Institut für Architektur von Anwendungssystemen

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit Nr. 101

Kapselung von bestehenden Simulationsanwendungen mit Hilfe von Web Services

Markus Nemet

Studiengang:	Informatik
Prüfer/in:	Jun.-Prof. Dr.-Ing. Dimka Karastoyanova
Betreuer/in:	M.Sc. Wirt.-Inf. Andreas Weiß
Beginn am:	8. Oktober 2013
Beendet am:	9. April 2014
CR-Nummer:	D.2.11, H.3.5, H.4.1, I.6.m

Kurzfassung

Das Exzellenzcluster Simulation Technology beschäftigt sich mit Simulationen in der Wissenschaft. In diesem Rahmen arbeitet das Institut für Architektur von Anwendungssystemen der Universität Stuttgart an einem Workflow-Management-System. Mit diesem soll es ermöglicht werden komplexe Simulationen zu erstellen. Diese Arbeit erstellt ein Webservice als Grundlage für einen wissenschaftlichen Simulations-Workflow. Dabei stellt der Webservice die Funktionalität einer bereits existierenden Simulationsanwendung, vom Institut für Materialprüfung, Werkstoffkunde und Festigkeitslehre, zur Verfügung. Später soll der Webservice in Rahmen einer weiteren Arbeit in ein Simulations-Workflow integriert werden. Es werden verschiedene Ansätze der Umsetzung als Webservice diskutiert und der Webservice mittels Java implementiert.

Inhaltsverzeichnis

Abbildungsverzeichnis

Verzeichnis der Listings

1. Einleitung

Ein großer Teil der Forschungsarbeit unserer Zeit wird mit Hilfe von rechnergestützten Experimenten unterstützt. Mit der stetig zunehmenden Rechenleistung der IT-Systeme ist es möglich immer schneller zu berechneten Ergebnissen zu kommen [M+65]. So liegt es nahe, dass auch in Zukunft in Wirtschaft und Forschung der Einsatz dieser Systeme bestehen bleibt [GKC+09].

Eine mögliche Art des Experimentes stellt die Simulation dar. Simulationen können dabei beliebig komplex gestaltet werden, so dass die Rechenleistung eines einzelnen Computer dennoch nicht ausreicht um in kurzer Zeit ein Resultat zu liefern. Daher empfiehlt es sich die Simulation auf verteilten Computern berechnen zu lassen. Hier stellt das *Cloud-Computing* eine gute Alternative dar, das sowohl die erforderliche Infrastruktur als auch die benötigte Software bereitstellen kann. Dabei muss geklärt werden wie die beteiligten Computer kommunizieren und wie Berechnungen und Ressourcen sinnvoll verteilt werden können. Des weiteren müssen entsprechende Abhängigkeiten, zum Beispiel in Form notwendiger Dateien oder Daten, auflösen und auf entsprechenden Rechnern verfügbar gemacht werden [TDGS07].

Um eine Automatisierung von Simulationen zu erreichen muss zunächst eine Möglichkeit vorhanden sein Simulationen zu modellieren. Dabei ist es unter anderem erforderlich einzelne Prozesse, komplexe Zusammenhänge sowie die zu verarbeitenden Daten und erzeugten Daten, beschreiben zu können.

In der Wirtschaft werden Geschäftsprozesse mittels Arbeitsabläufen (engl. Workflows) modelliert. Dieses Vorgehen, jedoch ohne die Betrachtung von wirtschaftlichen Kosten und Erlös, lässt sich für wissenschaftliche Simulationen übertragen. Dabei kann auf das vorhandene Wissen in Form von Referenzmodellen, Dokumenten und Standards zurückgegriffen werden. Um sich stärker von der Wirtschaft zu differenzieren und die konzeptionellen Unterschiede klarer zu machen, wird im wissenschaftlichen Kreis von Scientific Workflows oder Simulationsworkflows gesprochen [TDGS07].

Eine Möglichkeit die Geschäftsprozesse in der Informationstechnik abzubilden ist die Serviceorientierte Architektur (SOA). Die SOA ist ein Architekturmuster für den Bereich der verteilten Systeme. Sie erlaubt es Dienst von IT-Systemen zu strukturieren und zur Nutzung zur Verfügung zu stellen. Einzelne Dienste können dabei zusammengesetzt werden. Diese bilden dann eine so genannte Orchestrierung. Die Web Service-Technologie kann als eine Realisierung einer solchen SOA dienen. Die Dienste können dabei in einer abstrakten und plattformunabhängigen Art und Weise im Netzwerk zur Verfügung gestellt werden.[WCL+05]

Es gibt viele Legacy Anwendungen die nicht ohne weiteres in eine SOA integriert werden können, da diese keine native Programmierschnittstelle (API) anbieten. Die Anwendungen können dann entweder nachträglich angepasst oder mit einer anderen Anwendung als Adapter (Wrapper) umhüllt werden, die entsprechende Funktionalität zur Verfügung stellt. Dieses Vorgehen wird Kapselung genannt. Um eine Legacy Anwendung zu Kapseln sind drei Schritte notwendig. Als erstes muss die Anwendung

analysiert und gegebenenfalls von Altlasten befreit werden. Danach wird die Anwendung umhüllt und schließlich als Webservice zur Verfügung gestellt [S+06].

