Institut für Architektur von Anwendungssystemen

Universität Stuttgart
Universitätsstraße 38
D - 70569 Stuttgart

Diplomarbeit Nr. 3264

Simulation des Verhaltens von Zellkomponenten in biologischen Netzwerken mit Hilfe von Workflow Technologie

Yue Zou

Studiengang:	Informatik
Prüfer:	Jun.-Prof. Dr.-Ing. Dimka Karastoyanova
Betreuer:	Dipl.-Math. Michael Reiter
begonnen am:	21. Oktober 2011
beendet am:	07. Mai 2012
CR-Klassifikation:	H.3.5, H.3.4, I.6.3, I.6.7

Inhaltsverzeichnis

1 Einleitung

In einer zunehmend globalisierten Welt, in der die Bereiche Wirtschaft und Wissenschaft zusammenwachsen, sorgt die effektive und effiziente Arbeitsweise dafür, die qualitativ besseren Produkte in kürzerer Zeit zu entwickeln und gleichzeitig die menschliche Arbeit zu erleichtern. Durch die Globalisierung können die in der Komplexität gewachsenen Aufgaben nicht mehr zentral an einem Standort erledigt werden, sondern müssen in verschiedene Arbeitsabläufe aufgegliedert und dezentral auf lokalen Standorten verteilt werden. Dafür spielen das Koordinieren und das Verwalten der Workflows eine zentrale Rolle. Die Workflowsysteme, in denen die verschiedenen Workflows gesteuert und ausgeführt werden, werden durch Rechnerunterstützung optimal strukturiert, mit dem Ziel intelligente und automatisierte Arbeitsabläufe zu realisieren.

Die Workflow-Technologie wird auch im wissenschaftlichen Bereich, beispielsweise für die hochkomplexen Simulationen, die hohen Zeitaufwand und hohe Rechenkapazität benötigen, verwendet und weiterentwickelt. Deshalb wurden Scientific Workflows einführt, mit denen die während der Simulation und Berechnung entstehenden Daten, zusammenführend analysiert werden können [6]. Für eine optimale Zusammenführung und Verteilung von Daten und Rechenkapazitäten ist eine sinnvolle Infrastruktur notwendig, die benutzerfreundlich ist. Solche Infrastruktur kann auf Basis der serviceorientierten Architektur (SOA) realisiert werden.

Die Hauptbestandteile der SOA sind Dienstverzeichnis, Dienstanbieter und Dienstnutzer, die ein sogenanntes SOA Dreieck bilden. Auf der Basis des SOA-Dreieckkonzeptes wurde ein Web Service entwickelt, der die Zusammenarbeit der auf unterschiedlichen Plattformen betriebenen Anwendungsprogramme unterstützt. In dieser Arbeit wird eine Anwendung auf Basis einer Octave[1]-basierten Simulation für ein biologisches Reaktionsnetzwerk betrachtet. Für diese Anwendungen wird ein Web Service mit der XML-basierten Web Service Technologie erstellt. Darüber hinaus wird ein entsprechender Workflow aufgebaut, der durch eine deskriptive Sprache BPEL (Web Services Business Process Execution Language) [6] die Ausführungsreihenfolge von den erstellten Web Services bestimmt. Dadurch wird eine automatisierte Ausführung der Simulationsanwendung ermöglicht.

[1] GNU Octave: http://www.gnu.org/software/octave/

Diese Arbeit konzentriert sich auf die Bereiche Workflow- und Web Service Technologie sowie biologische Reaktionsnetzwerke und ist interdisziplinär sowie anwendungsorientiert ausgelegt. Sie ermöglicht eine Zusammenarbeit von Wissenschaftlern an verschiedenen Orten und auf weltweit verteilten Ressourcen. Dies kann unter dem Begriff e-Science zusammengefasst werden. In der Arbeit wird eine computergestützte und modellbasierte Simulation über Parameteridentifikation im Reaktionsnetzwerk dargestellt.

Eine interdisziplinäre Zusammenarbeit findet im Rahmen des Projektes „Exzellenzclusters Simtech[2]“ statt. Die in dieser Arbeit erstellten Methoden dienen als Grundlagen für die Ausführung von systembiologischen Simulationen mit Hilfe der Workflow- und Web Service-Technologie. Dadurch können die Benutzer mit weniger Programmierhintergrund als bislang komplexe Simulationen einfach durchführen.

1.1. Motivation und Aufgabenstellung

Im Rahmen dieser Arbeit wird eine automatisierte Ausführung von Octave-basierten Simulationen mit Hilfe von Workflow- und Web Service Technologie vorgestellt.

Auf Basis von einem Web Service Interface [23] soll ein Web Service Plugin erstellt werden, über den die Simulationsanwendungen angesprochen werden können. Zum Aufbau des Web Service Plugins gehört die Bereitstellung eines Web Services für die Octave-Anwendungen. Dabei wird die Web Services Description Language (WSDL) [3] benötigt, um die Web Service-Schnittstellen zu beschreiben.

Zur Ausführung der Web Service-basierten Simulation werden Java-Klassen verwendet, die aus WSDL-Operationen generiert und weiter ergänzt werden können. Die Ergebnisdateien der Simulation sollen in einer Datenbank gespeichert und zu einem anderen Zweck weiter verwendet werden. In der Simulation wird eine graphische Darstellung als Ergebnis erzeugt, dafür muss eine Schnittstelle für die Visualisierung unterstützt werden.

Weiterhin soll ein BPEL-Prozess erstellt werden, der dem Ablauf der Simulation entspricht. Die einzelne Aktivität des Prozesses setzt sich aus den entsprechenden WSDL-Operationen zusammen, mit deren Hilfe eine Web Service-Schnittstelle aufgerufen und die Simulationsanwendung ausgeführt wird.

Es soll eine Laufzeitumgebung erstellt werden, in der die Web Service-Schnittstelle, die Simulationsanwendungen sowie die Workflow-Umgebung zusammen ausgeführt werden können.

[2] Exzellenzclusters Simtech : http://www.simtech.uni-stuttgart.de/

1.2. Aufbau der Arbeit

Die vorliegende Arbeit gliedert sich wie folgt in neun Kapitel:

Kapitel 2 – Grundlagen: In diesem Kapitel findet sich zunächst eine kurz Vorstellungen von SOA und Web Service. Danach folgt eine Beschreibung der Workflow-Technologie. Bei dem Abschluss des Kapitels wird der Begriff e-Science erläutert.

Kapitel 3 – Verwendete Software: Sowohl die Simulationsprogramm als auch die Softwares in der Workflow- und Web Service-Umgebung werden in diesem Kapitel vorgestellt.

Kapitel 4 – Reaktionsnetzwerke: Dieses Kapitel bezieht sich auf den Hintergrund der Simulationen. Die Parameteridentifikation, die für die Simulation relevant ist, wird zum Schluss des Kapitels beschrieben.

Kapitel 5 – Spezifikation: Die Anforderungen an den Octave Web Service werden in diesem Kapitel vorgestellt. Die Idee zum Erstellen eines Web Service Plugin für die Octave-Anwendungen (auch Octave Adapter genannt) wird beschrieben.

Kapitel 6 – Entwurf: Kapitel sechs zeigt zunächst die Architektur des Octave Adapters. Die Parameter der einzelnen Web Service Operationen werden erklärt. Schließlich wird ein Octave-basierter Workflow-Prozess dargestellt.

Kapitel 7 – Implementierung: Der Anfang des Kapitels behandelt den ursprünglichen und den modifizierten Simulationsablauf. Dann werden die weiteren Möglichkeiten der Implementierungen vorgestellt.

Kapitel 8 – Laufzeitumgebung: Eine spezielle Laufzeitumgebung wird dargestellt, die von der Web Service-Schnittstelle sowie der Workflow-Prozess genutzt wird.

Kapitel 9 – Zusammenfassung und Ausblick: Abschließend erfolgen eine Zusammenfassung der Ergebnisse der Arbeit und ein Ausblick auf mögliche Erweiterungen.

2. Grundlagen

In diesem Kapitel werden die fachlichen Grundlagen der Arbeit dargestellt. Ein Überblick über SOA und Web Services wird zunächst gegeben (Abschnitt 2.1). Danach folgt eine kurz Einführung in Workflow, Scientific Workflow, Workflow Management Systeme sowie Workflow-Sprache (Abschnitt 2.2). Zum Abschluss werden das e-Science und eine Simulation in Reaktionsnetzwerke erläutert (Abschnitt 2.3)

2.1. SOA und Web Services

Service-orientierte Architekturen (SOA) und Web Service (WS) weisen zahlreiche Zusammenhänge auf, da der Web Service und eine Vielzahl der damit in Verbindung stehenden Spezifikationen eine mögliche Implementierungstechnologie bieten, um die Anforderungen einer SOA zu erfüllen. Aus diesen Gründen ist es sinnvoll auf beide Themen einzugehen.

2.1.1. Service-orientierte Architektur (SOA)

Da eine einheitliche Definition einer SOA nicht existiert, wird im Folgenden auf aufgrund ihrer Generalität eine Definition nach Dostal et al. [1] angegeben:

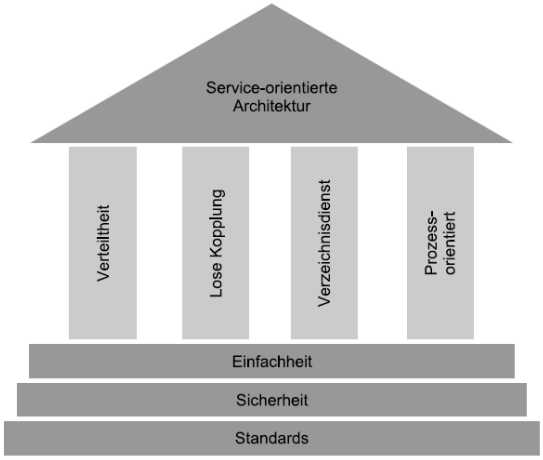


Abbildung. 1 Grundlegende Merkmale einer SOA [1]

Unter einer SOA versteht man eine Systemarchitektur, die vielfältige, verschiedene und eventuell inkompatible Methoden oder Applikationen als wiederverwendbare und offen zugreifbare Dienste repräsentiert und dadurch eine Plattform und sprachenunabhängige Nutzung und Wiederverwendung ermöglicht.

Abbildung. 2 verdeutlicht, dass das Fundament der SOA aus offenen Standards, Sicherheit und Einfachheit besteht. Die tragenden Säulen stellen die verteilten Dienste, die lose Kopplung, der Verzeichnisdienst und die prozessorientierte Struktur dar.

Wie der Name bereits aufzeigt, liegt der Dienst (engl. Service) einer SOA im Mittelpunkt. Um den Dienst eines unbekannten Anbieters verstehen zu können, müssen alle Schnittstellen des Dienstes in maschinenlesbarer Form beschrieben vorliegen. Dazu werden die offenen Standards genutzt. Die Einfachheit einer SOA bedeutet, dass die Nutzung der Dienste es ermöglicht, diese in verschiedenen Umgebungen mehrfach ohne Aufwand wiederzuverwenden. Eine weitere Voraussetzung für eine SOA ist die Sicherheit, die von Anfang an beachtet werden sollte. Bei der losen Kopplung (loose coupling) werden die Dienste von Anwendungen oder anderen Diensten bei Bedarf dynamisch gesucht, gefunden und eingebunden [1]. Damit Funktionalitäten dynamisch eingebunden werden können, müssen die gewünschten Dienste zunächst ausfindig gemacht werden, was mittels eines Verzeichnisdienstes erfolgen kann, in dem zur Verfügung stehende Dienste registriert werden.

Unter den Beteiligten der SOA bestehen darüber hinaus drei weitere Rollen: Dienstverzeichnis, Dienstanbieter, Dienstnutzer. Der Zusammenhang zwischen diesen Rollen lässt sich in einem SOA-Dreieck darstellen. Abbildung. 1 stellt dem Dienstanbieter eine Plattform zur Verfügung, welche über ein Netzwerk Zugriff auf mindestens einen Dienst ermöglicht. Um die Dienste von Nutzern finden zu können, registriert der Dienstanbieter seine Dienste unter einem Verzeichnisdienst. Das Finden von Diensten erfolgt mittels eines Dienstverzeichnisses. Der Dienstnutzer greift über die veröffentlichten Schnittstellen auf die Dienste eines Dienstanbieters zu, um nach den passenden Diensten zu suchen.

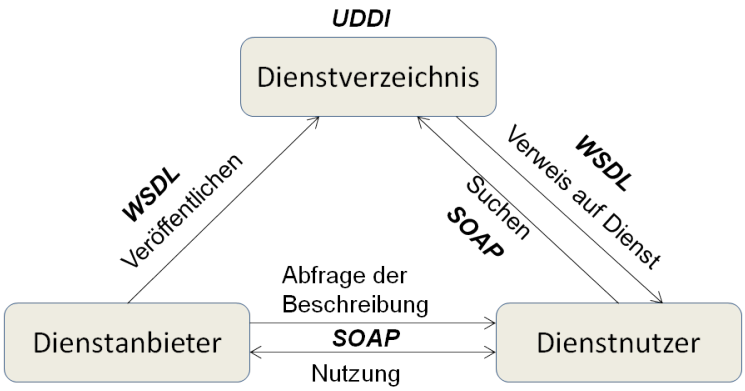


Abbildung. 2 Das Dreieck einer SOA [1]

2.1.2. Web Service

Ähnlich wie SOA existiert keine verbindliche Definition bezüglich Web Services. Jedoch sind diese durch eine ständige Weiterentwicklung, und Verfeinerung von Standards gekennzeichnet. Die Autoren Dostal et al. [2] werten die Entwicklung der Definition von WS des W3C (World Wide Web Consortium), die im Allgemeinen und Abstrakten folgende Definition liefert:

> „A Web Service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web Service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.“ (August 2003)

Dabei werden die SOA-Komponenten Kommunikation, Dienstbeschreibung und Verzeichnisdienst mit den Web Service Standards SOAP (engl. *Simple Object Access Protocol*), WSDL (engl. *Web Service Description Language*) und UDDI (engl. *Universal Description, Discovery and Integration*) konkretisiert.

Abbildung. 2 stellt das so genannte SOA-Dreieck dar. Wenn ein Dienstanbieter einen Dienst bereitstellen möchte, erstellt dieser zunächst eine Schnittstellenbeschreibung seines Dienstes mittels eines XML-basierten WSDL-Dokuments, welches sich in einem UDDI-basierten Verzeichnisdienst veröffentlichen lässt. Ein Dienstnutzer kann im Verzeichnisdienst nach passenden Diensten suchen, und erhält eine URI-Referenz auf die WSDL-Datei des gewünschten Web Service als Suchergebnis. Diese WSDL-Beschreibung kann zum Einbinden des gewünschten Web Service innerhalb der gewünschten Anwendung genutzt werden. Letztlich kommuniziert der Dienstnutzer über das Protokoll SOAP mit dem Dienstanbieter.

Dienstbeschreibung mittels WSDL

Zur Dienstbeschreibung wird die WSDL eingesetzt, die einen Dienst syntaktisch oder strukturell in Form eines XML-Dokuments beschreibt. Zwar bestehen zwei Versionen der WSDL, wobei die Version 1.0 bislang über die größte Unterstützung an Software-Tools verfügt.

Die WSDL-Spezifikation besteht aus einem abstrakten und einem konkreten Teil (siehe Abbildung. 3). Der abstrakte Teil beschreibt die Funktionalitäten eines Web Services. Innerhalb des Elements <wsdl:types> werden die Datentypen definiert. <wsdl:message> stellt dar, welche Nachrichten zwischen Dienstnutzer und Dienstanbieter ausgetauscht werden. Die Operationen, die dem Dienstnutzer zur Verfügung gestellt werden, befinden sich in <wsdl:portType>.

Im konkreten Teil der WSDL-Datei stehen die Informationen zu binding, und service. Mittels des Elements <wsdl:binding> wird das Nachrichtenformat definiert und dem abstrakten PortType ein konkretes Transportprotokoll (z.B. SOAP, HTTP) zugeordnet. Zur Modellierung

von SOAP Interaktionstypen, kann bei der Bindung an das SOAP-Protokoll zwischen Document-Style und RPC-Style ausgewählt werden. Im Element <wsdl:service> existieren ein oder mehrere Elemente <wsdl:port>, die die Endpunkte darstellen. Diese führen die Bindings mit konkreten Netzwerkadressen (URLs) zusammen, unter denen die Implementation eines abstrakten PortTypes zu erreichen ist.

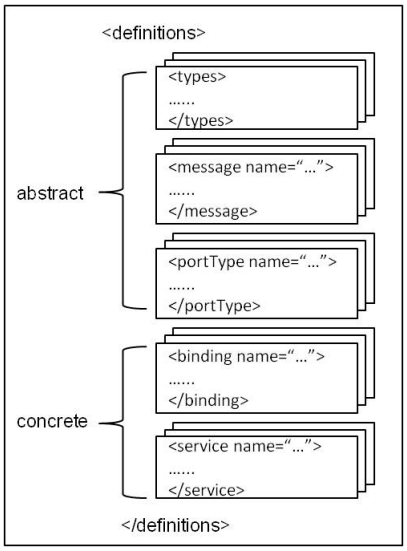


Abbildung. 3 Syntaktische Struktur eines WSDL-Dokuments [9]

Kommunikation mittels SOAP

Bei SOAP handelt es sich um „ ein XML-basiertes und sprach- und plattformunabhängiges Kommunikationsprotokoll zum Austausch strukturierter Informationen.“ [2] Zur Kommunikation senden und empfangen Web Services SOAP-Nachrichten. Eine SOAP-Nachricht besteht aus bis zu drei Teilen (siehe Abbildung. 4): dem SOAP-Envelope als Root-Element der SOAP-Nachricht, einem optionalen SOAP-Header und dem SOAP-Body, der die eigentlichen Nutzdaten (engl. payload) enthält.

Eine SOAP-Nachricht kann entweder direkt vom Sender zum Empfänger übermittelt oder über mehrere Zwischenstationen (engl. *intermediaries*) übertragen werden; jede Station in der Übertragungskette (engl. *message path*) ist ein Knoten (engl. *node*). Wenn eine Nachricht von einem Kommunikationspartner nicht verarbeitet werden kann, wird ein SOAP-Fehler (engl. *SOAP fault*) als Antwortnachricht versendet.

SOAP schreibt nicht vor, mit welchem Transportprotokoll eine Nachricht übertragen werden muss, sondern wählt ein passendes Protokoll aus. „Wird allerdings ein gewisses Maß an Übertragungssicherheit (ähnlich einem Einschreiben) benötigt, dann sollte eher ein Messaging-System wie WebSphereMQ gewählt werden. Aufgrund seiner Herkunft aus dem Internet-Umfeld ist das zurzeit am häufigsten genutzte Transportprotokoll für SOAP-Nachrichten natürlich HTTP.“[2]

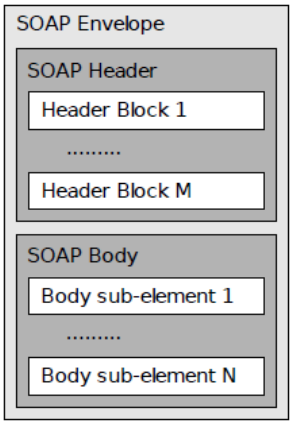


Abbildung. 4 Aufbau von SOAP-Nachrichten [9]

WS-Addressing

In den klassischen Web-Service-Mechanismen werden die Endpunkte lediglich mittels einer URI in der WSDL-Datei identifiziert, wodurch keine Möglichkeit besteht, zusätzliche Metadaten auf standardisierte Art und Weise zum Bestandteil der Angaben über den Endpunkt zu machen.

Das Web-Addressing liefert u.a. die Möglichkeit, die Endpunkte als XML-Dokument zu formulieren und weitere Metadaten (z.B. Policy-Informationen) zu kodieren. Dazu führt es noch zwei neue Konzepte ein: Die Endpunkt-Referenz(engl. *endpoint reference*) sowie zusätzliche Nachrichten-Adressierungseigenschaften (engl. *message addressing properties*).

Eine Endpunkt-Referenz enthält alle notwendigen Informationen für die Webservice-Kommunikation. Für den jeweiligen Endpunkt können zusätzliche Metadaten als Referenz-Parameter (engl. reference parameters) im Dokument formuliert, und bei der Nachrichtenübertragung auf eigenständige SOAP-Header-Elemente abgebildet werden.

Neben den Endpunkt-Referenzen bestehen zudem die von WS-Addressing definierten zusätzlichen Nachrichten-Adressierungseigenschaften, die diverse Angaben für die SOAP-Interaktion bezüglich Routing und Kommunikationspfad umfassen. Alle Angaben befinden sich in den SOAP-Header-Elementen.

Listing 1 stellt ein Beispiel einer SOAP-Nachricht mit eingebetteten WS-Addressing-Informationen dar, wobei die Angabe „To“ die Ziel-Adresse des Empfängers als URI enthält. „MessageID“ identifiziert die Nachrichten in Zeit und Raum. „ReplayTo“ definiert den Antwort-Endpunkt als Endpunkt-Referenz und „Action“ enthält die angedachte Service-Aktivität, welche typischerweise an ein Element aus der zugehörigen WSDL-Datei gebunden wird.

```
<S:Envelope xmlns:S=http://www.w3.org/2003/05/soap-envelope
            xmlns:wsa="http://www.w3.org/2005/08/addressing"
            xmlns:example="...">
   <S:Header>
      <wsa:MessageID> uuid:6B29FC40-CA47-1234-ABCD-00DD010662DA
      </wsa:MessageID>
      <example:subsidiary> foo street </example:subsidiary>
      <example:session> 42bdjhd8hw </example:session>
      <wsa:ReplyTo>
      <wsa:Address>http://example.org/customerNotify</wsa:Addre
ss></wsa:ReplyTo>
      <wsa:To>http://example.com/Purchasing</wsa:To>
      <wsa:Action>http://example.com/SubmitOrder</wsa:Action>
    </S:Header>
   <S:Body>
   ...
   </S:Body>
</S:Envelope>
```

Listing 1. Beispiel für SOAP 1.2-Nachricht mit WS-Addressing-Informationen [1]

Verzeichnisdienst

Der Verzeichnisdienst bildet ein weiteres zentrales Bestandteil einer SOA. Mittels standardisierter Schnittstellen wird eine lose Kopplung von Anwendungen ermöglicht. Die „Universal Description, Discovery and Integration“(UDDI) zählt zu einem der grundsätzlich verschiedenen Spezifikationen für Verzeichnisdienste.

