesi, F. Breitenecker: *Simulation Service Providing als Web Service.* - Erlangen, Angenommenes Paper der ASIM Tagung, 2005

[41] author: *SimASP - An Open Source Application Service Providing Framework for Simulation.* - Accepted Paper der Simulation News Europe SNE, 2005

[42] James O. Henriksen, Peter Lorenz, Andr Hanisch, Stefan Osterburg and Thomas J. Schriber: *Web based Simulation Center: Professional Support for Simulation Projects.* - Proceedings of the 2002 Winter Simulation Conference, 2002

[43] Martin Hitz; Gerti Kappel: *UML @ work.* - Heidelberg: dpunkt.Verl., 2003

[44] Peter H.M. Jacobs, Niels A. Lang, and Alexander Verbraeck: *D-SOL; A Distributed Java based Discrete Event Simulation Architecture..* - In Proceedings of the 2002 Winter Simulation Conference, 2002

[45] JAVA: http://java.sun.com, 08.2005

[46] JSIM Web Page: http://chief.cs.uga.edu/ jam/jsim/

[47] Gerti Kappel Hrsg.: *Web Engineering: systematische Entwicklung von Web-Anwendungen.* - Heidelberg: dpunkt-Verl., 2004

[48] Gerti Kappel: *Unterlagen zur Vorlesung Web Engineering.* - TU Wien, 2004

[49] Markus Keidl, Alfons Kemper, Stefan Seltzsam, Konrad Stocker: *Web Services* aus *Web und Datenbanken. Konzepte, Architekturen, Anwendungen.* - dpunkt, 2003

[50] Kilgore, R. A., Healy, K. J. and Kleindorfer: *The future of Java-based simulation.* - Proceedings of the 1998 Winter Simulation Conference, 1998

[51] Kilgore, R., K. Healy, and G. Kleindorfer: *SilkTM: usable and reusable Java-based object-oriented simulation.* - Proceedings of the 12th European Simulation Multiconference, 1998

[52] Richard A. Kilgore and Emmett Burke: *Object-Oriented Simulation of Distributed Systems Using Java and Silk.* - Proceedings of the 2000 Winter Simulation Conference, 2000

[53] Kilgore, R. A: *Open source simulation modeling language (SML).* - In Proceedings of the 2001 Winter Simulation Conference, ed., B. Peters,J. Smith. Piscataway, NJ: 2001

[54] Kilgore, R. A.: *Object-Oriented Simulation with Java, Silk and OpenSML.Net languages.* - In Proceedings of the 2002 Winter Simulation Conference, ed., 2002

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

[56] Michael Knuth: *Web Services: Einführung und Übersicht.* - Frankfurt: Software-und-Support-Verl., 2002

[57] Donald Kossmann, Frank Leymann: *Web Services.* - Informatik Spektrum 26, 117 - 128, 2004

[58] Henrik Kreutz und Johann Bacher Hrsg.: *Disziplin und Kreativität: sozialwissenschaftliche Computersimulation; theoretische Experimente und praktische Anwendung.* - Opladen: Leske + Budrich, 1991

[59] Kuljis, J. and R. J. Paul: *A Review of web based simulation: whiter we wander?.* - Proceedings of the 2000 Winter Simulation Conference, Orlando Florida, page 1872-1881, 2000

[60] Torsten Langner: *Web-basierte Anwendungsentwicklung: die wichtigsten Technologien für Webapplikationen im Überblick.* - Heidelberg [u.a.]: Elsevier, Spektrum Akad. Verl., 2004

[61] Law, A. and D. Kelton: *Simulation Modeling and Analysis, Third Edition,.* - McGraw-Hill, Boston, 2000

[62] Leymann, F., Roller, D.: *Production workflow concepts and techniques.* PTR Prentice Hall, 2000

[63] Leymann, F., Roller, D.: *Flows in information integration.* - IBM Systems Journal 41(4), 2002

[64] Leymann, F.: *Web services: Distributed applications without limits.* - Proceedings BTW03, (Leipzig, Germany, February 2628, 2003). Berlin Heidelberg New York Tokio: Springer, 2003

[65] Linux: http://www.linux.org, 08.2005

[66] Henning Lobin: *Informationsmodellierung in XML und SGML.* - Berlin [u.a.]: Springer, 2000