Diese Arbeit beschreibt anhand eines existierenden Simulationsprogrammes wie eine solche Kapselung mit Hilfe eines Webservices erreicht werden kann, um dieses später in einen Workflow aufnehmen zu können. Dabei werden alternative Umsetzungsmöglichkeiten diskutiert und die resultierende Entscheidung implementiert.

1.1. Motivation

Es gibt viele Programme die bereits aufgrund ihres Alters einen enormen Funktionsumfang haben und dabei ausgiebig getestet sind. Hier wäre es nicht wirtschaftlich diese Programme in einer neuen Technologie zu implementieren. Außerdem ist es auch fraglich wo der realistische Mehrwert, eines identischen Programms mit anderer Technologie, liegt. Da das neu implementieren viel Zeit und Geld kosten wird. Deshalb ist es wichtig diese Programme mit einer Schnittstelle auszustatten die es ermöglicht sie in neue Systeme zu integrieren. Die Schnittstelle sollte dabei Plattformunabhängig und unabhängig der gewählten Programmiersprache sein. Nur so ist gewährleistet das die Schnittstelle zukunftssicher und die Programmfunktionalität wiederverwendbar ist.

Mit Hilfe von Webservices wird erreicht das die in den Schnittstellen definierte Funktionalitäten auch über das Netzwerk erreichbar sind [WCL+05]. Dies Erlaubt, dass die Programme nicht in einer lokalen Umgebung laufen müssen. So können beispielsweise Webservices auf verschiedene Server verteilt werden und die Klienten brauchen die Programme nicht mehr selbst vorzuhalten. Die Kommunikation erfolgt dabei über Nachrichten, wobei den Klienten konkrete Implementierungsdetails unbekannt sind.

Durch diese lose Kopplung lassen sich mehrere Webservices zu Kompositionen zusammensetzen. Dies erlaubt eine flexible Ausführungsreihenfolge und automatisierte Abläufe. Mit diesem Vorgehen kann aus bestehenden Webservices auf neue Arbeitsabläufe reagiert und die Komposition entsprechend angepasst werden.

1.2. Aufgabe dieser Arbeit

Das Institut für Architektur von Anwendungssystemen (IAAS)[1] der Universität Stuttgart arbeitet an einem Workflow-Management-System (WfMs) das auf die Bedürfnisse von Wissenschaftlern zugeschnitten ist. Dieses System soll es ermöglichen komplexe Simulations-Workflows zu modellieren und auszuführen. Hierbei werden technische Details verborgen damit Wissenschaftler ohne Programmiererfahrung, sich ganz auf ihr Fachgebiet konzentrieren können. Das WfMs ist Teil des Projektes *Workflow technology in simulation, flexibility of simulation workflows*, das im Rahmen des

[1] IAAS, http://www.iaas.uni-stuttgart.de/

Exzellenzclusters SimTech[2] entstand. Das Projekt wird inzwischen in einem Folgeprojekt unter dem Namem *Modelling of Multi-Scale and Multi-Physics Simulations* fortgeführt.

Für dieses WfMs wird ein Webservice erstellt das die Funktionalität des bereits existierenden ITAP Molecular Dynamics Program[3] für das Institut für Materialprüfung, Werkstoffkunde und Festigkeitslehre (IMWF)[4] beinhaltet.

Der erstellte Webservice stellt die Grundlage für einen Simulations-Workflow dar. Später soll dieser im Rahmen einer weiteren Arbeit in ein Simulations-Workflow mittels der Business Process Execution Language (BPEL) [JEA+07] integriert werden. Diese Arbeit beschäftigt sich nur mit der Bereitstellung des Webservices und nimmt keine Integration in einen Simulations-Workflow vor.

Zunächst wird die Simulationsanwendung analysiert. Dafür wird das Vorgehen begutachtet wie mit Hilfe der IMD Bibliothek eine Simulation durchgeführt werden kann. Aus den Anforderungen des IAAS und des IMWF entsteht ein Entwurf für die Bereitstellung der Bibliothek als Webservice. Nachdem architektonische Details diskutiert worden sind, wird der Webservice mit Hilfe der Web Services Description Language (WSDL) [CCM+01] beschrieben und anschließend mittels Java implementiert.