Innerhalb eines UDDI-Verzeichnisses kann ein Dienst mit einer Dienstbeschreibung (z.B. in Form eines WSDL-Dokuments) in einer Datenbank verwahrt werden, damit ein potentieller Nutzer diesen Dienst und eine Anleitung zu dessen Nutzung finden kann.

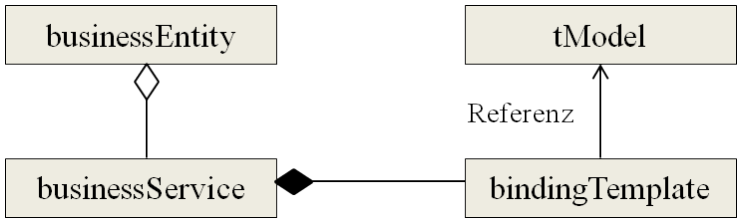


Abbildung. 5 UDDI-Datenmodell [2]

Das UDDI-XML-Schema und die UDDI-API sind zwei große Komponenten von UDDI. Beim UDDI-XML-Schema werden vier Hauptentitäten unterschieden. In der *businessEntity* stehen Informationen bezüglich Organisationen oder Unternehmen durch Serviceanbieter zur Verfügung. Im Teil BusinessService werden die von *businessEntity* bereitgestellten Dienste zusammengefasst, die jeweils weiter an verschiedene Übertragungsprotokolle gebunden werden. Das *bindingTemplate* liefert sowohl technische Informationen zur Nutzung eines Dienstes als auch eine Adresse, unter der ein Dienst aufgerufen werden kann. Ein oder mehrere Dokumente werden durch das *bindingTemplate* zusammengefasst, welche eindeutige technische Beschreibungen zum Dienst beinhalten (das sog. *tModel*).

2.2. Workflow

Das vorliegende Unterkapitel soll zunächst einen Einblick in die Workflows geben, um anschließend genauer auf Scientific Workflows und Workflow Management Systeme eingehen zu können. Zum Schluss erfolgt die Beschreibung der Business Process Execution Language (BPEL).

2.2.1. Geschäftsprozesse und Workflows

Die Begriffe Geschäftsprozess (kurz genannt Prozess) und Workflow liegen sehr eng zusammen, jedoch können je nach Sichtweise die Begriffe unterschieden werden. In Abbildung 6 wird der Prozess in der reellen Welt durch Prozessmodelle dargestellt, während der Workflow einen Prozess technisch unterstützt.

Ein Prozess wird durch die Modellierung eines Verlaufs festgelegt, mit dem Leistungen oder Informationen transportiert werden. Dieser Verlauf wird als ein Prozessmodell bezeichnet. Nach [8] enthält ein Prozessmodell alle Aktivitäten eines Prozesses sowie die Pfade zwischen diesen Aktivitäten und dient zudem als eine Vorlage, die ein Prozess initiiert. Aus dem Prozessmodell entsteht durch die Ausführung eine sogenannte Prozessinstanz.

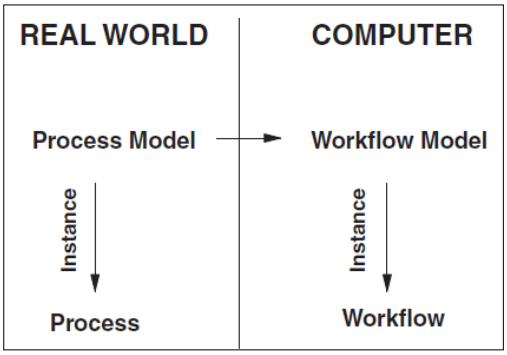


Abbildung. 6 Geschäftsprozess und workflow [8]

Prozesse können nicht nur im Alltag von Personen durchgeführt, sondern auch auf Rechnern ausgeführt werden. Dieser Anteil eines Prozesses bezüglich der Rechner führt zum Begriff des Workflow-Modells. Analog zum Prozessmodell kann das Workflow-Modell zur Erstellung eines Workflows verwendet werden.

Da alle Prozesse der vorliegenden Arbeit lediglich auf Rechnern ausgeführt wurden, besteht kein Unterschied zwischen den Begriffen Prozessmodell und Workflow-Modell, sowie Prozessinstanz und Workflow-Instanz. Daher werden diese im Folgenden jeweils als Synonym verwendet.

Workflow Dimensionen

Nach [8] werden Workflows mittels drei verschiedener Dimensionen dargestellt, die in einem Würfel graphisch gezeichnet werden.

What

In dieser Dimension spiegelt sich die Geschäftslogik eines Workflows wieder und beschreibt zugleich, welche Aktivitäten wie in dem Prozess ausgeführt werden müssen. Zwischen den Aktivitäten stehen die Control Flows, die sowohl sequentiell als auch parallel definiert werden können.

Who

Diese Dimension spezifiziert die Organisationsstruktur eines Unternehmens in Hinblick auf Abteilungen, Rollen und Personen. Verwendet werden all diese Informationen, um zu beschreiben, wer die Aktivitäten ausführen kann. Dazu kann eine entsprechende Query erstellt werden, um die Durchführung der Aktivitäten durch passende Personen zu koordinieren.

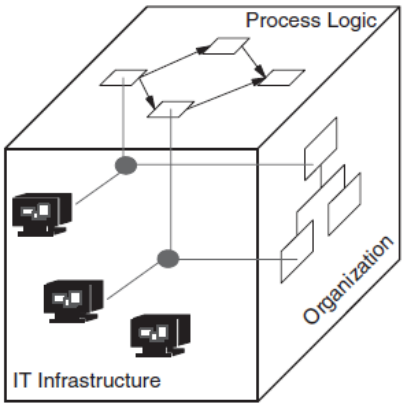


Abbildung. 7 Drei Workflow Dimensionen [8]

With

Diese Dimension bestimmt IT-Ressourcen (Anwendungen, Hardware usw.), die für die Ausführung der Aktivitäten notwendig sind.

2.2.2. Scientific Workflows

In den letzten Jahren schenkte man nicht nur im Geschäftsbereich den Workflows mehr Beachtung, sondern auch im wissenschaftlichen Bereich. Aufgrund von Berechnungen großer komplexer und heterogener Datenmengen konnte auf Basis von Workflows eine Technologie geschaffen werden, mit dessen Hilfe derartige Berechnungen automatisiert durchgeführt werden können. Entsprechend wurde der Begriff Scientific Workflow eingeführt.

Scientific Workflows werden verwendet, um beispielsweise Simulationen in der Wissenschaft zu beschreiben und auszuführen. Solche Simulationen sind oftmals lang laufend, was die Aufteilung einer Aufgabe in verschiedene kleine Aufgaben erfordert. Die Ergebnisse können in einem Repository zwischengespeichert oder als Eingabe für den nächsten Schritt weiter verwendet werden.

Zu den positiven Eigenschaften von Scientific Workflows zählen beispielsweise, der nahtlose Zugriff auf Ressourcen und Services sowie die Komposition auf Basis von wiederverwendbaren Workflow-Fragmenten. Einige Workflow Management Systeme unterstützen lang laufende Workflows durch den asynchronen Aufruf von Services. Dies gestattet der Workflow-Engine im Hintergrund zu laufen, ohne ständigen Kontakt zu den Services haben zu müssen. Weitere Eigenschaften von Scientific Workflows werden in [10] aufgezeigt.

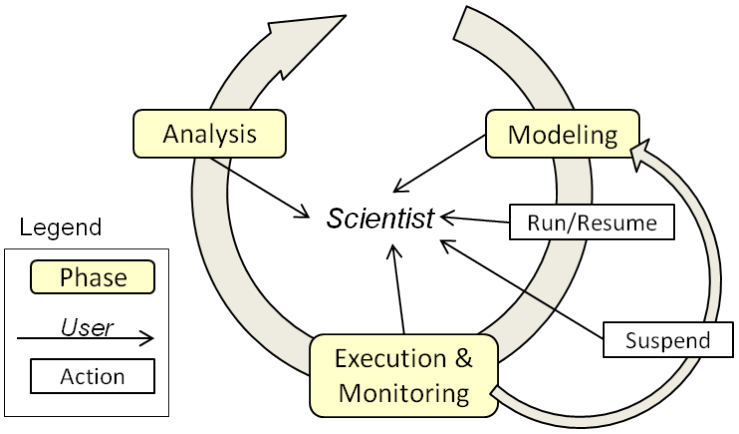


Abbildung. 8 Lebenszyklus eines Scientific Workflows [11]

Abbildung. 8 präsentiert ein Lebenszyklus von Scientific Workflows. Auf den ersten Blick

sieht es so aus, dass in der ersten Phase die Modellierung des Workflows erfolgt. Anschließend werden diese Workflows ausgeführt, und die Informationen der Prozessinstanzen durch eine Monitoring-Komponente gesammelt und bearbeitet. Schließlich werden die Workflows nach der Ausführung analysiert, wobei zum Beispiel die Qualität der gelieferten Daten überprüft oder nach Fehlern gesucht werden kann. Dadurch können notwendige Änderungen des Modells erkannt und umgesetzt werden.

Nach [11] unterliegen die Phasen der Modellierung und Ausführung in der Tat nicht einer strengen Reihenfolge, da Wissenschaftler normalerweise die Scientific Workflows selbstständig erstellen und sich die Workflows sehr häufig ändern können. In der Regel erfolgt die Erstellung von Scientific Workflows mittels eines Trial-and-Error Verfahrens. Durch die in [11] beschriebene Suspend-Aktion kann die Ausführungsphase erneut zurück zur Modellierungsphase geführt werden. Ausführung und Monitoring werden in eine Phase zusammengelegt, da nach Meinung von Wissenschaftlern das Monitoring häufig nur die Daten einer laufenden Workflow Instanz visualisiert.

2.2.3. Workflow Management Systeme

Für die Analyse, Modellierung, Simulation sowie für die Ausführung und Steuerung von Workflows wird ein Workflow Management System benötigt, welches über die einzelnen organisatorischen Arbeitsschritte und Abläufe verfügt, die dem Lebenszyklus eines Workflow entsprechen. Ein Workflow Management System (WfMS) ist ein System, das das Workflow Management durch IT-Werkzeuge unterstützt. Die *Workflow Management Coalition (WfMC)* [3]wie folgendermaßen definiert:

> *A system that defines, creates and manages the execution of workflows through the use of software, running on one or more workflow engines, which is able to interpret the process definition, interact with workflow participants and, where required, invoke the use of IT tools and applications.*

Das WfWS ist nach [7] in folgende vier Bereiche aufgeteilt:

Buildtime

Dieser Bereich dient zur Modellierung von Workflows. Alle Komponenten werden bereitgestellt, um alle Workflow betreffende Informationen erstellen, testen und verwalten zu können.

Metamodel

Das Metamodel ist gewissermaßen das Modell eines Modells. Es unterstützt die Erstellung von Regeln, um einen Prozess, eine Organisation oder eine benötigte Technologie zu modellieren. Ein Metamodell eignet sich insbesondere zur Darstellung

[3] Workflow Management Coalition :http://www.wfmc.org/

der grundlegenden Elemente eines Konzeptes oder Systems sowie zur Strukturierung von Methoden.

Runtime

Der Runtime-Bereich ist zuständig für die korrekte Ausführung eines modellierten Prozesses. Die Prozesse aus der realen Welt werden auf Basis der Funktionalität des Runtime-Bereichs in eine Computer-ausführbare Sprache übersetzt. Für die Instanziierung und Steuerung der Prozesse sowie für die Interaktion zwischen Anwender und computergestütztem Prozess stellt der Runtime-Bereich alle Komponenten zur Verfügung, die zur Ausführung einer Prozessinstanz benötigt werden.

Datebase

Die Database speichert die kompletten Informationen, die vom Buildtime- und Runtime-Bereich verwendet werden. In diesem Bereich können beliebig viele Datenbanken genutzt werden.

Für weitere Informationen zu den WfMS-Komponenten sei auf das Buch [7] verwiesen.

2.2.4. Business Process Execution Language

In diesem Abschnitt wird die Business Process Execution Language (nachfolgend BPEL genannt) beschrieben, die auf der WS-BPEL2.0 Spezifikation[4] basiert und zudem in dem Buch [9] beschrieben wird.

BPEL ist eine XML-basierte Sprache, mit der ein Prozess beschrieben und abgebildet werden kann. Die einzelnen Aktivitäten des Prozesses werden durch Web Services implementiert. BPEL basiert auf der Kalkül-basierten Sprache XLANG[5] und der Graph-basierten Sprache WSFL[6]. Aktuell stellt BPEL in der Version 2.0 zur Verfügung.

Listing 2 stellt ein Beispiel für den Aufbau eines BPEL-Prozesses dar. Die jeweiligen Bestandteile werden im Folgenden erläutert.

```
<process name="prozessname" >
      <partnerLinks> ... </partnerLinks>
      <partners> ... </partners>
      <variables> ... </variables>
      <correlationSets> ... </correlationSets>
      <faultHandlers> ... </faultHandlers>
      <compensationHandler> ... </compensationHandler>
```

[4] WS-BPEL2.0 Spezifikation: http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html

[5] XLANG: http://msdn.microsoft.com/de-de/library/aa577463.aspx

[6] WSFL: http://www.ibm.com/developerworks/webservices/library/ws-wsfl1/

```
        <eventHandlers> ... </eventHandlers>
        <!—Aktivitäten -->
        </activity>
</process>
```

Listing 2 Beispiel für den Aufbau eines BPEL-Prozesses

Prozess

Nach [9] existieren zwei unterschiedliche Arten von BPEL-Prozessen: abstrakte und ausführbare Prozesse. Erster beschreibt ein Business Protokoll, das die Kommunikation zwischen zwei Partnern abbildet, ohne die dahinter stehende Logik genauer zu spezifizieren. Bei den ausführbaren Prozessen werden die konkrete Implementierung von Prozessen sowie die eigentliche Logik konkret beschrieben.

Aktivität

Aktivitäten aus einem BPEL-Prozess können Web Services aufrufen, Daten manipulieren, Fehler erkennen oder einen Prozess beenden. Man unterscheidet zwischen strukturierten Aktivitäten und Basisaktivitäten.

Basisaktivitäten können nicht zur Gruppierung andere Basisaktivitäten verwendet werden. Ihre Ausführung innerhalb des jeweiligen Prozesses wird als atomare Operation betrachtet. Beispielsweise wird ein Web Service mit der Aktivität `<invoke>` synchron oder asynchron aufgerufen. Eine Aktivität `<receive>` muss stets einer Aktivität `<reply>` vorangegangen sein, während `<receive>` nicht unbedingt durch `<reply>` beantwortet werden muss.

Strukturierende Aktivitäten können andere strukturierte Aktivitäten oder Basisaktivitäten enthalten und daher rekursiv verwendet, beliebig verschachtelt und kombiniert werden. Sie beschreiben eine Kontrollflusslogik zwischen den enthaltenen Aktivitäten. Ein Beispiel hierfür ist die Aktivität `<sequence>`. Diese stellt dar, wie die enthaltenen Aktivitäten in einer angegebenen Reihenfolge ausgeführt werden. Mittels der Aktivität `<while>` können die enthaltenen Aktivitäten mehrfach in einer Schleife laufen, während die Aktivität `<flow>` eine parallele Ausführung realisiert.

Variablen

Mit Variablen können bestimmte Werte in den Prozessen zwischengespeichert werden. Diese Variablen sind nur in den Scopes sichtbar, in welchen sie definiert werden. Zur Steuerung eines Kontrollflusses können sie verwendet werden, um den Prozess zu beeinflussen.

Korrelationsmengen

Mittels Korrelationsmengen (Correlation Sets) können unterschiedliche Instanzen eines

Prozesses voneinander unterschieden werden. Sobald zu einem Zeitpunkt mehrere BPEL-Instanzen vorliegen, werden Korrelationsmengen zur Weiterleitung von Nachrichten an die richtige Instanz verwendet.

Partner Links

Um die in der WSDL-Datei definierten Web Service Schnittstellen innerhalb eines BPEL-Prozesses zu nutzen, werden Partner Links verwendet. In einem Partner Link werden eine oder mehrere Rollen definiert, die jeweils einem `wsdl:portType` zugeordnet sind.

Scopes und Handler

In BPEL kann ein Prozess aus hierarchisch verschachtelten Scopes bestehen. Ein Scope umfasst Aktivitäten, die Definition von Daten, Partner Links, Korrelationsmengen und Handler.

In einem Scope existieren drei verschiedene Handler: Event Handler, Fault Handler und Compensation Handler. Beispielsweise wird ein Event Handler dazu verwendet, im synchronen Fall den Prozess weiter zu verarbeiten, nachdem ein entsprechendes Ereignis eingetroffen ist. Sollten bei der Ausführung eines BPEL-Prozesses Fehler auftreten, können diese mit Hilfe von Fault Handler behandelt werden. Compensation Handler werden aufgerufen, wenn die dazugehörige Scope rückgängig gemacht werden soll.

2.3. e-Science

Heutzutage arbeiten Wissenschaftler zunehmend vernetzt und interdisziplinär an verschiedenen Orten und auf weltweit verteilten Ressourcen. Laut Ball [12] wird unter dem Begriff e-Science (Abkürzung von Enhanced Science) zusammengefasst, dass neue Plattformen für netzbasierte kooperative Forschung und Zusammenarbeit im Wissenschaftsbereich entstehen. E-Science beschreibt eine wissenschaftliche Arbeitsumgebung, die den Austausch von Ressourcen, die Zusammenarbeit in verteilten Teams und optimierte Kommunikationsprozesse umfasst. Oftmals wird es im Zusammenhang mit rechenintensiven Anwendungen in verteilten Systemen verwendet, wie beispielsweise im Umfeld von Grid-Computing. Zudem werden die Bereiche Wissensvernetzung, Open Access und e-Learning in e-Science gefördert.

Allerdings muss e-Science in einem weiteren Sinn verstanden werden. Das britische National e-Science Centre (NESC)[7] hat dies folgendermaßen zusammengefasst:

> In the future, e-Science will refer to the large scale science that will increasingly be carried out through distributed global collaborations enabled by the Internet. Typically,

[7] National e-Science: http://www.nesc.ac.uk

> a feature of such collaborative scientific enterprises is that they will require access to very large data collections, very large scale computing resources and high performance visualization back to the individual user scientists.

In vielen wissenschaftlichen Bereichen, wie auch im Bereich von e-Science, spielt die Simulation eine wichtige Rolle. Im Folgenden soll ein Überblick zur Simulation geschaffen werden.

Simulation

Eine Simulation, die normalerweise auf Rechnern durchgeführt wird, stellt neben Theorie und Experiment einen dritten Weg der Wissenschaften dar. Wie bereits erwähnt wurde, erfordert e-Science die vernetzte und interdisziplinäre Zusammenarbeit sowie den Austausch von weltweit verteilten Ressourcen. Daher wird eine computergestützte Simulation benötigt. Diese lassen sich häufig als numerische Lösungen der formalisierten Theorien und als numerische Computerexperimente darstellen.

Für die Realisierung von Simulationen ist in der vorliegenden Arbeit die Zusammenarbeit von Biologie, angewandter Mathematik und Informatik besonders wichtig. Dabei liefert die Mathematik die numerischen Verfahren zu den computergestützten Lösungen der biologischen Modelle, während die Informatik für diese Modelle verantwortlich ist, die durch effiziente Algorithmen und Programme computergestützt bearbeitet werden.

Es bedarf viel Vorarbeit, um biologische Prozesse mit Hilfe von Rechnern zu simulieren. Zu den für diese Arbeit geltenden Vorarbeiten zählen beispielsweise eine mathematische Modellierung und das Erstellen von Algorithmen für die wissenschaftliche Berechnung und die Visualisierung. Mathematische Modelle zur Beschreibung von biologischen Prozessen müssen dazu zunächst computergerecht aufbereitet werden: Algorithmen geben somit Vorschriften zur Ermittlung von Lösungen an. Schließlich können die Simulationsresultate je nach Bedarf visuell umgesetzt werden.

Simulation in biologischen Reaktionsnetzwerken

Die Simulation, die in der vorliegenden Arbeit eingesetzt wurde, setzt ihren Fokus auf die Analyse und Visualisierung durch die biologische Modellierung für Parameteridentifikation in biologischen Reaktionsnetzwerken. Die biologischen Modelle, die in dieser Arbeit verwendet werden, sind in verschiedenen nicht-linearen Gleichungen umgesetzt worden, die auf der chemischen Reaktionskinetik basieren. Dabei werden dauerhafte Messungen des Status in dem biologischen Netzwerk benötigt, da Strukturänderung in diesen Modellen spontan erfolgen können und daher berücksichtigt werden müssen. Um eine Realitätsnähe zu schaffen, werden so genannte Gauß'sche Messfehler in den Modellen auf Basis von Zufallszahlen erzeugt.