[67] P.Lorenz: *Web-basierte Simulation und HLA.* - Ghent, Belgien, Modellierung, Simulation und Künstöiche Intelligenz - Statusband, 417 - 436, 2000

[68] Charles McLean and Swee K. Leong, Charley Harrell, Philomena M. Zimmerman and Roberto F. Lu: *Simulation Standards: Current Status, Needs, and Future Directions.* - Proceedings of the 2003 Winter Simulation Conference, 2003

[69] John A. Miller, Rajesh Nair, Zhiwei Zhang and Hongwei Zhao: *JSIM: A Java-Based Simulation and Animation Environment.* - Proceedings of the 30th Annual Simulation Symposium (ANSS'97), Atlanta, Georgia, April 1997

[70] John A. Miller, Andrew F. Seila and Xuewei Xiang: *TThe JSIM Web-Based Simulation Environment.* - Future Generation Computer Systems (FGCS), Special Issue on Web-Based Modeling and Simulation, Vol. 17, No. 2 pp. 119-133. Elsevier North-Holland, October 2000

[71] Monsef, Youssef: *Modelling and simulation of complex systems.* - Erlangen [u.a.]: Society for Computer Simulation, 1997

[72] Rajesh Nair, John A. Miller and Zhiwei Zhang: *A Java-Based Query Driven Simulation Environment.* - Proceedings of the 1996 Winter Simulation Conference (WSC'96), Coronado, California, 1996

[73] Rajesh S. Nair: *JSIM: A Java-Based Query Driven Simulation and Animation Environment.* - Masters Thesis, University of Georgia, March 1997

[74] Bernd Oestereich: *Die UML-Kurzreferenz für die Praxis.* - München; Wien: Oldenbourg, 2001

[75] Tuecke S. et al: *Open grid service infrastructure (OGSI) version 1.0, GGF Proposed Recommendation.* - http://www.ggf.org/ogsi-wg/, 2003

[76] Open Services Gateway Initiative: http://www.osgi.org/, 08.2005

[77] Open Science Workplace: http://www.oswp.info, 08.2005

[78] Pawletta, T.: *Simulation von Fertigungssystemen.* - Wismar. Vorlesungsunterlagen

[79] Erik. T. Ray: *Einführung in XML.* - Köln, O'Reilly, 2001

[80] Steven W. Reichenthal: *Re-Introducing Web-Based Simulation.* - In Proceedings of the 2002 Winter Simulation Conference, 2002

[81] Reinefeld A., Schintke F: *Concepts and technologies for a worldwide grid infrastructure.* - Euro-Par 2002, Aug. 2002, Springer LNCS 2400, pp. 6271, 2002

[82] Alexander Reinefeld, Florian Schintke: *Grid Services - Web Services zur Nutzung verteilter Ressourcen.* - Informatik Spektrum 26, 129 - 135, 2004

[83] Thilo Rottach; Sascha Groß: *XML kompakt: die wichtigsten Standards.* - Heidelberg [u.a.]: Spektrum, Akad. Verl., 2002

[84] Rumbaugh, J.; Blaha, M.; Premerlani, W; Eddy, F.; Lorensen, W.: *Objektorientiertes Modellieren und Entwerfen*, Prentice-Hall International, London, 1993

[85] Schatten Alexander, A Min Tjoa, Amin Andjomshoa, M. Hassan Shafazand: *Building an Web-Based Open Source Tool to Enhance Project Management, Monitoring and Collaboration in Scientific Projects.* - Proceedings of the third International Conference on Information Integration, Web-Applications and Services, 2001

[86] Schatten Alexander: *Sustainable Web-Based Organisation of Project-Related Information and Knowledge.* - PhD Thesis, TU Wien, 2003

[87] Schatten Alexander: *Advanced XML-based Cross Publishing using Open Source Tools.* - Invited Talk at International Conference of Information Integration, Web Applications and Services 2004

[88] Schruben, L. and E. Yücesan: *Modeling Paradigms for Discrete Event Simulation.* - Operations Research Letters, 13, 265275, 1993

[89] Schruben, L.: *Graphical Simulation Modeling and Analysis Using Sigma for Windows.* - Boyd and Fraser Publishing Company, Danvers, 1995

[90] Schwinn, Hans: *Relationale Datenbanksysteme.* - München; Wien: Hanser, 1992

[91] Wilfried Sihn, Tom-David Graupner, Timm Kuhlmann, Hendrik Richter: *Internetbasierte Konfiguration und Simulation von Produktionssystemen.* - Magdeburg, SimVis - Konferenz für Simulation und Visualisierung, 2002

[92] James Snell; Doug Tidwell; Pavel Kulchenko: *Webservice-Programmierung mit SOAP.* - Köln, O'Reilly - Verlag, 2002

[93] SOAP: http://www.w3.org/TR/soap12, 08.2005

[94] W3C: *SRML.* - http://www.w3.org/TR/SRML, 06.2005

[95] Straßburger, S., T. Schulze: *Verteilte- und Web-basierte Simulation: Gemeinsamkeiten und Unterschiede.* - Proceedings of the 15th Simulation Simposium ASIM 2001, Paderborn, 2001

[96] Helmut Strohmeier: *Die Architektur erfolgreicher Projekte: Objekte und agile Strukturen statt Aktivitäten und Phasen.* - Wien: Hanser, 2003

[97] Yu-Hui Tao and Shin-Ming Guo: *The Design of a Web-Based Training System for Simulation Analysis.* - Proceedings of the 2001 Winter Simulation Conference, 2001

[98] Tomcat: http://jakarta.apache.org/tomcat, 08.2005

[99] Troch I., Breitenecker F., Rattay F.: *Modellbildung und Simulation.* Arbeitsunterlagen zur Vorlesung. - Wien, 1995

[100] UDDI: http://www.oasis-open.org/committees/ uddi-spec/doc/spec/v3/uddi-v3.0.2-20041019.htm, 08.2005

[101] Hans Vangheluwe (McGill University) and Juan de Lara (Universidad Autnoma de Madrid): *Meta-Models are Models Too.* - In Proceedings of the 2002 Winter Simulation Conference, 2002

[102] VDI Richtlinie 3633 Blatt 1: *Simulation von Logistik-, Materialfluss - und Produktionssystemen - Grundlagen,* - Düsseldorf 12/1993; VDI Verlag

[103] VDI Verlag: *Software-Engineering in der industriellen Praxis.* - Tagung Düsseldorf, 28. Februar und 1. März 2002 / VDI Kompetenzfeld Informationstechnik. - Düsseldorf: VDI-Verl., 2002

[104] Vienna Grid Environment: http://www.par.univie.ac.at/project/vge/, 08.2005

[105] Walsh, Aaron: *Die Java-Bibel.* - Bonn: MITP-Verl., 1998

[106] Weller, Lucien: *Design und Implementierung einer objektorientierten Umgebung für diskrete Simulation.* Diplomarbeit an der Universität Freiburg, 2001

[107] Thomas Wiedemann: *Simulation Application Service Providing (SIM-ASP).* - In Proceedings of the 2001 Winter Simulation Conference, 2001

[108] Thomas Wiedemann: *Next Generation Simulation Environments Founded on Open Source Software and XML-Based Standard Interfaces.* - In Proceedings of the 2002 Winter Simulation Conference, 2002

[109] Thomas Wiedemann: *Systemanalyse bei der Entwicklung von Simulationsmodellen.* - Dresden, Vorlesungsreihe Simulation betrieblicher Prozesse, 2002

[110] Thomas Wiedmann: *Aktuelle Entwicklungstendenzen in der Modellierung und Simulation.* - Dresden, Vorlesungsreihe Simulation betrieblicher Prozesse, 2003

[111] Web Services Description Language (WSDL) 1.1: http://www.w3.org/TR/wsdl, 08.2005

[112] Web Service Level Agreement (WSLA) Language Specifi- cation: http://www.research.ibm.com/wsla/, 08.2005

[113] Web Service Spezifikation - http://www-106.ibm.com/developerworks/views/webservices/standards.jsp, 08.2005

[114] 6. Foster I. et al.: *Modeling stateful resources with web services, Whitepaper,.* - http://www.globus.org/wsrf, 08.2005

[115] Web Services Resource Framework (WSRF) Technical Committee - OASIS: http://www.oasis-open.org/committees/wsrf, 08.2005