Um die biologischen Modelle zu entwickeln, müssen Parameter aus experimentellen Daten verwendet werden. Dies führt jedoch zu folgenden zahlreichen Problemen „the data available

for this purpose is usually scarce and noisy, and time resolution is low, the optimization problem is ill-posed, and the performance of standard methods such as least-squares or maximum likelihood estimation is poor "[15]. Um diese Probleme bei der Nutzung von Standardmethoden zu umgehen, wird eine so genannte Parameteridentifikation zur Auswahl des passenden Modells genutzt. Bei einer solchen Parameteridentifikation kann beispielsweise nach [16] ein bayes'sche Framework eingesetzt werden, mit dem Schätzwerte für die Parameterverteilung verwendet werden und anschließend deren „Unsicherheit" berechnet werden können. Um eine möglichst genaue Parameterverteilung gewährleisten zu können, wird versucht, die Unsicherheiten im Laufe der Simulation zu minimieren. Das Ergebnis der Simulation wird visualisiert, damit der Fortschritt überprüft und die aktuelle Unsicherheit des Modells visuell analysiert werden kann.

Die ausführliche Beschreibung der Reaktionsnetzwerke und Parameteridentifikation befindet sich in Kapitel 4.

3. Verwendete Software

In diesem Kapitel werden lediglich die Softwares dargestellt, die im Rahmen dieser Arbeit von wesentlicher Bedeutung sind.

3.1. Simulationssoftware: Octave

GNU Octave kann als Open-Source Implementierung von MatLab[8] betrachtet werden und stellt eine interaktive Skriptsprache dar, die speziell für numerische Berechnungen optimiert wird. Zusätzlich zu der Lösung von Problemen der linearen Algebra und der Integralrechnung, dem Lösen von Gleichungssystemen und Polynomen etc., ermöglicht es die Definition eigener Funktionen oder Module, die in C++, C oder Fortran geschrieben sind, zu verwenden. Dadurch kann Octave problemlos mit individuell benötigten Funktionen erweitert werden.

Die Syntax von Octave ist sehr ähnlich zu der von Matlab. Octave-Programme können meist von Matlab ausgeführt werden. Umgekehrt ist dies jedoch aufgrund des größeren Funktionsumfangs von Matlab nicht immer gewährleistet.

Für die grafische Ausgabe von zwei- oder dreidimensionalen Ergebnisdaten verwendet Octave das Visualisierungsprogramm GNUplot**.**

Datentypen

Der wichtigste Datentyp in Octave ist Matrix, da fast alle Datentypen in Octave als Matrizen intern bearbeitet werden. Auch Skalare werden intern als 1×1 Matrix behandelt. Die Datentypen unterscheiden sich durch:

- **Numerische Objekte**: sowohl Skalare als auch Matrizen sind durch die numerischen Werte zugewiesen. Der Wertebereich liegt zwischen $-2{,}2251 \times 10^{308}$ und $1{,}7977 \times 10^{308}$.
- **String-Objekte:** sie sind Zeichenfolgen, die zwischen einfachen oder doppelten Anführungszeichen eingeschlossen sind (” oder ’). Intern speichert Octave Strings als Matrizen von Zeichen. Alle für Matrizen definierten Indizierungsoperationen arbeiten auch auf Strings.
- **Datenstrukturen:** Zusammenfassung von Objekten verschiedener Typen. Die Syntax ist ähnlich wie die C-style Strukturen.

[8] http://www.mathworks.de/products/matlab/

Beispiel:

```
octave:1> [1 2; 3 4]
ans =
1 2
3 4
octave:2> y = ["a for apple"]
y = a for apple
```

Operationen

Bei Octave sind normale Operatoren wie z.B. Wertzuweisung (variable = expression), logische Operatoren (<, <=, >, >=…) und Skalaroperationen (-,*,/ …) vorhanden. Zudem bestehen zahlreiche Operationen zur Matrixmanipulation, da Matrizen die wichtigsten Bausteine zur Programmierung in Octave darstellen. Darüber hinaus kommen die üblichen mathematischen Funktionen z.B. *sin, cos, exp, log* usw. bei Octave vor. Zusätzlich bietet Octave viele weitere Funktionen an, die sich mit bestimmten Themengebieten der Mathematik befassen (lineare Algebra, nicht-lineare Gleichungen, Bildverarbeitung und Sprachverarbeitung usw.).

Skript und Funktionen

Bei Octave kann eine Skript-Datei so aufgebaut werden, dass die Befehlssequenzen mit einem beliebigen Texteditor in einer ascii-File mit der Endung .m (sogenannt m-File) abgespeichert werden können. Zur Ausführung lässt sich das Skript aus der Octave- Kommandozeile mit dem Dateinamen aufrufen.

In einer Skript-Datei können mehrere Funktionen definiert werden. Die Funktionen werden wie Skripte innerhalb von Textdateien mit dem Suffix .m gespeichert und analog aufgerufen. Diesen können jedoch Parameter übergeben werden. In der ersten Zeile der Skript-Datei steht hierzu:

```
function[Ausgabeparameter]= nameDerFunction(Eingabeparameter)
```

Ausgabeparameter steht hier für eine Liste von Parametern, in denen Ergebnisse gespeichert werden können. Eingabeparameter steht hier für die Liste der Eingabeparameter der Funktion.

3.2. Web Service Wrapper

Eine der Aufgaben dieser Arbeit ist es, Octave-basierte Anwendungen als Web Service zu erstellen. Dazu wird ein Programm benötigt, welches die Funktionen der Octave-Anwendungen über eine Web Service Schnittstelle verfügbar machen. Diese Aufgabe übernimmt ein Web Service Wrapper. Laut Freund [22] ist ein Web Service Wrapper ein

Programm, „das sich wie eine Schale um die einzelnen Funktionen der Anwendung legt, sie nach innen über verschiedenste Kanäle anbindet und nach außen als Web Service kapselt“.

Die Funktionalität eines Wrappers lässt sich durch zwei Szenarien beschreiben. Das erste Szenario zielt auf Legacy-Applikationen, die auf Web Services zugreifen wollen. Da dieses Szenario nicht Bestandteil dieser Arbeit ist, wird darauf nicht weiter eingegangen.

Der Schwerpunkt dieser Arbeit liegt auf dem zweiten Szenario, bei dem eine Legacy-Applikation die Funktionalität für einen Web Service bereit stellt. Hierbei wird ein Request in der Regel als SOAP-Message an den Wrapper versendet. Bei jedem Request ruft der Wrapper die Applikation auf, die die Anforderungen des Requests erfüllt. Zudem schickt der Wrapper die Nachrichten in der Regel als SOAP-Message zurück.

Um ein Wrapper für die Octave-Anwendung zu entwickeln, wird das in der Diplomarbeit [23] beschriebene Konzept des Web Service Interface (WSI) als Basis verwendet. Im Folgenden soll dieses vorgestellt werden.

3.2.1. Web Service Interface (WSI)

Der Zweck des Web Service Interfaces ist es, Web Services auf Basis von Simulationsanwendungen zu erstellen. Aufgrund der Vielzahl der verschiedenen Anwendungen und Bibliotheken wurde das Web Service Interface generisch verwendbar aufgebaut.

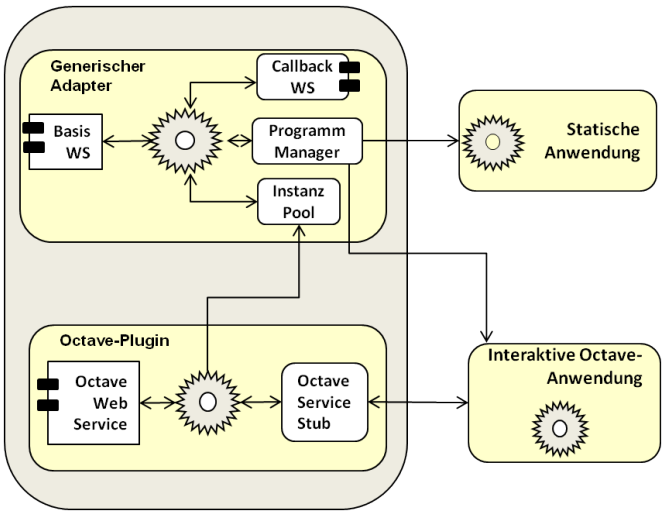


Abbildung. 9 Architektur des Web Service Interfaces [23]

Für eine Simulationsanwendung, die bisher noch nicht unterstützt wurde, kann das Web Service Interface durch Entwicklung eines Plugins erweitert werden (siehe Abbildung. 9). Da die Octave-Anwendung nicht direkt als Web Service erstellt werden kann, soll ein Octave Adapter als Plugin entwickelt werden, um eine Octave-basierte Simulation durchzuführen.

Ein generisches Basismodul (auch generischer Adapter genannt) des Web Service Interfaces wurde im Rahmen der Diplomarbeit [23] erstellt.

Der Java-basierte Basis Web Service läuft im Applikationsserver Apache Tomcat. Der Basis Web Service beschreibt mit Hilfe von WSDL grundlegende Operationen, die benötigt werden, damit ein Client eine Simulationsanwendung aufrufen kann. Dazu gehört etwa das Erstellen einer eindeutigen ID oder einer Verzeichnisstruktur auf einem Rechner, auf dem die Simulationsanwendung ablaufen soll. Der Basis Web Service bietet allerdings keine anwendungsspezifischen Operationen an.

Im Instanz Pool werden die Simulationsinstanzen verwaltet. Eine Simulationsinstanz besitzt stets eine eindeutige ID und umfasst eine Menge von Daten und Verzeichnissen, die Ausführung eines oder mehrerer Programme sowie deren aktueller Zustand. Als Beispiel soll Abbildung. 10 dienen, die die möglichen Übergänge zwischen den verschiedenen Zuständen aufzeigt. Diese Zustände entsprechen den WSDL-Operationen im Basis Web Service.

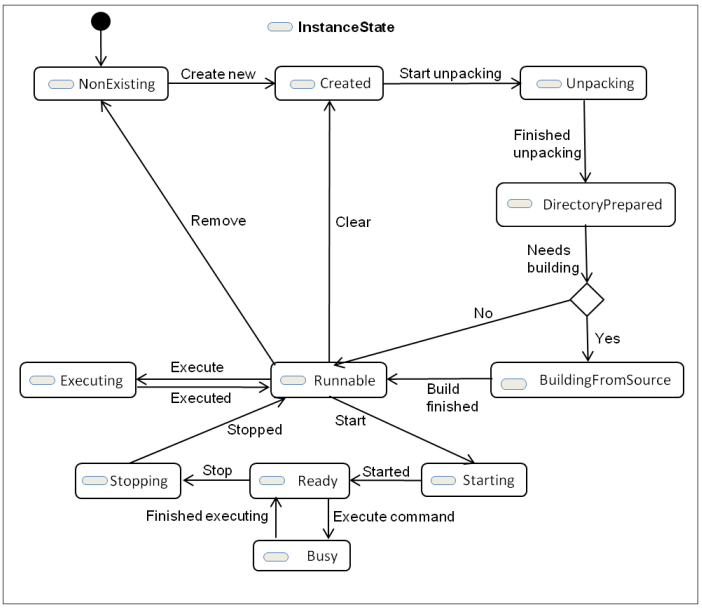


Abbildung. 10 Lebenszyklus einer Simulationsinstanz aus [23]

Die externen Programme (wie z.B. ein Entpacker) werden durch den Programm Manager ausgeführt. Sobald Anwendungen oder Skripts in einem Archive gespeichert sind, können diese vom Programm Manager entpackt werden. Weiterhin bietet der Programm Manager die Möglichkeit, die Ausgabedaten nach der Ausführung eines Programms zu speichern, um die Daten zu einem späteren Zeitpunkt weiterverwenden zu können.

Der generische Adapter stellt Callback Web Service als einen weiteren Web Service zur Verfügung. Diese Schnittstelle dient zur Benachrichtigung mittels Simulationsanwendung. Sobald die Anwendung bereit ist, erhält das Web Service Interface eine Mitteilung, die Anfragen zu bearbeiten. Anschließend schickt das Web Service Interface eine entsprechende

Benachrichtigung weiter an den Client.

Octave-Plugin enthält das auf Octave-Anwendung aufgebaute Web Service sowie ein Service Stub. In dem Octave Web Service werden beispielsweise die Grundfunktionalitäten der Octave-Anwendungen durch WSDL-Operationen dargestellt. Durch Octave Service Stub kann das Octave-Plugin mit interaktiven Simulationsanwendungen kommunizieren. Dies soll im folgenden Kapitel ausführlich behandelt werden.

3.3. Serverplattform

Für die Erstellung wie auch den Betrieb des Web Service Interfaces und Plugins wird ein Applikationsserver benötigt. In der vorliegenden Arbeit wird Apache Tomcat als Plattform verwendet. Der Applikationsserver wird zudem von Service Bus Apache Axis2 und BPEL-Engine Apache ODE verwendet. Im Folgenden soll auf diese näher eingegangen werden.

Apache Tomcat

Apache Tomcat dient als Applikationsserver und Basis für die Ausführung von Java-basierten Anwendungen und fungiert somit als die grundlegende Server-Komponente. Es basiert auf einem Open Source Projekt der Apache Software Foundation und ist ein in Java geschriebenr Servlet Container. In seiner aktuellen Version vereint der Apache Tomcat Technologieansätze wie JSP-Compiler Jasper[9] und Java Server Pages[10] mit einem kompletten HTTP-Server.

Zur Entwicklung von Web Service auf Basis von Otave wird ein "Dynamic Web Project" in Eclipse erstellt. Dazu wird zunächst eine eigene Serverumgebung konfiguriert. In der vorliegenden Arbeit wird die Version V6.0 von Tomcat eingesetzt.

Apache Axis2/Java

Bei Apache Axis 2/Java handelt sich um ein Service Bus sowie ein Java-Framework zur Entwicklung von Webservice-Anwendungen auf Basis von SOAP oder REST [11] (Representational State Transfer). Es wird häufig in einen Servlet Container (hier Apache Tomcat) genutzt und als Java Servlet betrieben.

Axis2 integriert verschiedene Binding-Frameworks. Zudem bietet es ein eigenes Binding-Framework „Axis2 Data Binding Framework“ (ADB) an. Da die Übertragung von XML-Daten in Java-Strukturen keine Probleme mit sich bringen. Mit dem ADB wird der Client einfach generiert, wobei die benötigten Klassen auch im Stub als innere Klassen

[9] JSP-Compiler Jasper : http://tomcat.apache.org/tomcat-4.1-doc/jasper-howto.html

[10] Java Server Pages : http://www.oracle.com/technetwork/java/javaee/jsp/index.html

[11] REST: http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

realisiert werden.

In Axis2 wird ein eigenes Objektmodell namens AXIOM verwendet, welches eine XML-Datenstruktur in eine Objektstruktur umwandelt. AXIOM ist leichtgewichtig und bewusst so aufgebaut, dass eine flache Klassenhierarchie mit wenigen Methoden und Attributen genutzt werden kann.

Eine weitere Komponente von Axis2 ist, dass es Tools zur Generierung von WSDL-Dateien sowie serverseitigen und clientseitigen Java Klassen durch die Batchdateien bzw. Shell Skripte `java2wsdl` und `wsdl2java` bietet. In der vorliegenden Arbeit werden die Java Klassen sowie ein Java Skeleton aus einer WSDL-Datei mit der Unterstützung von `wsdl2java` generiert. Mit dem Java Skeleton können Kommunikationskomponenten in externe Anwendungen integriert werden, sodass sie in der Lage sind, Webservices aufzurufen oder anzubieten.

Apache ODE

Als WS-BPEL-Engine kommt Apache ODE zum Einsatz. Die Abkürzung ODE steht für Orchestration Director Engine. Es handelt sich um ein Open Source Projekt der Apache Software Foundation und unterstützt die BPEL-Standards 1.1 und 2.0[12]. Es dient hierbei als Laufzeitumgebung für die BPEL-Prozesse, mit der Web Services aufgerufen, Messages gesendet und empfangen sowie Daten bearbeitet und Fehler behandelt werden können.

Apache ODE besteht aus folgenden Komponenten [24]:

- ODE BPEL Compiler: kompiliert die einzelnen BPEL-Artefakte, das BPEL-Prozess-Dokument, WSDL-Dokumente und XML Schemas in einen ausführbaren Prozess
- ODE BPEL Engine Runtime: stellt eine Umgebung für die Ausführung der kompilierten BPEL-Prozesse zur Verfügung.
- ODE Integration Layer: stellt die Verbindung von ODE BPEL Engine Runtime mit der Umgebung bereit. Beispielsweise ermöglicht der Integration Layer in Zusammenspiel mit Axis2 der BPEL Engine Runtime den Aufruf von Web Service.
- ODE Data Access Objects (DAO): zuständig für die Interaktion zwischen der ODE BPEL Engine Runtime und der darunterliegenden Datenbank, die normalerweise eine JDBC Datenbank[13] ist.

[12] BPEL-Standards 2.0: http://ode.apache.org/ws-bpel-20.html

[13] http://www.oracle.com/technetwork/java/javase/jdbc/index.html

4. Reaktionsnetzwerke

Dieses Kapitel verschafft einen Überblick über die biologische Netzwerke sowie Reaktionsnetzwerke, die im Rahmen dieser Arbeit einen wesentlichen Hintergrund für die Simulationsanwendungen gebracht haben. Den Abschluss bilden einige Möglichkeiten zur Parameteridentifikation in Reaktionsnetzwerke.

4.1. Biologische Netzwerke

Zahlreiche aktuelle Forschungen im Bereich der Biowissenschaften befassen sich aktuell mit Gene, Proteine und Metabolite sowie mit deren komplexen Zusammenspiel. Dabei steht ein besseres Verständnis der molekularen Prozesse als Grundlage primär im Zentrum der Forschungen.

Um das Verhalten und die Abhängigkeiten aller Elemente eines biologischen Systems auf einer globalen Ebene zu studieren, hat sich der Begriff Systembiologie durchgesetzt. Dabei ist die Integration von experimentellen Methoden sowie Datenanalyse und Simulation in einem iterativen Prozess erforderlich. Daher wird sie oftmals als synergetische Integration von Theorie, rechnergestützte systembiologischer Modellierung und Experiment charakterisiert.

In der Systembiologie spielen die biologischen Netzwerke eine entscheidende Rolle, da sie ein geeignetes Medium zur Integration der unterschiedlichen Bestandteile darstellen. Üblicherweise werden biologische Netzwerke durch Graphen modelliert, die aus Knoten und Kanten bestehen. Diese Elemente der Graphen werden oft durch Attribute wie Namen, numerische Werte usw. repräsentiert.

In der Abbildung. 11 wird eine vereinfachte Form der biologischen Netzwerke dargestellt. Typischerweise existiert nicht nur ein homogenes biologisches Netzwerk, sondern eine Vielzahl verschiedenster, miteinander interagierender Netzwerke. Auf unterster Ebene befinden sich die intrazellulären Netzwerke, die auf vier Elementen basieren: Gene, Transkripte, Proteine und Metabolite. Ein Gen ist ein DNA-Abschnitt, der für die Synthese eines biologischen Produkts erforderlich ist. Proteine werden durch die Informationen gebildet, die entsprechend in Genen gespeichert und durch Transkripte übermittelt werden. Viele Proteine wiederum sind Enzyme, die biochemische Reaktionen in Zellen ermöglichen. Durch Enzyme werden Metabolite umgewandelt.

Neben diesen Beziehungen existiert zudem eine Vielzahl weiterer Interaktionen zwischen den vier Bausteinen. So können Proteine miteinander interagieren, Proteine die Aktivität von Genen regulieren, und Metabolite die Aktivität von Proteinen beeinflussen usw. Somit ergibt sich ein komplexes Netzwerk vielfältiger Interaktionen und Abhängigkeiten zwischen den Bausteinen [18].

Auf den intrazellulären Netzwerken aufbauend befinden sich weitere verschiedene biologische Netzwerke, welche in den nächsten Ebenen in Abbildung. 11 dargestellt werden.
Die biologischen Netzwerke dieser Ebenen umfassen beispielsweise ökologische Netzwerke, die die Abhängigkeiten zwischen Organismen in einer Lebensgemeinschaft zeigen, hormonelle Netzwerke, die die Kommunikation zwischen Geweben und Organen repräsentieren, sowie neuronale Netzwerke, welche die Verschaltungen von Neuronen darstellen [18].

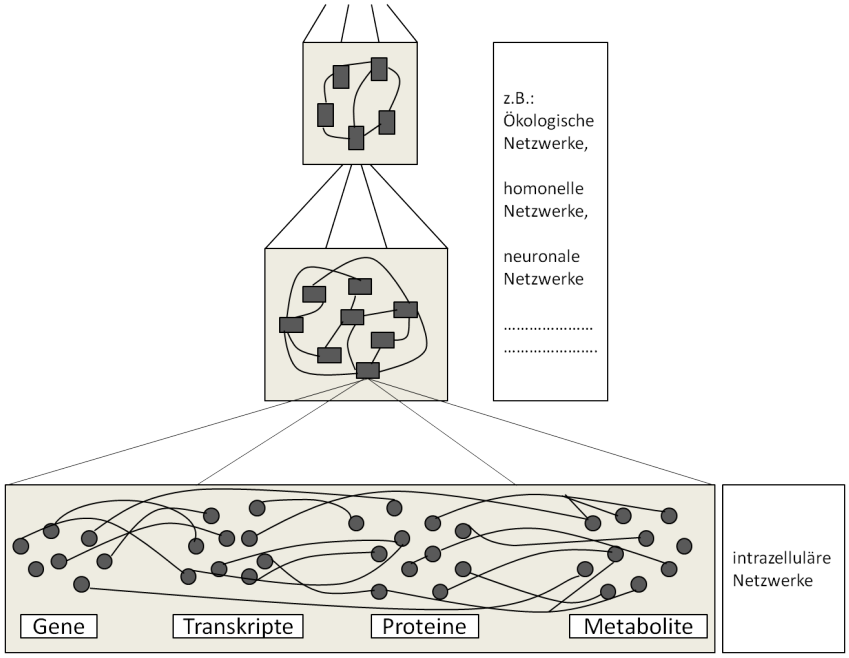


Abbildung. 11 Vereinfachte Darstellung von interagierenden biologischen Netzwerken [18]

4.2. Reaktionsnetzwerke

Wie in Abschnitt 4 beschrieben wird, werden viele Netzwerk-Modelle für die Untersuchungen im Bereich der Systembiologie erstellt. Die meisten Modelle basieren auf mathematischen Darstellungen und hängen oftmals von einer Vielzahl unbekannter oder nur ungenau bekannter Reaktionsparameter ab. Aufgrund der hohen Kosten, des Zeitaufwands zur Durchführung der Experimente oder der Komplexität durch die Vielzahl an Einzelreaktionen erweist sich die direkte Messung dieser Parameter als unrealistisch. Deshalb müssen diese Parameter aus

indirekten Messungen an der realen Zelle, zum Beispiel aus Zeitreiheninformationen, gewonnen werden.