[116] Web Service Security: *SOAP Message Security 1.0.* - OASIS Standard 200401, March 2004

[117] Web Service Security for Java - WSS4J: http://ws.apache.org/wss-fx/wss4j/, 08.2005

[118] XERCES, XALAN: http://xml.apache.org

[119] XML: http://w3c.org, 08.2005

[120] XML.com: *Web Services Resource Center.* - http://www.xml.com/webservices/, 08.2005

[121] XML Encryption: http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/, 08.2005

[122] XML Schema: http://www.w3.org/TR/xmlschema-0/, http://www.w3.org/TR/xmlschema-1/, http://www.w3.org/TR/xmlschema-2/, 08.2005

[123] XML Signature: http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/, 08.2005

[124] XMLSpy: http://www.altova.com/, 08.2005

[125] XQuery: http://www.w3.org/TR/2005/WD-xquery-20050404/, 08.2005

[126] XPath: http://www.w3.org/TR/1999/REC-xpath-19991116, 08.2005

[127] XPointer: http://www.w3.org/TR/2002/WD-xptr-20020816/, 08.2005

[128] XSLT: http://www.w3.org/TR/xslt, 08.2005

[129] Zhai Lili: *Objektmodellierung mit XML und Java.* - Wien, Diplomarbeit der TU Wien, 2000

[130] Zhang Liang-Jie, Mario Jeckle: *The Next Big Thing: Web Services Collaboration.* - International Conference on Web Services 2003, 2003

Über den Autor …

Michael Gyimesi studierte Technische Mathematik an der Technischen Universität Wien. Bereits während des Diplomstudiums befasste er sich neben der Analyse dynamischer Systeme mit Discrete Event Simulation und veröffentlichte unter anderem in Kooperation mit der ARC Seibersdorf GmbH wissenschaftliche Beiträge in einschlägigen Zeitschriften.
Während des Doktoratsstudiums arbeitete er über Grundlagen der Discrete Event Simulation und alternative Modellierungsansätze.
Ein Teil der Ergebnisse ist in dieser Ausgabe der Fortschrittsberichte Simulation veröffentlicht.
Nach der Dissertation arbeitet der Autor am Institut für Wirtschaftswissenschaften/ Abteilung für Operations Research und Dynamische Systeme an der TU Wien weiter.

Über diesen Band …

In dieser Arbeit soll gezeigt werden, wie die Entwicklungen moderner Internettechnologien, im Speziellen XML und darauf aufbauend Web Services im Bereich der Discrete Event Simulation (DES) genutzt werden können, um neue Strömungen der DES zu unterstützen bzw. auszubauen.
Neben einer Einführung in den Bereich der Web Sevices und Grid Services samt zugehöriger Technologien wird eine generische Beschreibung von DES-Modellen präsentiert, die eine simulationssprachenunabhängige Darstellung dieser Modelle erlaubt.
Außerdem wird ein für diese Arbeit entwickeltes Framework beschrieben, das das Anbieten von Simulation als Application Service ermöglicht.

Über diese Reihe …

Die Bände dieser neuen ASIM - Reihe Fortschrittsberichte Simulation konzentrieren sich auf neueste Lösungsansätze, Methoden und Anwendungen der Simulationstechnik (Ingenieurwissenschaften, Naturwissenschaften, Medizin, Ökonomie, Ökologie, Soziologie, etc.).
ASIM, die deutschsprachige Simulationsvereinigung (Fachausschuss 4.5 der GI - Gesellschaft für Informatik) hat diese Reihe ins Leben gerufen, um ein rasches und kostengünstiges Publikationsmedium für derartige neue Entwicklungen in der Simulationstechnik anbieten zu können.
Die Fortschrittsberichte Simulation veröffentlichen daher: * Monographien mit speziellem Charakter, wie z. B. Dissertationen und Habilitationen * Berichte zu Workshops (mit referierten Beiträgen) * Berichte von Forschungsprojekten * Handbücher zu Simulationswerkzeugen (User Guides, Vergleiche, Benchmarks), und Ähnliches.
Die Kooperation mit den ARGESIM Reports der ARGESIM vermittelt dabei zum europäischen Umfeld und zur internationalen Publikation.