Im Folgenden werden eine kurze Einführung in die biologische Modellierung der Reaktionsnetzwerke und ein Überblick über die Parameteridentifikation gegeben.

4.2.1. Modellierung der Reaktionsnetzwerke

Die Basis zur Modellierung biologischer Reaktionsnetzwerke bildet zumeist eine Beschreibung der auftretenden Reaktionen in der Form

$$\alpha_1 S_1 + \ldots + \alpha_{ns} S_{ns} \rightarrow \beta_{n1} P_1 + \ldots + \beta_{np} P_{np} \qquad (1)$$

Hierbei werden die Substrate S_i in die Produkte P_i umgewandelt. Die Faktoren α_i und β_i beschreiben die stöchiometrischen Verhältnisse der beteiligten Reaktionspartner.

Aus den stöchiometrischen Faktorenα_i, β_i der Reaktionen sowie etwaiger Umrechnungsfaktoren zur Kompensation unterschiedlicher Einheiten lässt sich die stöchiometrische Matrix (folgend durch *S* dargestellt) erstellen. Die stöchiometrische Matrix beschreibt, welche chemischen Verbindungen in welchen Reaktionen miteinander reagieren und welche neuen Verbindungen dabei entstehen. Die Matrix besteht aus m Zeilen und n Spalten. Die m Zeilen repräsentieren die Verbindungen (Moleküle, Metabolite), und jede der n Spalten stellt eine Reaktion dar. In komplexen Reaktionsnetzwerken ist in der Regel die Anzahl der Reaktionen größer als die Anzahl der beteiligten Moleküle (n > m).

Werden die räumlichen und stochastischen Effekte vernachlässigt, können die Reaktionsnetzwerke mittels gewöhnlicher Differenzialgleichungen beschrieben werden. Die Modelle bestehen dann aus dem Fluss (Flux) jeder Reaktion und den Differenzialgleichungen:

$$\dot{\mathrm{X}}(\mathrm{t}) = \mathrm{S} \cdot \mathrm{v} \qquad (2)$$

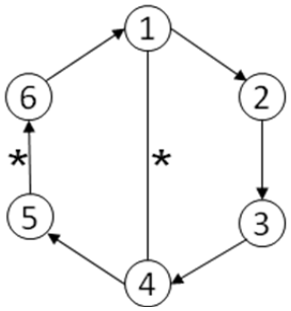
Der Flux *v* beschreibt die Reaktionsintensitäten pro Zeiteinheit für die n Reaktionen. Die Änderungsraten der Verbindungskonzentrationen *X(t)* ergeben sich aus dem Produkt der stöchiometrischen Matrix *S* mit den Reaktionsraten (Flux *v*).

4.2.2. Parameteridentifikation

In der Systembiologie werden zumeist zeitkontinuierliche Modelle, der in Abschnitt 4.2.1 beschriebenen Form, verwendet. Ein Grund dafür ist, dass derartige Modelle der physikalischen Wirklichkeit eher entsprechen. Außerdem gewährleisten diese die Linearität in den Parametern. Ein weiterer Grund für die Nutzung zeitkontinuierlicher Modelle ist, dass in der Biologie häufig stark verrauschte Messungen auftreten. Für ein solches Rauschen sind zeitkontinuierliche Modelle oftmals weniger anfällig als zeitdiskrete Modelle.

Für die zeitkontinuierlichen Modelle werden globale, und optimierungsbasierte Parameterschätzverfahren eingesetzt. Im Bereich der Reaktionstechnik sind in Bezug auf zahlreiche Reaktionsmechanismen nur unzureichende Informationen oder Parameterwerte zu finden. Beispielsweise führt die Parameterschätzung oftmals zu nichtkonvexen Problemen, weshalb die Komplexität dieser Verfahren mit der Dimension des Parameterraumes stark zu nimmt. Alternativ kommen lokale Suchstrategien zum Einsatz, die jedoch einen gut geschätzten Initialwert benötigen. Aufgrund dieser Nachteile werden mathematische Verfahren benötigt, die die Anpassung der Modellparameter an reale Experimente unterstützen. Man spricht hierbei von der Parameteridentifikation.

Für die Parameteridentifikation existieren verschiedene Methoden. Beispielsweise wird in [19] ein Verfahren zur Parameteridentifikation biochemischer Reaktion vorgestellt. Zunächst wird das Modell (2) (auf der vorherigen Seite) in eine parameterunabhängige Systemdarstellung transformiert. Anschließend wird ein Entwurf eines Beobachters aufgebaut, der zur Rekonstruierung des aus den Messungen erweiterten Zustandsvektors und zur Konvergenz des Schätzfehlers dient. Schließlich erfolgt eine Schätzung der Parameter auf Basis der geschätzten Systemzustände. Zur Illustrierung wird in diesem Beispiel das Verfahren auf Basis eines einfachen Modells des zirkadischen Rhythmus angewendet.



$$\dot{x}_1 = \theta_1 x_6 - x_1 \qquad \text{PKD}$$
$$\dot{x}_2 = \theta_2 x_1 - x_2 \qquad \text{PI(4)KIII}\beta$$
$$\dot{x}_3 = \theta_3 x_2 - x_3 \qquad \text{PI(4)P}$$
$$\dot{x}_4 = \theta_4 x_3 - x_4 - \theta_8 \frac{x_1 x_4}{1 + x_4} \qquad \text{CERT}$$
$$\dot{x}_5 = \theta_5 x_4 - \theta_6 x_5 - \theta_9 \frac{x_5}{1 + x_5} \qquad \text{ceramide}$$
$$\dot{x}_6 = -\theta_7 x_6 + \theta_9 \frac{x_5}{1 + x_5} \qquad \text{DAG.}$$

Abbildung. 12 Interaktionsgraph in der trans-Golgi Netzwerk und Modelle der Differenzialgleichungen [16]

In [16] und [17] werden andere Methoden zur Parameteridentifikation verwendet, die die Modellierung mit Hilfe von Bayesian Framework beinhalten. Bei der Modellierung werden gewöhnliche Differenzialgleichungen, die auf chemische Reaktionskinetik basieren, in folgender Form definiert:

$$\dot{x} = f(x,\theta), \qquad x \in \Re^n, \theta \in \Re^m, f : \Re_+^n \times \Re_+^m \to \Re^n \in C^1(x,\theta) \qquad (3)$$

Wobei das Vektorfeld f kontinuierlich differenzierbar ist. Der Vektor θ enthält Parameter, die aus Daten stammen. In diesem Modell werden zufällige Gauß'sche Messfehler eingebettet, um ein statistisches Bayesian Framework nutzen zu können. Bei der Modellierung wird hierbei versucht, ein ideales Experiment zur Parameteridentifikation durch die Maximierung der

Entropie einer Posterior Distribution zu erstellen, die durch so genannte Markov Chain Monte Carlo (MCMC)[14] Samplings approximiert werden müssen. Basieren auf solchen effizienten Einschätzungen der Entropie wird ein Bayesian Framework erstellt. Dieses kann in ein Netzwerkmodell überführt werden, wie beispielsweise an das Netzwerkmodell von Secretory Pathway Control im trans-Golgi Netzwerk[15]. Auf der linken Seite der Abbildung. 12 befindet sich ein Interaktionsgraph der trans-Golgi Netzwerk. Auf der rechten Seite sind verschiedene Differenzialgleichungsmodelle zu sehen, die der Beschreibung des Graphs dienen sollen.

Zum Verständnis werden zahlreiche mathematische Vorkenntnisse benötigt. [19] geht tiefer auf das Thema ein.

[14]Markov Chain Monte Carlo: http://www.cs.princeton.edu/courses/archive/spr06/cos598C/papers/AndrieuFreitasDoucetJordan2003.pdf

[15] trans-Golgi Netzwerk : http://www.zytologie-online.net/golgi-apparat.php

5. Spezifikation

In diesem Kapitel wird ein Web Service für Octave-Anwendungen spezifiziert. Es wird zunächst auf die Anforderungen an den Web Service eingegangen. Danach folgt eine Beschreibung des Lebenszyklus des Octave Adapters inklusive der möglichen Zustände, die er annehmen kann. Schließlich werden die Anwendungsfälle des Octave Adapters dargestellt.

5.1. Anforderungen

Die Anforderungen an den Octave Web Service werden in diesem Abschnitt erläutert.

Bereitstellung der Octave-Anwendung als Web Service

Der Web Service für Octave-Anwendungen soll folgende Anforderungen erfüllen. Ein Client schickt eine SOAP-Nachricht an den Octave Web Service. Der Tomcat Server bekommt die Nachricht, extrahiert und analysiert den Inhalt der Nachricht. Die Web Service Operationen sollen dabei den Anwendungsfällen eines Ablaufs entsprechen, mit dem die Octave-Anwendungen gestartet werden können. Diese Operationen werden zunächst als WSDL-Operationen definiert, dann sollen sie mit Hilfe von Axis2 in eine Java-Implementierung überführt werden. Mit den Java Codes kann die konkrete Implementierung ergänzt werden. z.B. indem ein Verzeichnis erstellt wird, in dem die Octave-Anwendungen abgelegt werden können. Rückmeldungen von den Octave-Anwendungen sollen als SOAP-Nachricht vom Octave Web Service an den Client zurückgeschickt werden.

Visualisierung des Ergebnisses

Nach der Ausführung der Octave-Anwendung entsteht eine graphische Darstellung als Ergebnis, die die Qualität der Simulation beschreibt. Da die Web Service Operationen als WSDL-Operationen definiert sind, in denen der Output nur die Formate string, long usw. enthält, kann eine graphische Darstellung nicht als Output einer WSDL-Operation genutzt werden. Es soll daher die graphische Darstellung in einer Datenbank gespeichert werden und ein Pfad für sie angegeben werden. Durch die Nutzung des Pfades kann der Client die Ergebnis-Darstellung abholen.

BPEL-Prozess für die Durchführung der Simulation

Ein BPEL-Prozess soll erstellt werden, der den Octave Web Service verwendet. In dem Prozess sollen Aktivitäten, Kontrolllogik usw. definiert werden, damit die Simulation richtig durchgeführt werden kann.

5.2. Lebenszyklus einer Simulationsinstanz von Octave Service Adapter

In dem Abschnitt 3.2.1 wird eine Simulationsinstanz des generischen Adapters dargestellt (Abbildung. 10), die die möglichen Übergänge zwischen den verschiedenen Zuständen zeigt. Im Octave Plugin (auch Octave Service Adapter genannt) wird eine Simulationsinstanz auf Basis des generischen Adapters nach der Octave Anwendung erstellt. In dieser Arbeit wird der Lebenszyklus der Simulationsinstanz nach der Octave-Simulation wie folgt definiert. Hierbei ist zu beachten, dass nicht alle Zustandsübergänge in der Abbildung. 13 dargestellt werden. Vielen Zuständen können auch in andere Zustände wechseln.

NonExisting

Dieser Zustand beschreibt, dass eine Instanz noch nicht oder nicht mehr existiert. Er kann als einen Anfangszustand dargestellt werden. Falls ein existierendes Simulationsprogramm nicht mehr ausgeführt werden soll, wird es von der Datenbank entfernt und der Zustand wird auf „NonExisting“ zurückgesetzt.

Ausgangszustände: -
Folgezustände: SimID

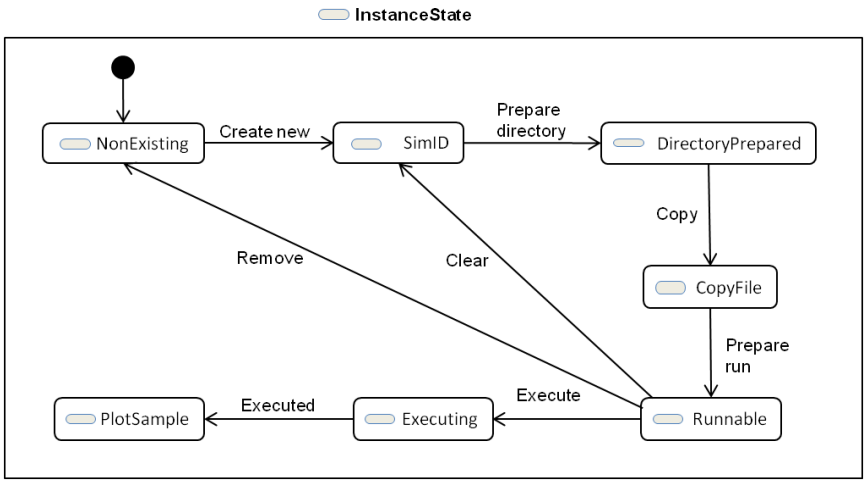


Abbildung. 13 Lebenszyklus der Simulationsinstanz von Octave Adapter

SimID

Eine neue Simulationsinstanz ist erstellt. Dazu wurde eine eindeutige Identifikationsnummer für die Instanz vergeben.
Ausgangszustände: NonExisting
Folgezustände: DirectoryPrepared, NonExisting

DirectoryPrepared

Ein Instanzverzeichnis wurde erstellt, in dem Programmdateien später vorliegen können. In diesem Zustand stehen aber noch keine ausführbaren Programme zu Verfügung.

Ausgangszustände: -
Folgezustände: SimID, CopyFile, NonExisting

CopyFile

Alle Dateien zur Ausführung der Simulationsanwendung sind in diesem Zustand in das erstellte Instanzverzeichnis kopiert worden.

Ausgangszustände: DirectoryPrepared
Folgezustände: SimID, Runnable, NonExisting

Runnable

Nachdem die Simulationsanwendung und alle benötigten Dateien im Instanzverzeichnis kopiert wurden, lässt sich die Simulation starten. Dieser Zustand kann im Verlauf einer Simulation mehrmals verwendet werden. Wenn die Simulation nicht gestartet wird, können die SimID und das Instanzverzeichnis gelöscht werden, dann geht der Lebenszyklus wieder zum Anfangszustand zurück.

Ausgangszustände: CopyFile, Executing
Folgezustände: SimID, Executing, NonExisting

Executing

Dieser Zustand wird der Ausführung der Simulation zugeordnet. Sobald die Ausführung terminiert, wechselt die Instanz in den Zustand „PlotSample“. Das heißt, das Ergebnis der Simulation wird zum „PlotSample“ geliefert.

Ausgangszustände: Runnable
Folgezustände: PlotSample

PlotSample

Dieser Zustand beschreibt ein Ergebnis nach der Ausführung der Simulationsanwendungen. In dieser Arbeit wird ein Bild als Ergebnis dargestellt. Die

Visualisierung des Simulationsergebnisses wird in diesem Zustand realisiert.

Ausgangszustände: Executing
Folgezustände: -

5.3. Octave Service Adapter

Dieser Abschnitt stellt die einzelne Anwendungsfälle des Octave Service Adapters (kurz Octave Adapter genannt) dar, die die Funktionalität des Octave Adapters beschreibt. Der Client des Adapters wird durch Akteur repräsentiert. Ein Überblick über diese Anwendungsfälle wird in der Abbildung. 14 gezeigt.

PrepareSimulation

Beschreibung: Eine neue Simulationsinstanz wird erstellt, um eine Simulation mit dem Octave Adapter verwalten zu können. Diese Aktion muss als Anfangsaktion ausgeführt werden. Eine Identifikationsnummer (im Folgenden wird SimID genannt) und ein Instanzverzeichnis werden in dieser Aktion erzeugt, damit die Instanz als existierend registriert werden kann.

Vorbedingungen: Der Octave Adapter sowie der generischer Adapter müssen aktiv sein.

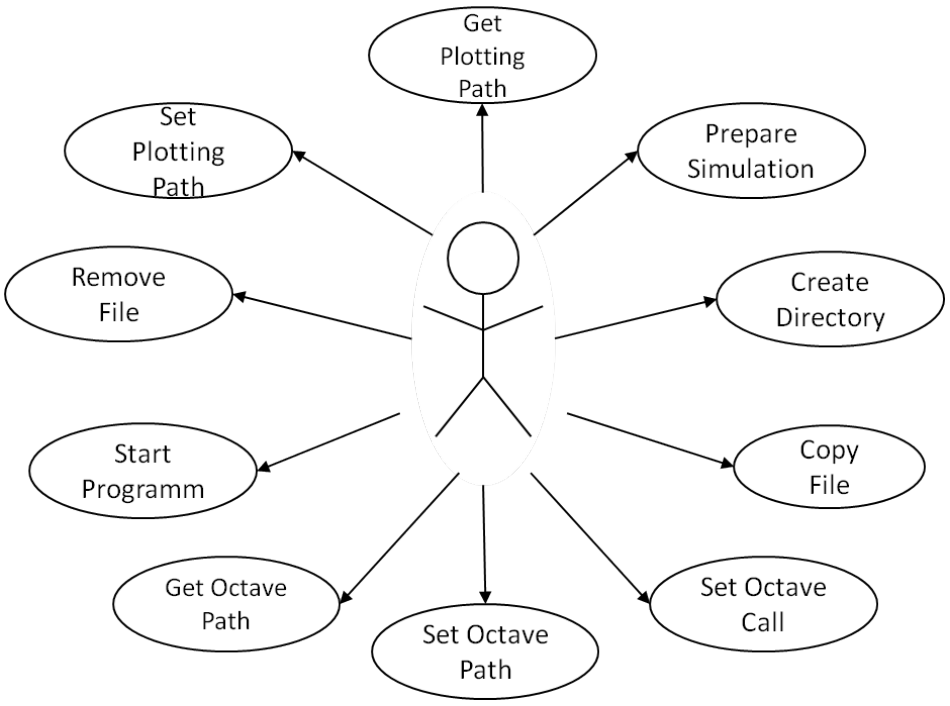


Abbildung. 14 Übersicht über die Anwendungsfälle

Nachbedingungen: Die Instanz existiert und wird mit der SimID registriert, dazu wird ein entsprechendes Instanzverzeichnis erzeugt. Jede Instanz verfügt über eine eindeutige SimID, die nicht mehr an andere Instanz vergeben kann. Andere Aktionen lassen sich auf Basis von dieser SimID erstellen oder ausführen.

Fehler:

- Die Simulationsinstanz sowie Instanzverzeichnis können nicht erstellt werden.

Regulärer Ablauf:

- Die Instanz wird erzeugt und registriert.
- Eine SimID und ein Instanzverzeichnis werden zur Instanz erstellt.

CreateDirectory

Beschreibung: ein neues Verzeichnis wird mit Hilfe von SSH auf Basis der SimID erstellt, auf dem alle Datei der Octave-Anwendungen kopiert werden können. Diese Aktion ist eine Vorbedingung für die Ausführung der Anwendung.

Vorbedingung: Die Instanz sowie das Instanzverzeichnis existieren. Zugangsrechte über SSH sind erlaubt.

Nachbedingung: Das neue Verzeichnis existiert unter Instanzverzeichnis.

Fehler:

- Die Instanz sowie das Instanzverzeichnis existieren nicht.
- Zugangsrecht über SSH ist nicht erlaubt.
- Das neue Verzeichnis kann nicht erstellt werden.

Regulärer Ablauf:

- Ein neues Verzeichnis wird erstellt.

CopyFile

Beschreibung: Nachdem ein neues Verzeichnis erstellt wurde, werden Dateien aus einem Quellverzeichnis in das neue Verzeichnis kopiert. Diese Dateien enthalten die Octave-Anwendungen, die später ausgeführt werden. Die Datei wird mit WinSCP vom Ursprungsort (hier ein Windows Server) zum Zielort (hier ist Linux Server) kopiert.

Vorbedingung: Das neue Verzeichnis sowie die Datei müssen existieren. Die Instanz muss das Kopieren der Datei erlauben.

Nachbedingung: Die Datei befindet sich im Verzeichnis.

Fehler:

- Die zu kopierende Datei existiert nicht.
- Das Verzeichnis, auf dem die Dateien kopiert werden, existiert nicht.
- Die Datei kann nicht richtig kopiert werden. Es kann sein, dass der Akteur über kein Zugangsrecht verfügt, auf die Datei zuzugreifen. Wenn ein Fehler während dem Kopieren auftritt, kann die Datei auf den Zielort nicht vollständig sein.

Regulärer Ablauf:
- Die Datei wird auf neues Verzeichnis kopiert.

SetOctaveCall

Beschreibung: Es kann sein, dass Octave auf verschiedenen Rechnern in unterschiedlichem Pfad installiert wird. Daher wird diese Aktion dazu verwendet, ein Kommando zur Ausführung eines Octave-Anwendungsfiles (nämlich m-File) in einer bestimmten Simulationsinstanz festzulegen. Ein Property-File wird, fals es noch nicht existiert, bei dieser Aktion erstellt der den konkreten Octave-Aufruf enthält.

Vorbedingung: Die Simulationsinstanz und SimID existieren.

Nachbedingung: Ein Property-File wird erstellt.

Fehler:

- Die Simulationsinstanz und SimID existieren nicht.

Regulärer Ablauf:

- Ein Property-File für einen Octave-Aufruf wird erstellt.
- Ein neuer Wert mit dem Property-File wird zurückgeliefert, sonst wird das Property-File neu erzeugt.

SetOctavePath

Beschreibung: Ein Pfad zur Ausführung der Octave-Anwendung wird definiert. Dabei ist zu beachten, dass es viele m-Files in der auf neuem Verzeichnis kopierten Datei gibt. Wie in dem Abschnitt 3.1 beschrieben existiert ein m-File als Skript-Datei, die die Organisation des Programmablaufs beschreibt. Die in der Skript-Datei benötigten Funktionen werden in weiteren m-Files definiert. Der Pfad bestimmt die Skript-Datei, die ausgeführt werden soll.

Vorbedingung: Die Datei von Octave-Anwendung muss vollständig kopiert werden. Sie enthält mindestens eine Skript-Datei und alle benötigte Funktionen.

Nachbedingung: Ein Pfad wird an der Skript-Datei gesetzt.

Fehler:

- Die Octave-Files wurden nicht vollständig kopiert.

- Skript-Datei existiert nicht.
- Pfad wird aus verschiedenen Gründen nicht erfolgreich gesetzt.

Regulärer Ablauf:

- Ein absoluter Pfad ist einem m-File zugeordnet.

GetOctavePath

Beschreibung: Der in der Aktion *SetOctavePath* erstellte Pfad wird hier zurückgegeben. Zum Starten der Octave-Anwendung werden dieser Pfad und die SimID usw. benötigt.

Vorbedingung: Der Pfad zu einem m-File existiert.

Nachbedingung: Ein Pfad wird zurückgegeben.

Regulärer Ablauf:

- Ein Pfad zur Ausführung der Octave-Anwendung wird gespeichert.

StartProgram

Beschreibung: Über einen SSH-Aufruf wird die Octave-Anwendung gestartet. Dazu wird ein cmd-Befehl definiert, in der ein Pfad verwendet wird, der über *GetOctavePath* erfragt wird. Zur Ausführung der Octave-Anwendung werden außer dem cmd-Befehl noch eine Shell-Datei, SimID sowie weitere Pfadangaben benötigt. Die Shell-Datei dient zum Starten des Octave-Programms.

Vorbedingung: Das Instanzverzeichnis, Octave-Anwendungen, Skript-Datei und alle benötigte Funktionen existieren. Das Octave-Programm muss auf den Server installiert sein. Die Zugangsrechte stehen zu dem Zeitpunkt zur Verfügung, zudem der Aufruf per SSH stattfindet. Alle zum Starten der Octave Anwendungen benötigte Dateien beispielsweise die Shell-Datei, sowie der cmd-Befehl müssen definiert sein. Wenn die Octave-Anwendungen in dieser Aktion nicht gestartet werden können, kann das Verzeichnis mit der Aktion *RemoveFile* aufgeräumt werden.

Nachbedingung: Die Octave-Anwendungen werden auf den Server ausgeführt.

Fehler:
- Das Octave-Programm wird nicht gestartet.
- Das Octave-Pakete zur Ausführung der Anwendungen fehlt.
- Die auszuführende Skript-Datei existiert nicht.
- Zugang zur Anwendung per SSH kommt nicht zu Stande.
- Die Shell-Datei ist nicht vorhanden.

Regulärer Ablauf:

- Eine SSH-Verbindung wird aufgebaut.

- Die Octave-Anwendung wird gestartet und ausgeführt.

RemoveFile

Beschreibung: Wenn die Octave-Anwendungen nicht ausgeführt werden oder die Datei, die die diese Anwendungen enthält, nicht mehr benötigt wird, kann die Aktion *RemoveFile* durchgeführt werden. Über SSH werden die Dateien von dem Server aufgeräumt. Dadurch kann Speicherplatz gespart werden.

Vorbedingung: Die zu löschende Datei muss existieren. Zugangsrecht über einen SSH-Aufruf steht zur Verfügung.

Nachbedingung: Die Datei wird aufgeräumt.

Fehler:

- Die Datei existiert auf den Zielort nicht.
- Zugangsrecht oder das Schreibrecht per SSH sind nicht vorhanden.

Regulärer Ablauf:

- Die Datei, die die Octave-Anwendungen enthält, wird über SSH-Befehl gelöscht.

SetPlottingPath

Beschreibung: Nach der Ausführung der Octave-Anwendungen werden Ergebnisse erzeugt, die durch verschiedene Koordinatenachsen auf ein Bild dargestellt werden. Ein Beispiel zeigt in Abbildung. 15. Die Visualisierung der Simulationsergebnisse ist eine Aufgabe dieser Arbeit. Eine Aktion wird ähnlich der die Aktion *SetOctavePath* durchgeführt. In dieser Aktion ist dem Ergebnisbild ein Pfad zugeordnet.

Vorbedingung: Die Octave-Anwendungen wurden erfolgreich ausgeführt. Der Octave Adapter wartet bis alle Ergebnisse in einem Bild erzeugt worden sind. Das Bild kann beispielsweise im Verzeichnis WebContent des Octave Adapters gespeichert werden.

Nachbedingung: Das Ergebnisbild wird gespeichert und ist einem Pfad zugeordnet.

Fehler:

- Die Octave-Anwendungen wurden nicht erfolgreich ausgeführt.
- Die Entstehung der Ergebnisse dauert zu lang, der Octave Adapter beendet sich per „timeout“.
- Das Ergebnisbild kann nicht gespeichert werden.

Regulärer Ablauf:

- Ein Pfad wird für das Ergebnisbild erstellt.

GetPlottingPath

Beschreibung: Diese Aktion wird dazu verwendet, ein Pfad zum Ergebnisbild zu erhalten. Durch Aufruf des Pfads können die Ergebnisse dargestellt werden.

Vorbedingung: Der Pfad zum Ergebnisbild existiert.

Fehler:

- Der Pfad kann nicht zurückgegeben werden.

Regulärer Ablauf:

- Der Pfad, der in der Aktion *SetPlottingPath* definiert wurde, wird zurückgegeben.
- Für die Visualisierung der Simulationsergebnisse wird dieser Pfad benötigt.

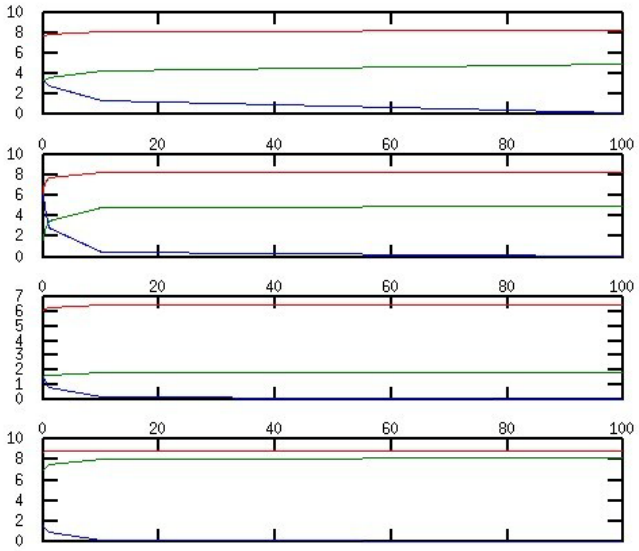


Abbildung. 15 Ergebnisbilder nach der Ausführung der Octave-Anwendungen

6. Entwurf

Dieses Kapitel beschreibt die Architektur und die Web Service-Operationen vom Octave Plugin. Die im Abschnitt 5.2 beschriebenen Anwendungsfälle des Octave Plugins werden durch Web Service Operationen und deren Parameter ausführlich dargestellt.

6.1. Architektur des Octave Adapters

Im Kapitel 3.2.1 wurde der generische Adapter des WSIs dargestellt, wobei das Octave Plugin bereits erwähnt wurde. Da Octave-Anwendungen nicht direkt als Web Service erstellt werden können, braucht man einen Adapter, der auf das WSI basiert und auf dem ein entsprechenden Web Service für die Octave-Anwendung bereitgestellt wird. In diesem Abschnitt wird dies genauer beschrieben.

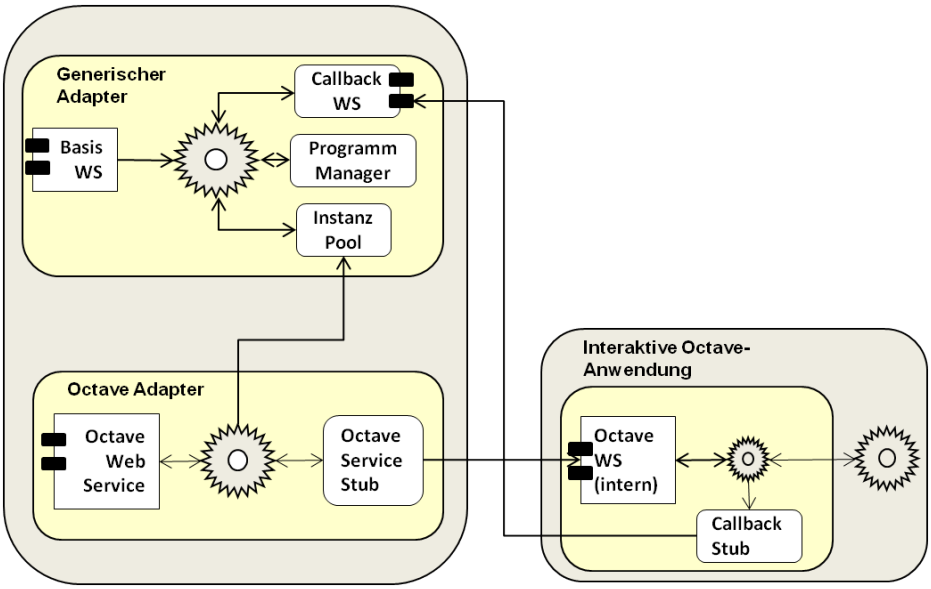


Abbildung. 16 Architektur des Octave Adapters

In der Abbildung. 16 wird die Architektur des Octave Adapters gezeigt, die die Interaktion zwischen dem generischen Adapter, dem Octave Adapter und einer interaktiven Octave-Anwendung darstellt. Der Octave Adapter stellt einerseits einen Web Service für den

Client bereit, andererseits stellt er die Kommunikation mit der Octave-Anwendung zur Verfügung. Er kann die Funktionalitäten des generischen Adapters verwenden, zum Beispiel greift es auf den Instanz-Pool und den Programm-Manager zu, um eine Simulationsinstanz zu erzeugen. Mit Hilfe des Adapters kann die Octave-Anwendung synchron oder asynchron gestartet werden. Durch den Octave Service Stub kann der Octave Adapter die Web Service Anfragen an die Octave-Anwendung weiterleiten. Ein Callback Stub wird für die Benachrichtigung bereitgestellt. Wenn die Octave-Anwendungen bereit sind, schickt dieses Callback Stub eine Rückmeldung an den Callback Web Service des generischen Adapters. Dann fängt der Web Service an, die Anfragen zu bearbeiten.

6.2. Web Service Operationen vom Octave Adapter

In diesem Abschnitt werden die Web Service Operationen beschrieben. Diese Web Service Operationen entsprechen der Anwendungsfälle des Octave-Adapter in Abbildung. 14. Die WSDL-Beschreibung liegt im Anhang dieser Arbeit.

prepareSimulation

Eine neue Simulationsinstanz wird in dieser Operation erstellt.

Anwendungsfall: Erstellen von Simulation ID

Input/Output	Parametername	Parametertyp	Beschreibung
Input	`Name`	String	Name der Applikationsanwendung
Output	`simID`	Long	Die Simulation ID der neuen Instanz
Output	`ReturnMessage`	String	Return message

Tabelle 1 Parametre der Operation prepareSimulation

Mögliche Fehlerfälle: `InvalidStateFault, CommandFault`

createDirectory

Ein neues Verzeichnis wird über SSH auf dem Zielrechner erstellt

Anwendungsfall: Erstellen eines Verzeichnisses auf Zielrechner.

Input/Output	Parametername	Parametertyp	Beschreibung
Input	`SimID`	Long	Simulation ID der Instanz
Input	`User`	String	Username des Rechners, wo das Verzeichnis steht
Input	`Host`	String	Hostname, wo das Verzeichnis steht
Input	`Directory`	String	Der Pfade des erstellten

			Verzeichnisses
Output	`ReturnMessage`	String	Nachricht zurück: „error“ oder „no error“

Tabelle 2 Parametre der Operation createDirectoty

Mögliche Fehlerfälle: `InvalidStateFault, CommandFault`

copyFile

Eine Datei wird über WinSCP von einem Ursprungspfad zu einem Zielpfad kopiert. Die Datei existiert in neuem Ort mit neuem Pfad.

Anwendungsfall: Kopieren der Dateien zwischen Rechnern

Input/Output	Parametername	Parametertyp	Beschreibung
Input	`SimID`	Long	Simulation ID der Instanz
Input	`SourceUser`	String	Username des Rechners, wo die Quelledatei steht
Input	`SourceHost`	String	Hostname, wo die Quelledatei steht
Input	`SourceFile`	String	Der Ursprungspfad der Datei
Input	`TargetUser`	String	Username des Zielrechners, wo die Datei hin kopiert wird
Input	`TargetHost`	String	Hostname, wo die Datei hin kopiert wird
Input	`TargetFile`	String	Der Zielpfad der Datei
Output	`ReturnMessage`	String	Nachricht zurück: „error“ oder „no error“

Tabelle 3 Parametre der Operation copyFile

Mögliche Fehlerfälle: `InvalidStateFault, CommandFault`

setOctavePath

Ein absoluter Pfad zur Ausführung einer Instanz einer Octave-Anwendung wird gesetzt.

Anwendungsfall: Setzten des Pfads für Aufruf der Octave-Anwendung.

Input/Output	Parametername	Parametertyp	Beschreibung
Input	`SimID`	Long	Simulation ID der Instanz
Input	`OctavePath`	String	Der Pfad zur Octave-Anwendung
Output	`ReturnMessage`	String	Nachricht mit „OctavePath set“ oder „error! OctavePath not set!“ zurück

Tabelle 4 Parametre der Operation setOctavePath

Mögliche Fehlerfälle: `InvalidStateFault, CommandFault`

getOctavePath

Der gesetzte Pfad für Aufruf der Octave-Anwendung wird nach Nennung der SimID ausgegeben.

Anwendungsfall: Erhalten des Pfads für Aufruf der Octave-Anwendung.

Input/Output	Parametername	Parametertyp	Beschreibung
Input	`SimID`	Long	SimID der Instanz
Output	`OctavePath`	String	Der Pfad zur Octave-Anwendung

Tabelle 5 Parametre der Operation getOctavePath

Mögliche Fehlerfälle: `InvalidStateFault, CommandFault`

startProgram

Die Octave-basierte Anwendung wird über SSH ausgeführt. Die Voraussetzung für die Ausführung ist, dass der Pfad für Octave schon gesetzt worden ist.

Anwendungsfall: Ausführung des Anwendungsprogramms von Octave

Input/Output	Parametername	Parametertyp	Beschreibung
Input	`SimID`	Long	Simulation ID der Instanz
Input	`User`	String	Username des Rechners, auf dem Octave installiert ist.
Input	`Host`	String	Hostname des Rechners, auf dem Octave installiert ist
Input	`Path`	String	Pfad, unter dem ein installiertes Octave zu finden ist.
Input	`Program`	String	Der Name des zu auszuführenden Octave-Programms
Output	`ReturnMessage`	String	Nachricht mit „no errors“ oder „error“ zurück

Tabelle 6 Parametre der Operation startProgram

Mögliche Fehlerfälle: `InvalidStateFault, CommandFault`

removeFile

Wenn die Octave-Anwendungen nicht gestartet werden, wird die Datei über SSH auf dem Zielrechner gelöscht.

Anwendungsfall: Entfernung einer Datei.

Input/Output	Parametername	Parametertyp	Beschreibung
Input	`SimID`	Long	Simulation ID der Instanz
Input	`User`	String	Username des Rechners, auf dem Octave installiert ist.
Input	`Host`	String	Hostname des Rechners, auf dem Octave installiert ist.
Input	`Path`	String	Pfad, unter dem ein installiertes Octave zu finden ist.
Output	`ReturnMessage`	String	Nachricht mit „no errors“ oder „error“ zurück

Tabelle 7 Parametre der Operation removeFile

setPlottingPath

Das Ergebnis der Ausführung einer Octave-Anwendung, ist beispielsweise ein Bild. Es wird ein Pfad gesetzt, damit das Ergebnis durch Aufruf des Pfads abgeholt werden kann.

Anwendungsfall: Setzten des Pfads für die Ergebnisdaten (Bilder) nach der Ausführung der Octave-Anwendung

Input/Output	Parametername	Parametertyp	Beschreibung
Input	`SimID`	Long	Simulation ID der Instanz
Input	`PlottingPath`	String	Der Pfad von Plotting
Output	`ReturnMessage`	String	Nachricht mit „Path set“ oder „error! Path not set!“ zurück

Tabelle 8 Parametre der Operation setPlottingPath

getPlottingPath

Der gesetzte Pfad zu den Ergebnisdateien stimmt mit SimID überein.

Anwendungsfall: Erhalten des Pfads zu den Ergebnisdateien

Input/Output	Parametername	Parametertyp	Beschreibung
Input	`SimID`	Long	Simulation ID der Instanz
Output	`PlottingPath`	String	Der Pfad von Plotting

Tabelle 9 Parametre der Operation getPlottingPath

6.3. Octave basierte Workflows

Ein Octave-basierter Workflow wird mit Hilfe eines BPEL-Prozesses in Eclipse erstellt. Wobei der Prozess den Ablauf der Octave-Anwendungen darstellt. Der BPEL-Prozess ist in der Abbildung. 17 zu sehen.

Wenn der BPEL-Prozess gestartet wird, fängt der Teilprozess `Prepare Steps` als erster Schritt an. Dieser Teilprozess wird als eine Schleife definiert. Es wird eine Simulationsinstanz für die Octave-Anwendungen mit der ersten Aktion `Create Octave Instance` erzeugt (siehe Abbildung. 18). Dann wird mit der Aktion `Create Directory` ein Simulationsverzeichnis auf dem Rechner erstellt, auf dem Octave ausgeführt wird. Als nächste Aktion `Copy m.file` wird die Anwendungsdatei in das erstellte Verzeichnis kopiert. Das auszuführende Octave Script File wird dann durch die Aktion `set a path to m.file` mit einem Pfad gesetzt. Anschließend wird mit der Aktion `get the path from m.file` der Pfad zurückgegeben.

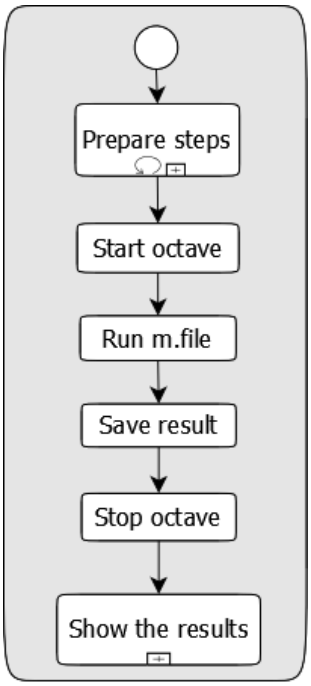


Abbildung. 17 Workflow-Prozess für die Octave-Anwenudng

Ein Grund für die Definition des Schrittes `Prepare Steps` als Schleife ist, dass sich die Octave-Anwendungen verändern können. Sie werden so oft modifiziert, um bessere Simulationsergebnisse zu bekommen. Diese Schleife kann so oft ausgeführt werden, bis man die neuesten modifizierten Anwendungen und den entsprechenden Pfad bekommt.

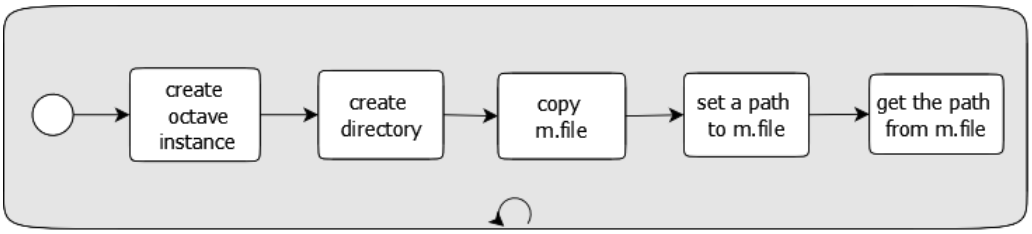


Abbildung. 18 Prepare Steps

Nachdem die Aktivität `Prepare Steps` durchgeführt wurden, wird Octave mit der Aktion `Start Octave` gestartet. Danach wird eine Rückmeldung vom Web Service Server an den Octave Adapter geschickt, um dem Client mitzuteilen, dass Octave gestartet wird, die Simulationsanwendungen bereit sind und die Anfragen bearbeitet werden können. Als Nächstes wird die Octave-Anwendung (auf Basis eines m-File) mit einer aktuellen SimID und einem erstellten Pfad als Startparameter ausgeführt, wobei dies in der Aktion `Run m.file` stattfindet. Nach der Durchführung der Anwendungen bekommt man die Ergebnisse, die bei der Aktion `Save results` in der Datenbank gespeichert werden. Schließlich wird Octave mit der Aktion `Stop octave` beendet.

Je nach Anspruch können die Octave-Anwendungen modifiziert werden um unterschiedliche Ergebnis-Typen zu erstellen. Hier in der Arbeit wird eine visuelle Darstellung als Ergebnis erzeugt, die durch die letzte Aktion `Show the results` behandelt wird. Zur Visualisierung der Ergebnis-Darstellung wird der Pfad zu den Ergebnisdaten übergeben. Ähnlich wie `Prepare Steps` ist `Show the results` wieder ein Teilprozess, und wird auch als eine Schleife definiert. In der Abbildung. 19 sieht man, dass bei dem Aufruf des Workflowfragments `Show the results` die erste Aktion `set plotting path` gestartet wird. Damit wird ein Pfad zur zuvor von Octave abgespeicherten Darstellung gesetzt. Der Pfad wird mit Hilfe der Aktivität `get plotting path` übergeben. Wenn man die Ergebnis-Darstellung anschauen möchte, wird dies durch Aufruf des Pfads realisiert.

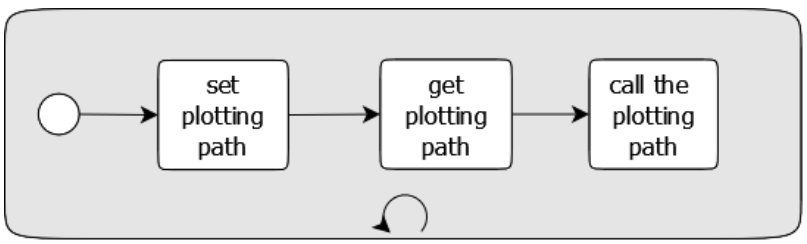


Abbildung. 19 show the results

In folgenden Tabellen sind die Parameter der Workflow-Operationen aufgelistet.

Parametername	**Datentyp**	**Beschreibung**
SimID	long	Identifikationsnummer der Simulationsinstanz
User	string	der Benutzername auf Zielrechner
Host	string	der Hostname auf Zielrechner
SrcUser	string	der Benutzername auf Ursprungsrechner
SrcHost	string	der Hostname auf Ursprungsrechner
SrcFile	string	ursprüngliches File
File	string	File auf Zielrechner
OctavePath	string	Pfad zur auszuführenden Octave-Anwendung

Tabelle 10 Parameter der Workflow-Operationen

7. Implementierung

Im Rahmen dieser Arbeit wurden die Octave-Anwendungen betrachtet und modifiziert. In diesem Kapitel wird zunächst der ursprüngliche Ablauf der Octave-Anwendung dargestellt. Danach folgt die Beschreibung der interaktiven Ausführung auf Basis eines modifizierten Simulationsablaufs und des Octave Adapters (siehe dazu auch die Abschnitte 5.2 und 5.3).

Die Implementierungen des Octave Adapters und die Modifikation an den Octave-Anwendungen wurden in der Umgebung Eclipse 3.7 entwickelt, wobei Apache Tomcat und Axis2 installiert waren. Die einzelnen Web Service Operationen wurden im WSDL-Editor erstellt. Ein entsprechendes Java Skeleton wurde mit Axis2 generiert. Schließlich wurde der Octave Service Adapter mit Hilfe von Tomcat veröffentlicht.

7.1. Ablauf der Octave-Anwendungen

Die Octave-Anwendungen wurden je nach Anspruch unterschiedlich aufgebaut. Es gibt jedoch keinen großen Unterschied zwischen den Abläufen der verschiedenen Anwendungen. Als ein repräsentatives Beispiel wird ein Octave m-File **`insulin.m`** vorgestellt, das auf Basis einer Skript-Datei definiert wird. Der entsprechende ursprüngliche Ablauf ist in der Abbildung. 20 zu sehen und wird im Folgenden beschrieben:

- Vor der Ausführung muss Octave gestartet werden.
- Das Octave Main Script File (hier die Datei **`insulin.m`**) wird wie folgend ausgeführt.

    1. Ein Modell, das zur Ausführung des Script Files verwendet wird, soll von dem Benutzer je nach Anspruch erstellt werden. Es kann direkt im Script File definiert werden oder als ein zusätzliches m.File gespeichert werden, das vom Script File eingelesen werden kann. Im Beispiel **`insulin.m`** wird ein Modell, das eine Dosiswirkung von Insulin beschreibt, mit Hilfe eines parameterfreien Modeling-Frameworks direkt im Script File definiert.
    2. Das erstellte Modell kann überprüft werden, ob es zur Simulation passt. In der Datei **`insulin.m`** wird das Modell nach der Überprüfung als eine Variable in dem „current working directory“ gespeichert, auf dem Octave installiert wurde und der Octave-Befehl aufgerufen wird.
    3. Wie die Erstellung von „Modell“ kann die Variable „Data“ auch in einem

zusätzlichen m.File definiert werden, das vom Script File eingelesen wird. Oder die Variable „Data“ wird direkt im Script File definiert. In diesem Fall soll zuerst ein Wert der Variable „Prior“ vor der Definition des „Data“ zuordnen. Im File **`insulin.m`** wird ein Wert **`a`** (a={0.4, ones(m,1)}) der Variablen „Prior“ zugewiesen.

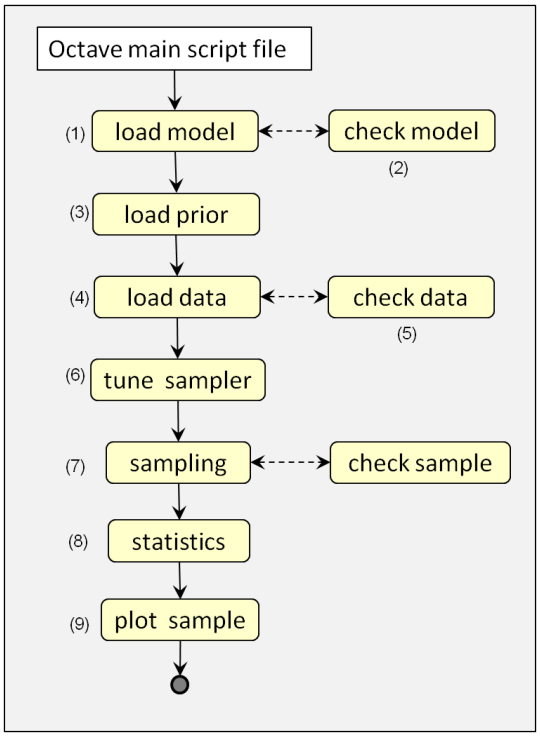


Abbildung. 20 Ablauf der Octave-Anwendungen

4. Nach der Zuweisung eines Wertes an die Variable „Prior“ wird die Variable „Data“ definiert. Die Erstellung der Variable „Data“ erfolgt auch in der Datei **`insulin.m`**.

5. Die erstellte Variable „Data“ kann getestet werden. Dann kann sie im Arbeitsverzeichnis gespeichert werden.

6. Ähnlich wie in Schritt 1, wenn kein zusätzliches m.File für die Variable „tune sampler“ erstellt wurde, wird sie direkt im Script File definiert. Zur Definition von „tune sampler“ wird eine der fünf Funktionen verwendet, die zusätzlich als m.Files erstellt wurden. Diese Funktionen sind **`ssHMClf.m`**, **`ssME.m`**, **`ssaME.m`**, **`ssHMC.m`** und **`xssHMC.m`**. Mit Hilfe von den Funktionen wird die Variable „tune sampler“ erstellt und im Arbeitsverzeichnis gespeichert. In der Datei **`insulin.m`** wird die Variable „ftuned“ als „tune sampler“ gespeichert.

7. Die Variable „sampling“ wird in **`insulin.m`** als „fsample“ definiert. Nach der

Überprüfung wird sie im Arbeitsverzeichnis gespeichert.

8. Statistische Daten für „plot sample“ werden erstellt.

9. Der letzte Schritt „plot sample“ verwendet alle in den vorherigen Schritten erstellte Variable und entstandene Daten, damit die Ergebnis-Darstellung erzeugt werden kann. Im File **`insulin.m`** wird die Darstellung schließlich gespeichert.

- Alle Variable und die Ergebnis-Darstellung wurden im Script File **`insulin.m`** nicht als zusätzliche Dateien sondern nur im Arbeitsverzeichnis gespeichert. Dies soll modifiziert werden, da die Darstellung in einer Datenbank gespeichert werden soll, auf die der Web Service zugreifen kann.
- Wenn das Octave-Programm das Script File ausführt, werden die erstellten Variablen und die Funktionen automatisch aus anderen Dateien eingelesen.

7.2. Modifizierter Ablauf mit dem Octave Adapter

Wie im Abschnitt 6.1 beschrieben stellt der Octave Adapter einerseits einen Web Service für den Client bereit, andererseits stellt er die Kommunikation mit der Octave-Anwendung zur Verfügung. Damit werden die Anfragen des Clients an den richtigen TCP-Port weitergeleitet und der Zustand der Simulationsinstanz wird geprüft.

Eine Simulationsanwendung kann im Normalfall mit dem generischen Adapter synchron oder asynchron ausgeführt werden. Im Rahmen dieser Arbeit wird die Anwendung durch den Octave Adapter nur synchron gestartet. Da die Ergebnis-Darstellung nun am Ende des Anwendungsablaufs entsteht, muss die Anwendung auf die Terminierung der Darstellung warten.

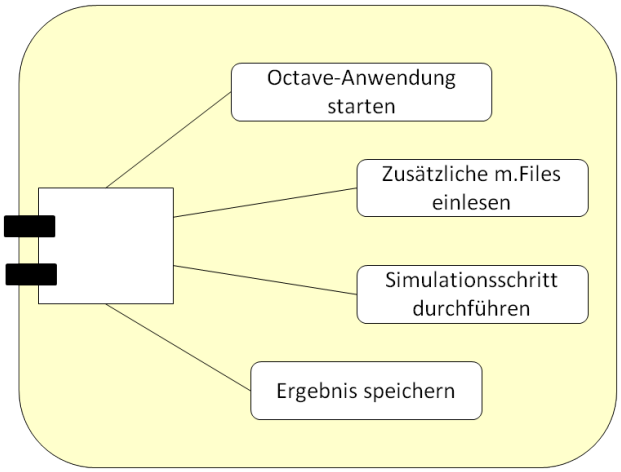


Abbildung. 21 Aufteilung der Octave-Anwendungen

Eine Zeitüberschreitung kann ausgeschlossen werden, da sich die Octave-Anwendung und die entsprechenden Adapter auf demselben Rechner befinden. Für HTTP-Anfragen kann die Zeitbeschränkung auf den maximalen Wert, ca. 290 Millionen Jahre, erweitert werden [23]. Zu Verbindungsproblemen kann es aber wegen fehlender oder nicht passender Zugangsrechte bei der Nutzung von SSH oder WinSCP kommen.

Der ursprüngliche Ablauf der Octave-Anwendung musste stark modifiziert werden, damit sich die Funktionalitäten des Octave Adapters nutzen lassen können. Abbildung. 21 zeigt einen modifizierten Anwendungsablauf, der in entsprechende Teile unterteilt wird. „Diese Teile entsprechen teilweise den Methoden, die in der ursprünglichen Anwendung enthalten waren bzw. verwendet wurden und nun als Web Service-Operationen aufgerufen werden können“ [23]. Nicht alle Teile werden verwendet, je nach Anwendungsfall können sie weggelassen werden. Zum Beispiel wurden in der Anwendung **`insulin.m`** keine zusätzliche m.Files eingelesen.

Abbildung. 22 zeigt eine Interaktion zwischen den Web Services und den modifizierten Ablauf, der durch einen gelben Kasten repräsentiert wird. Es wird im Folgenden beschrieben.

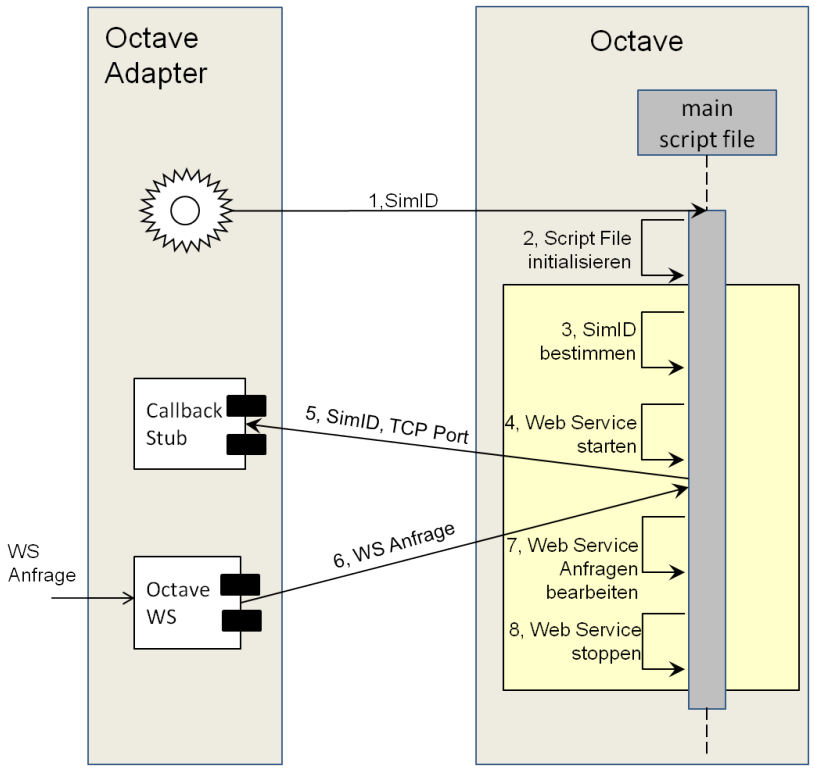


Abbildung. 22 Modifizierter Ablauf mit Interaktion vom Octave Adapter

1. Das Octave Programm wird über den Octave Adapter gestartet. Dazu wird eine Simulationsinstanz für die Anwendungen erstellt und eine Identifikationsnummer (SimID) als Kommandozeilenparameter genutzt.

2. Die Octave-Anwendung, hier das Script File **`insulin.m`**, wird mit dem Octave Adapter über SSH gestartet.
3. Die SimID wird aus den Kommandozeilenparametern ausgelesen.
4. Der von Axis2 erstellte Octave Web Service Server wird gestartet und an einen freien TCP-Port gebunden.
5. Sobald das Callback Stub die Information über die SimID und den TCP-Port bekommen, teilt es dem Web Service Interface mit, dass die Anwendung bereit ist, damit die Anfrage des Clients weiterleitet werden und die Simulationsinstanz den korrekten TCP-Port zuordnen kann.
6. Der Client schickt eine Anfrage an den in Octave integrierte Web Service Server, der dann diese Anfrage bearbeitet.
7. Wenn diese Anfrage vom Web Service Server aufgenommen und weiter bearbeitet wird, werden die entsprechenden Operationen des Web Services aufgerufen. Die Antwort wird an den Octave Adapter zurückgesendet. Dann wartet der Server auf eine nächste Anfrage des Clients.
8. Der Web Service stoppt, wenn eine Operation zum Stoppen vom Client bzw. Octave Adapter aufgerufen wird. Weitere Operationen des Octave Adapters sowie die Octave-Anwendungen können dann nicht mehr ausgeführt werden.

7.3. Weitere Anmerkungen zur Implementierung

In diesem Abschnitt befinden sich weitere Beschreibungen zur Implementierung in den Bereichen von Web Service und Simulationstest.

7.3.1. Erstellen eines Dynamic Web Projects

Es gibt zwei Möglichkeiten, das Erstellen eines Web Services in Eclipse zur Verfügung zu stellen: die Top-Down-Methode und die Bottom-Up-Methode. Für die Implementierung des Octave Web Services kommt die Top-Down Methode zum Einsatz. Bei dieser Vorgehensweise erzeugt Eclipse auf Basis einer vorliegenden WSDL-Datei alle notwendigen Java-Klassen, die um die konkrete Implementierung ergänzt werden müssen.

Zum Erstellen des Octave Web Services muss ein „Dynamic Web Project“ in Eclipse genutzt werden. Es bietet eine Umgebung an, bei der der Service erstellt, modifiziert und veröffentlicht werden kann. Dabei müssen zwei Voraussetzungen erfüllt werden: zunächst muss Tomcat korrekt in Eclipse installiert sein, dann muss Axis2 mit Eclipse verbunden werden. In dieser Arbeit werden Tomcat 6.0 und Axis2-1.6.1 in Eclipse verwendet.

Beim Erstellen des Web Services kann man unter Target Runtime den Server Tomcat 6.0 auswählen. Die eigentliche Konfiguration des Servers erfolgt nach dem Einstellen des Dynamic Web Projects. Dann werden eine ganze Reihe von Verzeichnissen und Dateien im Dynamic Web Projects erzeugt.

7.3.2. Erstellen einer WSDL-Datei

Die in dem Abschnitt 6.2 dargestellten Web Service Operationen werden im Eclipse WSDL-Editor definiert. Die WSDL-Operationen dienen zur Beschreibung des Web Services. Auf der linken Seite der Abbildung. 23 sieht man den Octave Web Service, der einen Port besitzt. In diesem Fall ist dieser Port unter einer lokalen Adresse des Tomcat-Servers erreichbar, die sich je nach Anspruch verändern kann. Rechts steht das Interface des Web Services, das durch ein entsprechendes Binding mit dem Web Service verbunden ist. Die WSDL-Operationen werden in diesem Web Service Interface definiert. Jede Operation verfügt über Input- und Output-Elemente, wobei beliebige Fault-Elemente auch hinzugefügt werden können. Als Beispiel zeigt die Abbildung. 23 zwei WSDL-Operationen. Die Operation „createDirectory“ erhält SimID, User, Host sowie Directory als Eingabe (siehe Abbildung. 24) und liefert eine Message als Ergebnis zurück.

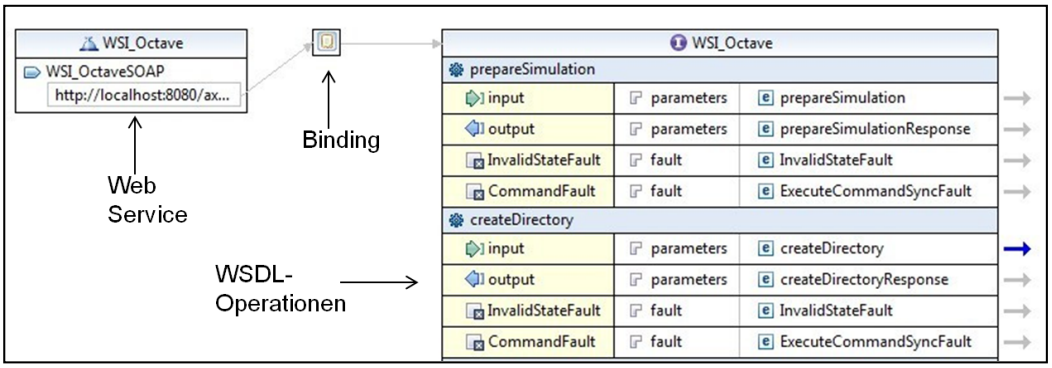


Abbildung. 23 Ansicht der WSDL-Datei im Eclipse WSDL-Editor

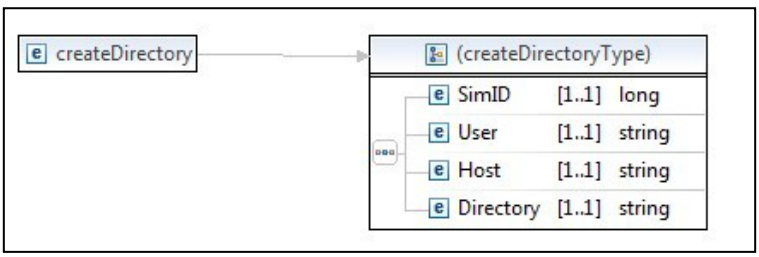


Abbildung. 24 Input-Typ der Operation “createDirectory”

7.3.3. Erstellen der Web Service-Klassen mit Axis2

Mit den WSDL-Operationen in einer WSDL-Datei kann ein Web Service erstellt werden. Dazu kommt das Tool „`wsdl2java`“ von Axis2 zum Einsatz, das im Abschnitt 3.3 vorgestellt wurde.

Durch Ausführung der Batchdatei `wsdl2java` können die Web Service-Klassen und ein Java Skeleton generiert werden. Mit unterschiedlichen Kommandos können verschiedene Java-Klassen erstellt werden, zum Beispiel die Java Stubs, die Bestandteil des Plugins für interaktive Simulationsanwendungen sind. Die durch unterschiedliche Kommandos erstellten Java-Klassen werden innerhalb des Eclipse-Projekts in verschiedenen Verzeichnissen gespeichert.

7.3.4. Erstellen eines BPEL-Prozesses

Eine Aufgabe dieser Arbeit ist es einen BPEL-Prozess zu erstellen. Dies wird durch das Erstellen eines BPEL Projects oder eines BPEL Process Files in Eclipse durchgeführt. Dabei wird vorausgesetzt, dass Apache ODE in Eclipse eingebunden ist. Hier wird die Version apache-ode-war-1.3.5 verwendet.

Die Definition eines BPEL-Prozesses setzt sich aus einem BPEL Process File sowie einer oder mehreren WSDL-Dateien zusammen. Diese WSDL-Dateien definieren sowohl die Web Service-Schnittstellen als auch die Schnittstelle, unter der der Prozess selbst erreicht werden kann. Die WSDL-Datei, die die Schnittstellen des Octave Web Services beschreiben, sind im Abschnitt 6.2 beschrieben. Für den aufzurufenden Service wird diese WSDL-Datei im BPEL-Prozess importiert.

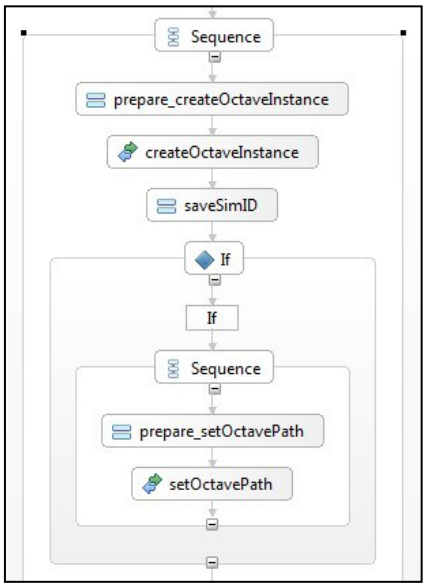


Abbildung. 25 Beispiel eines Octave-basierten Workflow-Prozesses

Um einen Web Service aufzurufen, muss eine Invoke-Aktivität erstellt werden. Sie existiert normalerweise zwischen Receive- und Reply-Aktivitäten, die die eingehende Nachricht empfängt bzw. die Antwortnachricht an den Aufrufer zurücksendet. Über die Eigenschaften des Invokes kann man einen Partnerlink sowie die verwendete Operation angeben, über den der Service aufgerufen werden kann.

Ein Beispiel des Octave-basierten BPEL-Prozesses ist in der Abbildung. 25 zu sehen. Ein Sequence-Konstrukt des Prozesses wird dargestellt, mit dem alle in diesem Konstrukt enthaltenen Aktivitäten sequenziell abgearbeitet werden. Im Beispiel wird zuerst eine Assign-Aktivität ausgeführt, die normalerweise durch ein Task oder ein Sub-Process dargestellt wird. Hier wird sie dazu verwendet, einen neuen Wert den Variablen zuzuweisen. Ein Request der Operation „*prepareSimulation*“ wird der Assign-Aktivität zugewiesen. Im nächsten Schritt folgt eine Invoke-Aktivität, die die Operation „*prepareSimulation*“ des Octave Web Services aufruft. In der Abbildung sieht man noch eine If-Aktivität, die eine bedingte Ausführung der Aktivitäten ermöglicht. Optional kann diese Aktivität mit einer Elseif- oder Else-Aktivität verbunden sein. Im Beispiel werden die If-Aktivität und ihre nachfolgenden Aktivitäten nur ausgeführt, nur wenn die definierte Bedingung erfüllt ist.

7.3.5. Verwendete Verzeichnisse auf dem Server

Java System Properties werden von dem Web Service Interface verwendet, um bestimmte Einstellungen dynamisch zu ändern. Diese System Properties werden immer erst ausgelesen, wenn sie benötigt werden. In dieser Arbeit beschreiben die System Properties drei Pfade zu Verzeichnissen (siehe Tabelle 11), auf die während der Laufzeit zugegriffen werden muss.

Pfad	Beschreibung
/srv/wsi/octave/instance	Der absolute Pfad des Instanzverzeichnisses
/srv/wsi/octave/src	Der absolute Pfad des Verzeichnisses, in dem die Dateien der Octave-Anwendungen gespeichert werden.
/srv/wsi/octave/export	Der absolute Pfad des Verzeichnisses, in das die Ergebnis-Dateien exportiert werden können.

Tabelle 11 System Properties

7.3.6. Test

Während der Implementierung kann man den erstellten Octave Web Service testen. Z.B. kann man den Web Service unter Tomcat-Server laufen lassen, damit man sieht, welcher Service und welche Service-Operationen zur Verfügung zu stehen.

Zum Testen der Web Service Operationen wird das Werkzeug SoapUI verwendet. Man kann die WSDL-Datei in den SoapUI Client laden, um die entsprechenden Funktionen zu testen.

Bei jeder auf SoapUI importierten WSDL-Operation steht einen entsprechender Eintrag. Durch die Aktivierung dieses Eintrags wird eine Anfrage an den Server geschickt und man kann sehen, ob die Anfrage den Server erreicht.

Die Simulation soll ohne Interaktion mit dem Octave Adapter ausgeführt werden. Bei der Ausführung der Octave-Anwendungen kann man testen, ob alle Octave-Pakete installiert sind. Für die Visualisierung des Ergebnisses wird noch ein Xming-Server benötigt. Die notwendigen Ergebnis-Dateien werden in der Datenbank gespeichert. Dann werden die modifizierten Anwendungen durchgeführt, die über den Octave Adapter kontrolliert werden. Man kann die Ergebnisdateien und die Ausführungszeit in beiden Simulationsabläufe vergleichen.

8. Laufzeitumgebung

In diesem Kapitel wird eine Laufzeitumgebung dargestellt, in der alle benötigten Softwares enthalten sind. Zunächst wird eine virtuelle Maschine erläutert, mit der eine Arbeit in einer Unix-Umgebung unter Windows ermöglicht. Anschließend befindet sich eine Beschreibung über die Interaktion zwischen Client und Laufzeitumgebung.

8.1. Virtualisierte Komponenten

Im Rahmen dieser Arbeit wurden die Octave-Anwendungen und der auf Web Service Interface basierte Octave Service Adapter in einem unix-artigen Betriebssystem entwickelt. Dazu wurde eine Laufzeitumgebung in Form einer virtuellen Maschine verwendet, in der Octave und alle notwendigen Softwares installiert und lauffähig sind.

Diese Umgebung wird mit einem VMware Player[16] erstellt, der als eine Software zur vollständigen Visualisierung verwendet wird und ermöglicht, mehrere Betriebssysteme auf einem Rechner auszuführen oder eine isolierte virtuelle Maschine zu erstellen. Hierbei wird ein virtuelles Ubuntu in Form von einer ISO[17]-Abbild-Datei im VMware Player importiert. Zudem wird ein virtuelles Laufwerk mit einer Datei Ubuntu.vmdk erstellt. Zur Konfiguation des Ubuntus und zur Verbindung mit dem VMware Player wird eine Datei Ubuntu.vmx benötigt, die mit einem Texteditor geöffnet werden kann und beispielsweise die Speichergröße des Ubuntus ändert

Harware-Komponente:	**Einstellung:**
Version	Workstation 6,5-7,x virtual machine
Hauptspeicher	512 MB
Prozessoren	2
Festplatte	20 GB
Betriebssystem	Ubuntu 10.04
Netzwerke-Adapter	Network Address Translation (NAT)

Tabelle 12 Hardware-Komponenten der Laufzeitumgebung

[16] VMware Player :http://www.vmware.com/de/products/desktop_virtualization/player/overview

[17] ISO: http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=17505

Ein Grund für die Verwendung des Ubuntus liegt an einer kostenlosen und auf Debian[18] basierten Linux-Distribution[19], die sich sehr leicht an spezielle Bedürfnisse anpassen lässt. Während der mehrmals grundlegenden Überarbeitung von GNU Octave wurde es in Debian oder Linux integriert. Daher kann das Debian- und Linux-basierte Ubuntu bei der Entwicklung einer Octave- Anwendung eine gute Umgebung anbieten.

Tabelle 12 zeigt die Hardware-Komponenten, die das Profil des Ubuntus darstellen. Zudem wurden noch folgende Softwares installiert, die zur Erstellung des Octave Service Adapters sowie zur Ausführung der Octave-Anwendung benötigt werden:

- GNU Octave 3.6.1 mit dem Pakete Multicore
- Eclipse IDE for Java EE Developers mit Axis2, Apache Tomcat und Apache ODE
- Java Runtime Environment 1.6

Benutzung des Betriebssystems

Die erstelle virtuelle Maschine bietet die Möglichkeit, das Betriebssystem Ubuntu unter Windows direkt zu verwenden und die sämtliche Laufzeitumgebung zu modifizieren und zu verwalten. Daher werden gewisse Grundkenntnisse über die Verwendung des Betriebssystems in Anspruch genommen. Im Folgenden wird ein Überblick über die Verwendung und oft verwendete Befehle von Ubuntu gegeben.

- Der VMware Player muss zunächst gestartet werden, um das importierte Betriebssystem Ubuntu auszuführen. Dann erfolgt das Starten des Ubuntus, wobei normalerweise ein Benutzername und ein Passwort benötigt werden. Schließlich kann man einfach auf die Taste „log out“ klicken, um das Ubuntus zu stoppen.

- Ein Vorteil von Ubuntu ist, dass die Software-Programme gleich auf ihrem Rechner installiert werden und ihr System ausprobieren können, ohne stundenlang nach den passenden Programmen suchen zu müssen. Dies wird durch eine Konsole `apt-get` im Terminal realisiert, die ermöglicht, die Anwendung zu installieren und zu aktualisieren. Weiterhin können auch alle unnötigen Komponenten durch Konsole gleich weggelassen werden, um den Speicherplatz zu sparen.

8.2. Interaktion mit der Laufzeitumgebung

In der Arbeit [23] wurden drei Benutzungsmöglichkeiten dargestellt, die eine Interaktion zwischen den Client-Rechner, den WfMS-Server und den (virtuellen) Rechner der

[18] Debian: http://www.debian.org/intro/about

[19] Linux-Distribution: http://distrowatch.com/dwres.php?resource=popularity

Laufzeitumgebung beschreiben. Im Rahmen dieser Arbeit erfolgt nur eine Interaktion, die sich in dem Client und in der Laufzeitumgebung befindet. Hierbei wird ein Überblick in der Abbildung. 26 verschafft.

In der Laufzeitumgebung werden folgende Komponenten enthalten: ein Workflow-Prozess (hier der BPEL-Prozess), das Web Service Interface, der erstellte Octave Service Adapter und die Octave-Simulationsanwendungen. Zunächst kommuniziert der Client über Web Service mit dem BPEL-Prozess in der Laufzeitumgebung. Dies wird durch Versendung einer Anfrage des Client realisiert. Nach Empfangen der Anfrage geht der BPEL-Prozess zur Ausführung los, der in der BPEL-Engine *Apache ODE* bereitgestellt und eine automatisierte Ausführung der Simulation ermöglicht. Ein BPEL-Prozess besteht aus verschiedenen Aktivitäten, die sich mit unterschiedlichen Kontrollen verbinden lassen. Zur Ausführung der Simulationsanwendungen über Web Services muss die Web Service-Operation in der entsprechenden Aktivität des BPEL-Prozesses integriert werden. Dadurch werden die Web Service-Operationen während der Durchführung des BPEL-Prozess aufgerufen und die Octave-Anwendungen ausgeführt.

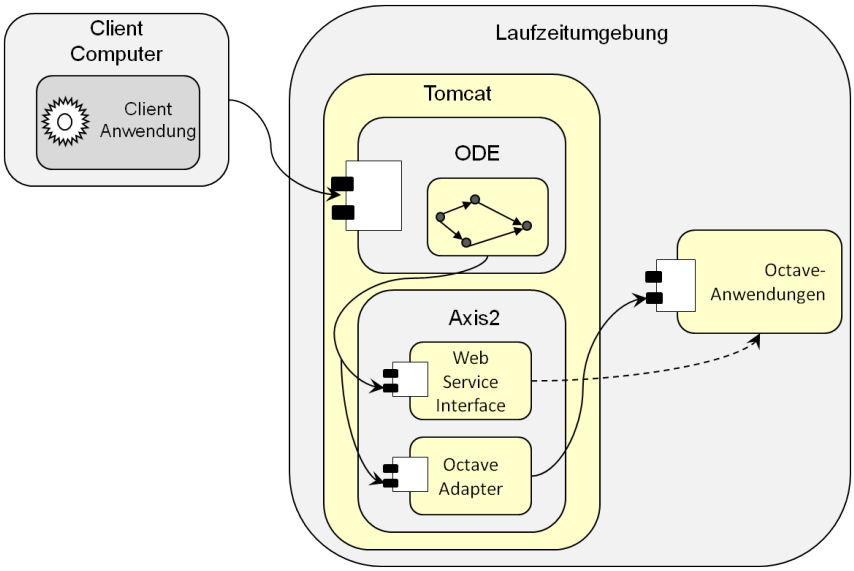


Abbildung. 26 Kommunikation in der Laufzeitumgebung [23]

9. Zusammenfassung und Ausblick

Aufgrund der Berechnungen der kontinuierlich wachsenden und heterogenen Datenmengen wird eine Zusammenarbeit zwischen Simulationen, Workflow- und Web Service Technologien entwickelt. Dies ermöglicht, bereits definierte Funktionalitäten der Simulation in Form von Web Services über das Internet zu integrieren. Ein Vorteil der Web Services ist, dass sie von einer gewissen Struktur verwendet werden können, die oft mittels sogenanntes Workflow-Prozess definiert wird. Um Orchestrierungen der Web Service zu definieren und ausführbar zu machen, wird die Workflow-Technologie eingeführt.

Im Rahmen der vorliegenden Arbeit wurde eine Octave-basierte Simulationsanwendung betrachtet. Es wurde zunächst ein Octave Service Adapter auf Basis eines plattformunabhängig generischen Web Service Interface aufgebaut. Dieser Adapter wurde als Web Service Plugin angesehen. Weiterhin verfügt dieser Adapter eigene Simulationsinstanz, die stets eine eindeutige Identifikationsnummer besitzt und eine Menge von Verzeichnissen und Daten umfasst sowie Simulationsanwendungen ausführt. Einerseits bietet der Adapter die Kommunikation mit den Anwendungen an. Hierfür wurde ein Web Service zur Beschreibung der Funktionalitäten bei ihren Anwendungen erstellt. Andererseits stellt er einen weiteren Web Service für den Client zur Verfügung. Ein Callback Stub wurde hierbei bereitgestellt, um den Client mitzuteilen, dass die Anwendungen bereit sind und die Anfrage des Clients bearbeitet werden kann.

Anschließend wurde ein BEPL-Prozess mit Hilfe des Octave Adapters aufgebaut. Dieser Prozess dient dazu, den Ablauf für Ausführung der Octave-Anwendungen darzustellen. Die Anwendungen können dann in einem BPEL-Prozess koordiniert werden, wenn die Aktivitäten des Prozesses eine Verbindung mit den Web Service-Operationen gesetzt wurden. Anschließend wurde der BPEL-Prozess durch die Verwendung von BPEL-Engine automatisiert ausgeführt.

Die Simulationsanwendungen und das Web Service Interface wurden unter Unix-artigen Betriebssystemen definiert. Aus diesem Grund wurde eine Laufzeitumgebung *Ubuntu* in einer virtuellen Maschine *VMware Player* erstellt. In dieser Umgebung wurden alle in dieser Arbeit verwendeten Software installiert und Programme ausgeführt. Der Web Service und seine WSDL-Operationen wurden mittels *Apache Tomcat* und *Axis2* in Eclipse erstellt. Die Ausführung des BPEL-Prozesses wurde anhand der BPEL-Engine *Apache ODE* realisiert.

9. Zusammenfassung und Ausblick

Die Integration der Workflow- und Web Service Technologie ermöglicht weitere anwendungsorientierte Aufgaben. Beispielsweise können mehrere Web Service-Operationen bereitgestellt werden, um mehrere Funktionalitäten der Simulationsanwendung zu beschreiben. Die Simulationen, die aus verschiedenen Plattformen erstellt werden, können in einem Workflow System ausgeführt werden. Weiterhin können Workflow-Prozesse asynchron durchgeführt werden, damit mehr Flexibilität bei der Änderung und bei der Ausführung der Prozesse erzeugt werden kann

Anhang

WSDL-Operationen von Octave Web Service

```
1.  <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2.  <wsdl:definitions name="WSI_Octave"
3.   targetNamespace="http://wsi.simtech.de/WSI_Octave/"
4.   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
5.   xmlns:tns="http://wsi.simtech.de/WSI_Octave/"
6.   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
7.   xmlns:types="http://wsi.simtech.de/ws/types/"
8.   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
9.
10.     <wsdl:types>
11.         <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
12.             <xsd:import namespace="http://wsi.simtech.de/ws/types/"
13.                 schemaLocation="types.xsd">
14.             </xsd:import>
15.         </xsd:schema>
16.         <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
17.                 targetNamespace="http://wsi.simtech.de/WSI_Octave/">
18.
19.         <xsd:element name="prepareSimulation">
20.             <xsd:complexType>
21.                 <xsd:sequence>
22.                     <xsd:element name="name" type="xsd:string" minOccurs="1"
    maxOccurs="1"></xsd:element>
23.                 </xsd:sequence>
24.             </xsd:complexType>
25.         </xsd:element>
26.
27.         <xsd:element name="prepareSimulationResponse">
28.             <xsd:complexType>
29.                 <xsd:sequence>
30.                     <xsd:element name="retrunMessage"
31.                         type="xsd:string" minOccurs="1" maxOccurs="1">
```

```
32.                         </xsd:element>
33.
34.                         <xsd:element name="SimID"
35.                             type="xsd:long" minOccurs="1" maxOccurs="1">
36.                         </xsd:element>
37.                     </xsd:sequence>
38.                 </xsd:complexType>
39.             </xsd:element>
40.
41.             <xsd:element name="ExecuteCommandSyncFault"
42.                 type="tns:ExecuteCommandSyncFaultType">
43.             </xsd:element>
44.
45.             <xsd:complexType name="ExecuteCommandSyncFaultType">
46.                 <xsd:sequence>
47.                     <xsd:element name="returnCode" type="xsd:int"
48.                         minOccurs="1" maxOccurs="1">
49.                     </xsd:element>
50.                     <xsd:element name="errorMessage" type="xsd:string" minOccurs="1"
    maxOccurs="1"></xsd:element>
51.                 </xsd:sequence>
52.             </xsd:complexType>
53.
54.             <xsd:element name="createDirectory">
55.                 <xsd:complexType>
56.                     <xsd:sequence>
57.                         <xsd:element name="SimID" type="xsd:long"
58.                             minOccurs="1" maxOccurs="1">
59.                         </xsd:element>
60.                         <xsd:element name="User" type="xsd:string"
61.                             minOccurs="1" maxOccurs="1">
62.                         </xsd:element>
63.                         <xsd:element name="Host" type="xsd:string"
64.                             minOccurs="1" maxOccurs="1">
65.                         </xsd:element>
66.                         <xsd:element name="Directory"
67.                             type="xsd:string" minOccurs="1" maxOccurs="1">
68.                         </xsd:element>
69.                     </xsd:sequence>
70.                 </xsd:complexType>
71.             </xsd:element>
72.
```

```
73.             <xsd:element name="createDirectoryResponse">
74.                 <xsd:complexType>
75.                     <xsd:sequence>
76.                         <xsd:element name="returnMessage" type="xsd:string"
    minOccurs="1" maxOccurs="1"></xsd:element>
77.                     </xsd:sequence>
78.                 </xsd:complexType>
79.             </xsd:element>

80.             <xsd:element name="copyFile">
81.                 <xsd:complexType>
82.                     <xsd:sequence>
83.                         <xsd:element name="SimID" type="xsd:long"
84.                             minOccurs="1" maxOccurs="1">
85.                         </xsd:element>
86.                         <xsd:element name="SrcUser" type="xsd:string"
87.                             minOccurs="1" maxOccurs="1">
88.                         </xsd:element>
89.                         <xsd:element name="SrcHost" type="xsd:string"
90.                             minOccurs="1" maxOccurs="1">
91.                         </xsd:element>
92.                         <xsd:element name="SrcFile" type="xsd:string"
93.                             minOccurs="1" maxOccurs="1">
94.                         </xsd:element>
95.                         <xsd:element name="DstUser" type="xsd:string"
96.                             minOccurs="1" maxOccurs="1">
97.                         </xsd:element>
98.                         <xsd:element name="DstHost" type="xsd:string"
99.                             minOccurs="1" maxOccurs="1">
100.                        </xsd:element>
101.                        <xsd:element name="DstFile"
102.                            type="xsd:string" minOccurs="1" maxOccurs="1">
103.                        </xsd:element>
104.                    </xsd:sequence>
105.                </xsd:complexType>
106.            </xsd:element>
107.
108.            <xsd:element name="copyFileResponse">
109.                <xsd:complexType>
110.                    <xsd:sequence>
111.                        <xsd:element name="returnMessage" type="xsd:string"
    minOccurs="1" maxOccurs="1"></xsd:element>
```

```
112.                    </xsd:sequence>
113.                </xsd:complexType>
114.            </xsd:element>
115.
116.            <xsd:element name="setOctavePath">
117.                <xsd:complexType>
118.                    <xsd:sequence>
119.                        <xsd:element name="SimID" type="xsd:long"
120.                            minOccurs="1" maxOccurs="1">
121.                        </xsd:element>
122.                        <xsd:element name="OctavePath"
123.                            type="xsd:string" minOccurs="1" maxOccurs="1">
124.                        </xsd:element>
125.                    </xsd:sequence>
126.                </xsd:complexType>
127.            </xsd:element>
128.
129.            <xsd:element name="setOctavePathResponse">
130.                <xsd:complexType>
131.                    <xsd:sequence>
132.                        <xsd:element name="returnMessage" type="xsd:string"
    minOccurs="1" maxOccurs="1"></xsd:element>
133.                    </xsd:sequence>
134.                </xsd:complexType>
135.            </xsd:element>
136.
137.            <xsd:element name="getOctavePath">
138.                <xsd:complexType>
139.                    <xsd:sequence>
140.                        <xsd:element name="SimID" type="xsd:long" minOccurs="1"
    maxOccurs="1"></xsd:element>
141.                    </xsd:sequence>
142.                </xsd:complexType>
143.            </xsd:element>
144.
145.            <xsd:element name="getOctavePathResponse">
146.                <xsd:complexType>
147.                    <xsd:sequence>
148.                        <xsd:element name="OctavePath" type="xsd:string" minOccurs="1"
    maxOccurs="1"></xsd:element>
149.                    </xsd:sequence>
150.                </xsd:complexType>
```

```
151.        </xsd:element>
152.
153.        <xsd:element name="startProgram">
154.            <xsd:complexType>
155.                <xsd:sequence>
156.                    <xsd:element name="SimID" type="xsd:long"
157.                        minOccurs="1" maxOccurs="1">
158.                    </xsd:element>
159.                    <xsd:element name="User" type="xsd:string"
160.                        minOccurs="1" maxOccurs="1">
161.                    </xsd:element>
162.                    <xsd:element name="Host" type="xsd:string"
163.                        minOccurs="1" maxOccurs="1">
164.                    </xsd:element>
165.                    <xsd:element name="Path" type="xsd:string"
166.                        minOccurs="1" maxOccurs="1">
167.                    </xsd:element>
168.                    <xsd:element name="Program"
169.                        type="xsd:string" minOccurs="1" maxOccurs="1">
170.                    </xsd:element>
171.                </xsd:sequence>
172.            </xsd:complexType>
173.        </xsd:element>
174.
175.        <xsd:element name="startProgramResponse">
176.            <xsd:complexType>
177.                <xsd:sequence>
178.                    <xsd:element name="plotSample"
179.                        type="tns:plot">
180.                    </xsd:element>
181.                    <xsd:element name="DataFile1"
182.                        type="xsd:string">
183.                    </xsd:element>
184.                </xsd:sequence>
185.            </xsd:complexType>
186.        </xsd:element>
187.
188.            <xsd:element name="setPlottingPath">
189.            <xsd:complexType>
190.                <xsd:sequence>
191.                    <xsd:element name="SimID" type="xsd:long"></xsd:element>
192.                    <xsd:element name="PlottingPath"
```

```
193.                                type="xsd:string">
194.                            </xsd:element>
195.                        </xsd:sequence>
196.                    </xsd:complexType>
197.                </xsd:element>
198.
199.                <xsd:element name="setPlottingPathResponse">
200.                    <xsd:complexType>
201.                        <xsd:sequence>
202.                            <xsd:element name="returnMessage" type="xsd:string"
    minOccurs="1" maxOccurs="1"></xsd:element>
203.                        </xsd:sequence>
204.                    </xsd:complexType>
205.                </xsd:element>
206.
207.                <xsd:element name="getPlottingPath">
208.                    <xsd:complexType>
209.                        <xsd:sequence>
210.                            <xsd:element name="SimID" type="xsd:long" minOccurs="1"
    maxOccurs="1"></xsd:element>
211.                        </xsd:sequence>
212.                    </xsd:complexType>
213.                </xsd:element>
214.
215.                <xsd:element name="getPlottingPathResponse">
216.                    <xsd:complexType>
217.                        <xsd:sequence>
218.                            <xsd:element name="returnMessage" type="xsd:string"
    minOccurs="1" maxOccurs="1"></xsd:element>
219.                        </xsd:sequence>
220.                    </xsd:complexType>
221.                </xsd:element>
222.
223.                <xsd:element name="removeFile">
224.                    <xsd:complexType>
225.                        <xsd:sequence>
226.                            <xsd:element name="SimID" type="xsd:long"
227.                                minOccurs="1" maxOccurs="1">
228.                            </xsd:element>
229.                            <xsd:element name="User" type="xsd:string"
230.                                minOccurs="1" maxOccurs="1">
231.                            </xsd:element>
```

```
232.                            <xsd:element name="Host" type="xsd:string"
233.                                 minOccurs="1" maxOccurs="1">
234.                            </xsd:element>
235.                            <xsd:element name="Path"
236.                                 type="xsd:string" minOccurs="1" maxOccurs="1">
237.                            </xsd:element>
238.                        </xsd:sequence>
239.                    </xsd:complexType>
240.                </xsd:element>
241.
242.                <xsd:element name="removeFileResponse">
243.                    <xsd:complexType>
244.                        <xsd:sequence>
245.                            <xsd:element name="returnMessage" type="xsd:string"
    minOccurs="1" maxOccurs="1"></xsd:element>
246.                        </xsd:sequence>
247.                    </xsd:complexType>
248.                </xsd:element>
249.            </xsd:schema>
250.        </wsdl:types>
251.
252.    <wsdl:message name="prepareSimulationRequest">
253.        <wsdl:part name="parameters" element="tns:prepareSimulation"></wsdl:part>
254.    </wsdl:message>
255.
256.    <wsdl:message name="prepareSimulationResponse">
257.        <wsdl:part name="parameters"
    element="tns:prepareSimulationResponse"></wsdl:part>
258.    </wsdl:message>
259.
260.    <wsdl:message name="InvalidStateException">
261.        <wsdl:part name="fault" element="types:InvalidStateFault"></wsdl:part>
262.    </wsdl:message>
263.
264.    <wsdl:message name="ExecuteCommandException">
265.        <wsdl:part name="fault" element="tns:ExecuteCommandSyncFault"></wsdl:part>
266.    </wsdl:message>
267.
268.    <wsdl:message name="createDirectoryRequest">
269.        <wsdl:part name="parameters" element="tns:createDirectory"></wsdl:part>
270.    </wsdl:message>
271.
```

```
272.    <wsdl:message name="createDirectoryResponse">
273.       <wsdl:part name="parameters" element="tns:createDirectoryResponse"></wsdl:part>
274.    </wsdl:message>
275.
276.    <wsdl:message name="copyFileRequest">
277.       <wsdl:part name="parameters" element="tns:copyFile"></wsdl:part>
278.    </wsdl:message>
279.
280.    <wsdl:message name="copyFileResponse">
281.       <wsdl:part name="parameters" element="tns:copyFileResponse"></wsdl:part>
282.    </wsdl:message>
283.
284.    <wsdl:message name="setOctavePathRequest">
285.       <wsdl:part name="parameters" element="tns:setOctavePath"></wsdl:part>
286.    </wsdl:message>
287.
288.    <wsdl:message name="setOctavePathResponse">
289.       <wsdl:part name="parameters" element="tns:setOctavePathResponse"></wsdl:part>
290.    </wsdl:message>
291.
292.    <wsdl:message name="getOctavePathRequest">
293.       <wsdl:part name="parameters" element="tns:getOctavePath"></wsdl:part>
294.    </wsdl:message>
295.
296.    <wsdl:message name="startProgramRequest">
297.       <wsdl:part name="parameters" element="tns:startProgram"></wsdl:part>
298.    </wsdl:message>
299.
300.    <wsdl:message name="startProgramResponse">
301.       <wsdl:part name="parameters" element="tns:startProgramResponse"></wsdl:part>
302.    </wsdl:message>
303.
304.    <wsdl:message name="setPlottingPathRequest">
305.       <wsdl:part name="parameters" element="tns:setPlottingPath"></wsdl:part>
306.    </wsdl:message>
307.
308.    <wsdl:message name="getPlottingPathRequest">
309.       <wsdl:part name="parameters" element="tns:getPlottingPath"></wsdl:part>
310.    </wsdl:message>
311.
312.    <wsdl:message name="getPlottingPathResponse">
313.       <wsdl:part name="parameters" element="tns:getPlottingPathResponse"></wsdl:part>
```

```
314.    </wsdl:message>
315.
316.    <wsdl:message name="removeFileRequest">
317.        <wsdl:part name="parameters" element="tns:removeFile"></wsdl:part>
318.    </wsdl:message>
319.
320.    <wsdl:message name="removeFileResponse">
321.        <wsdl:part name="parameters" element="tns:removeFileResponse"></wsdl:part>
322.    </wsdl:message>
323.
324.    <wsdl:portType name="WSI_Octave">
325.
326.            <wsdl:operation name="prepareSimulation">
327.             <wsdl:input message="tns:prepareSimulationRequest"></wsdl:input>
328.             <wsdl:output message="tns:prepareSimulationResponse"></wsdl:output>
329.                 <wsdl:fault name="InvalidStateFault"
    message="tns:InvalidStateException"></wsdl:fault>
330.                 <wsdl:fault name="CommandFault"
    message="tns:ExecuteCommandException"></wsdl:fault>
331.            </wsdl:operation>
332.
333.            <wsdl:operation name="createDirectory">
334.             <wsdl:input message="tns:createDirectoryRequest"></wsdl:input>
335.             <wsdl:output message="tns:createDirectoryResponse"></wsdl:output>
336.                 <wsdl:fault name="InvalidStateFault"
    message="tns:InvalidStateException"></wsdl:fault>
337.                 <wsdl:fault name="CommandFault"
    message="tns:ExecuteCommandException"></wsdl:fault>
338.            </wsdl:operation>
339.
340.            <wsdl:operation name="copyFile">
341.             <wsdl:input message="tns:copyFileRequest"></wsdl:input>
342.             <wsdl:output message="tns:copyFileResponse"></wsdl:output>
343.                 <wsdl:fault name="InvalidStateFault"
    message="tns:InvalidStateException"></wsdl:fault>
344.                 <wsdl:fault name="CommandFault"
    message="tns:ExecuteCommandException"></wsdl:fault>
345.            </wsdl:operation>
346.
347.            <wsdl:operation name="setOctavePath">
348.             <wsdl:input message="tns:setOctavePathRequest"></wsdl:input>
349.             <wsdl:output message="tns:setOctavePathResponse"></wsdl:output>
```

```
350.                <wsdl:fault name="InvalidStateFault"
    message="tns:InvalidStateException"></wsdl:fault>
351.                <wsdl:fault name="CommandFault"
    message="tns:ExecuteCommandException"></wsdl:fault>
352.            </wsdl:operation>
353.
354.            <wsdl:operation name="getOctavePath">
355.             <wsdl:input message="tns:getOctavePathRequest"></wsdl:input>
356.             <wsdl:output message="tns:getOctavePathResponse"></wsdl:output>
357.                <wsdl:fault name="InvalidStateFault"
    message="tns:InvalidStateException"></wsdl:fault>
358.                <wsdl:fault name="CommandFault"
    message="tns:ExecuteCommandException"></wsdl:fault>
359.            </wsdl:operation>
360.
361.            <wsdl:operation name="startProgram">
362.             <wsdl:input message="tns:startProgramRequest"></wsdl:input>
363.             <wsdl:output message="tns:startProgramResponse"></wsdl:output>
364.                <wsdl:fault name="InvalidStateFault"
    message="tns:InvalidStateException"></wsdl:fault>
365.                <wsdl:fault name="CommandFault"
    message="tns:ExecuteCommandException"></wsdl:fault>
366.            </wsdl:operation>
367.
368.            <wsdl:operation name="setPlottingPath">
369.             <wsdl:input message="tns:setPlottingPathRequest"></wsdl:input>
370.             <wsdl:output message="tns:setPlottingPathResponse"></wsdl:output>
371.                <wsdl:fault name="InvalidStateFault"
    message="tns:InvalidStateException"></wsdl:fault>
372.                <wsdl:fault name="CommandFault"
    message="tns:ExecuteCommandException"></wsdl:fault>
373.            </wsdl:operation>
374.
375.            <wsdl:operation name="getPlottingPath">
376.             <wsdl:input message="tns:getPlottingPathRequest"></wsdl:input>
377.             <wsdl:output message="tns:getPlottingPathResponse"></wsdl:output>
378.                <wsdl:fault name="InvalidStateFault"
    message="tns:InvalidStateException"></wsdl:fault>
379.                <wsdl:fault name="CommandFault"
    message="tns:ExecuteCommandException"></wsdl:fault>
380.            </wsdl:operation>
381.
```

```
382.            <wsdl:operation name="removeFile">
383.             <wsdl:input message="tns:removeFileRequest"></wsdl:input>
384.             <wsdl:output message="tns:removeFileResponse"></wsdl:output>
385.            </wsdl:operation>
386.    </wsdl:portType>
387.
388.    <wsdl:binding name="WSI_OctaveSOAP" type="tns:WSI_Octave">
389.      <soap:binding style="document"
390.            transport="http://schemas.xmlsoap.org/soap/http" />
391.      <wsdl:operation name="prepareSimulation">
392.            <soap:operation
393.                  soapAction="http://wsi.simtech.de/WSI_Octave/prepareSimulation" />
394.            <wsdl:input>
395.                  <soap:body use="literal" />
396.            </wsdl:input>
397.            <wsdl:output>
398.                  <soap:body use="literal" />
399.            </wsdl:output>
400.            <wsdl:fault name="InvalidStateFault">
401.                  <soap:fault use="literal" name="InvalidStateFault" />
402.            </wsdl:fault>
403.            <wsdl:fault name="CommandFault">
404.                  <soap:fault use="literal" name="CommandFault" />
405.            </wsdl:fault>
406.      </wsdl:operation>
407.      <wsdl:operation name="createDirectory">
408.            <soap:operation
409.                  soapAction="http://wsi.simtech.de/WSI_Octave/createDirectory" />
410.            <wsdl:input>
411.                  <soap:body use="literal" />
412.            </wsdl:input>
413.            <wsdl:output>
414.                  <soap:body use="literal" />
415.            </wsdl:output>
416.            <wsdl:fault name="InvalidStateFault">
417.                  <soap:fault use="literal" name="InvalidStateFault" />
418.            </wsdl:fault>
419.            <wsdl:fault name="CommandFault">
420.                  <soap:fault use="literal" name="CommandFault" />
421.            </wsdl:fault>
422.      </wsdl:operation>
423.      <wsdl:operation name="copyFile">
```

```
424.            <soap:operation
425.                soapAction="http://wsi.simtech.de/WSI_Octave/copyFile" />
426.            <wsdl:input>
427.                <soap:body use="literal" />
428.            </wsdl:input>
429.            <wsdl:output>
430.                <soap:body use="literal" />
431.            </wsdl:output>
432.            <wsdl:fault name="InvalidStateFault">
433.                <soap:fault use="literal" name="InvalidStateFault" />
434.            </wsdl:fault>
435.            <wsdl:fault name="CommandFault">
436.                <soap:fault use="literal" name="CommandFault" />
437.            </wsdl:fault>
438.        </wsdl:operation>
439.        <wsdl:operation name="setOctavePath">
440.            <soap:operation
441.                soapAction="http://wsi.simtech.de/WSI_Octave/setOctavePath" />
442.            <wsdl:input>
443.                <soap:body use="literal" />
444.            </wsdl:input>
445.            <wsdl:output>
446.                <soap:body use="literal" />
447.            </wsdl:output>
448.            <wsdl:fault name="InvalidStateFault">
449.                <soap:fault use="literal" name="InvalidStateFault" />
450.            </wsdl:fault>
451.            <wsdl:fault name="CommandFault">
452.                <soap:fault use="literal" name="CommandFault" />
453.            </wsdl:fault>
454.        </wsdl:operation>
455.        <wsdl:operation name="getOctavePath">
456.            <soap:operation
457.                soapAction="http://wsi.simtech.de/WSI_Octave/getOctavePath" />
458.            <wsdl:input>
459.                <soap:body use="literal" />
460.            </wsdl:input>
461.            <wsdl:output>
462.                <soap:body use="literal" />
463.            </wsdl:output>
464.            <wsdl:fault name="InvalidStateFault">
465.                <soap:fault use="literal" name="InvalidStateFault" />
```

```
466.            </wsdl:fault>
467.            <wsdl:fault name="CommandFault">
468.                <soap:fault use="literal" name="CommandFault" />
469.            </wsdl:fault>
470.        </wsdl:operation>
471.        <wsdl:operation name="startProgram">
472.            <soap:operation
473.                soapAction="http://wsi.simtech.de/WSI_Octave/startProgram" />
474.            <wsdl:input>
475.                <soap:body use="literal" />
476.            </wsdl:input>
477.            <wsdl:output>
478.                <soap:body use="literal" />
479.            </wsdl:output>
480.            <wsdl:fault name="InvalidStateFault">
481.                <soap:fault use="literal" name="InvalidStateFault" />
482.            </wsdl:fault>
483.            <wsdl:fault name="CommandFault">
484.                <soap:fault use="literal" name="CommandFault" />
485.            </wsdl:fault>
486.        </wsdl:operation>
487.    </wsdl:binding>
488.    <wsdl:service name="WSI_Octave">
489.        <wsdl:port binding="tns:WSI_OctaveSOAP" name="WSI_OctaveSOAP">
490.            <soap:address location="http://localhost:8080/axis2/services/WSI_Octave"/>
491.        </wsdl:port>
492.    </wsdl:service>
493. </wsdl:definitions>
```

Abkürzungsverzeichnis

ADB: Axis2 Data Binding Framework

BPEL: Business Process Execution Language

HTTP: Hypertext Transfer Protocol

MCMC: Markov Chain Monte Carlo

NAT: Network Address Translation

NESC: National e-Science Centre

REST: Representational State Transfer

SOA: Serviceorientierte Architektur

SOAP: Simple Object Access Protocol

TCP: Transmission Control Protocol

UDDI: Universal Description, Discovery and Integration

URI: Uniform resource identifier

URL: Uniform Resource Locators

W3C: World Wide Web Consortium

WfMC: Workflow Management Coalition

WfMS: Workflow Management System

WS: Web Service

WSDL: Web Services Description Language

WSFL: Web Services Flow Language

WSI: Web Service Interface

XML: Extensible Markup Language

Abbildungsverzeichnis

0. Abbildungsverzeichnis

Tabelleverzeichnis

Literaturverzeichnis

[1]. Ingo Melz et al. : *Serviceorientierte Architekturen mit Webservice: Konzepte- Standards- Praxis,* 2010

[2]. DOSTAL, Wolfgang ; JECKLE, Mario ; MELZER, Ingo ; ZENGLER, Barbara: *Service-orientierte Architekturen mit Web Services. Konzepte – Standards – Praxis,* 2005

[3]. F. Christensen, E.; Curbera: *Web Services Description Language (WSDL) 1.1,* 2001. URL: http://www.w3.org/TR/wsdl.

[4]. *W3C. SOAP Version 1.2*, 2007. URL:http:/www.w3.org/TR/soap/.

[5]. Microsoft: *UDDI Business Registry Shutdown*, 2005.

URL: http://uddi.microsoft.com/about/FAQshutdown.htm

[6]. *Web Services Business Process Execution Language Version 2.0, 2007 URL:*http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html

[7]. BURGHARDT, Markus: *Web Services. Aspekte von Sicherheit, Transaktionalität, Abrechnung und Workflow,* 2004

[8]. Frank Leymann, Dieter Roller: *Production Workflow: Concepts and Techniques,* 2000

[9]. S. Weerawarana, F. Curbera, F. Leymann, T. Storey, D. Ferguson: *Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging and More,* 2008.

[10]. Bertram Ludäscher, Mathias Weske, Timothy McPhillips, Shawn Bowers: *Scientific Workflows: Business as Usual?* 2009

[11]. Katharina Görlach, Mirko Sonntag, Dimka Karastoyanova,Frank Leymann, Michael Reiter: *Conventional Workflow Technology for Scientific Simulation.* 2011

[12]. Rafael Ball: *Wissenschaftskommunikation der Zukunft,* 2007

[13]. Ute Rusnak, Matthias Razum, Leni Helmes: *Wissensvernetzung im Forschungsprozess*, 2007

[14]. Gabriele Gramelsberger: *Computersimulationen in den Wissenschaften- Neue Instrumente der Wissensproduktion: Schnittstellen zwischen Theorie und Experiment.* 2007

[15]. F. Steinke, M. Seeger, K. Tsuda: *Experimental design for efficient identification of gene regulatory networks using sparse bayesian models*

[16]. Andrei Kramer, Nicole Radde: *Towards experimental design using a Bayesian framework for parameter identification in dynamic intracellular network models,* 2010

[17]. Andrei Kramer, Jan Hasenauer, Frank Allgöwer, Nicole Radde: *Computation of the posterior entropy in a Bayesian framework for parameter estimation in biological networks,*

[18]. Falk Schreiber : *Analyse und Visualisierung biologischer Netzwerke,* 2009

[19]. Eric Bullinger, Dirk Fey, Marcello Farina und Rolf Findeisen: *Identification of Biochemical Reaction Networks: An Observer based Approach,* 2008

[20]. Joachim Pfister: *Integration und Modellierung von verteilten Geschäftsprozessen mittels Web Services am Beispiel eines Prozessportals,*2006

[21]. Joachim Müller *: Workflow-based Integration: Grundlagen, Technologien, Management, 2005*

[22]. Jakob Freund, Klaus Götzer: *Vom Geschäftsprozess zum Workflow: Ein Leitfaden für die Praxis*, 2008

[23]. Jens Rutschmann: *Generisches Web Service Interface um Simulationsanwendungen in BPEL-Prozesse einzubinden,* 2009

[24]. ODE- Architectural Overview: http://ode.apache.org/architectural-overview.html

[25]. David Schumm: *Graphische Modellierung von BPEL Prozessen unter Verwendung der BPMN Notation,* 2008

[26]. Enrico Serb: *Arbeitsabläufe in der Modellierung und Simulation*

[27]. Joachim Müller: *Workflow-Based Integration: Grundlagen, Technologien, Managemen,*

[28]. Brian Skibinski: *How to set up Ubuntu under VMWare Player for CS384* http://people.msoe.edu/~durant/courses/cs384/ubuntu-vmwarePlayer/

[29]. Tristan Glatard, David Emsellem, Johan Montagnat: *Generic web service wrapper for efficient embedding of legacy codes in service-based workflows*

[30]. Ingo Melzer: *Service-oritentierte Architekturen mit Web Services*

[31]. Joachim Pfister: *Integration und Modellierung von verteilten Geschäftsprozessen mittels Web Services am Beispiel eines Prozessportals,*2006

[32]. Roger J. Castaldo, Michael A. McKay, Vladimir Tosic: *Exposing GNU Octave signal processing functions as extensible markup language (XML) web services,* 2006

[33]. Wesal Al Belushi, Youcef Baghdadi: *An Approach to Wrap Legacy Applications into Web Services,* 2007

[34]. Michael Stollberg, Martin Hepp, und Dieter Fensel: *Semantic Web Services – Realisierung der SOA Vision mit semantischen Technologien,* 2007

[35]. Cornelia Boles, Jörg Friebe, Till Luhmann: *Typische Integrationsszenarien und deren Unterstützung durch Web Services und andere Technologien*

[36]. Ewa Deelmana, Dennis Gannonb, Matthew Shields, Ian Taylor: *Workflows and e-Science: An overview of workflow system features and capabilities,* 2007

[37]. Peter Reimann, Michael Reiter, Holger Schwarz, Dimka Karastoyanova, and Frank Leymann: *SIMPL – A Framework for Accessing External Data in Simulation Workflows*

[38]. Katharina Görlach, Mirko Sonntag, Dimka Karastoyanova, Frank Leymann, Michael Reiter: *Conventional Workflow Technology for Scientific Simulation*

[39]. Christoph Marian Müller: *Development of an Integrated Database Architecture for a Runtime Environment for Simulation Workflows,* 2009

Erklärung

Hiermit versichere ich, diese Arbeit selbstständig verfasst und nur die angegebenen Quellen benutzt zu haben.

Unterschrift:

< Ort, Datum >