1.3. Aufbau der Arbeit

Die Arbeit ist in folgender Weise gegliedert:

Kapitel 2 – Grundlagen: Es werden die benötigten Grundlagen erläutert um Verständnis für die verwendeten Technologien und Konzepte im Bereich der Webservices zu erhalten. Dies umfasst eine Beschreibung der aktuellen Standards sowie verwendete Methoden der Umsetzungen, um eine Anwendung als Webservice bereitzustellen. Des weiteren werden verwandte Arbeiten betrachtet die ähnliche Umsetzungen vorgenommen haben. Hier werden Gemeinsamkeiten und Unterschiede herausgearbeitet.

Kapitel 3 – Simulationsanwendung: Die bereitgestellte Simulationsanwendung, welche als Webservice gekapselt werden soll, wird genauer untersucht. In diesen Rahmen werden Funktion und Verwendung vorgestellt. Es wird dabei symbolisch die Vorgehensweise demonstriert wie eine Simulation durchgeführt werden kann. Hierbei werden die Anforderungen für den zu erstellenden Webservice erfasst.

Kapitel 4 – Umsetzung: Auf Basis der in Kapitel 3 erfassten Anforderungen wird der Webservice entworfen. Die Vor- und Nachteile unterschiedlicher Varianten werden diskutiert. Aus einer Variante entsteht anschließend eine formale Beschreibung, welche mittels Java implementiert wird. Wichtige Details der Implementierung werden genannt.

[2] SimTech, http://www.simtech.uni-stuttgart.de/
[3] IMD, http://imd.itap.physik.uni-stuttgart.de/
[4] IMWF, http://www.imwf.uni-stuttgart.de/

Kapitel 5 – Verwendung: Der Aufbau der Webservice Nachrichten wird untersucht. Weiter wird demonstriert wie die Webservices verwendet werden können um eine Simulation durchzuführen.

Kapitel 6 – Zusammenfassung und Ausblick Die Kernelemente der Arbeit werden zusammengefasst, wobei die präsentierte Lösung analysiert wird. Später wird ein Ausblick auf kommende Arbeiten gegeben.

Allgemeines

Diese Arbeit ist auf Deutsch verfasst, dennoch werden Fachbegriffe aus dem Englischen übernommen, da es oft keine adäquate Übersetzung gibt und auch andere Arbeiten die englischen Begrifflichkeiten verwenden.

Im folgenden meint ein Workflow immer implizit ein Scientific-Workflow.

2. Grundlagen

Im folgenden werden die Grundlagen aufgeführt die für das allgemeine Verständnis der verwendeten Technologien und Konzepte notwendig sind.

2.1. Service orientierte Architektur

Es gibt verschiedene Möglichkeiten wie eine Anwendung grundlegend aufgebaut werden kann. Der Begriff der Architektur lässt sich hier analog zur Architekturlehre im Bauwesen definieren. Dort umfasst eine Architektur den Bauplan eines Systems im Sinne einer Spezifikation seiner Komponenten und ihrer Beziehungen, sowie die Konstruktionsregeln für die Erstellung des Bauplans. Der Grad der Detaillierung der Architektur lässt sich dabei beliebig grob oder feingranular spezifizieren [FS12]. Diese Definition lässt sich auf die Architektur in der Softwareentwicklung übertragen.

In der Softwareentwicklung werden verschiedene Herangehensweisen an die Umsetzung unter Architekturmuster zusammengefasst. Ein Architekturmuster beschreibt dabei wie die einzelnen Komponenten unterteilt und organisiert werden, aber auch wie die Komponenten miteinander kommunizieren und interagieren können. Es wird eine High-Level Ansicht betrachtet, welche sich nicht mit den internen Details beschäftigt sondern das Zusammenspiel der einzelnen Bauteile im Fokus hat. Somit sind Architekturmuster unabhängig von der verwendeten Technologie anwendbar und liefern ein abstraktes Konzept um Softwaresysteme aufzubauen [VAC+05].

Eine Service orientierten Architektur (SOA) beschreibt ein solches Konzept. Im Mittelpunkt dieser Architektur steht dabei das Suchen, Finden und Nutzen von Diensten (engl. Services) über das Netzwerk. Dabei können Dienste plattformübergreifend von Anwendungen oder anderen Diensten verwendet werden. Die entstehende Anwendung ist meist nicht für eine Mensch-Computer Interaktion gedacht, sondern soll von anderen Computern verwendet werden [Mel10]. Das Architekturmuster gehört in die Kategorie für verteilte Systeme und wird von der *Organization for the Advancement of Structured Information Standards* (OASIS) wie folgt definiert: SOA ist ein Paradigma für die Strukturierung und Nutzung verteilter Funktionalität, die möglicherweise unter der Kontrolle von unterschiedlichen Besitzern steht[MLM+06].

In einer SOA sind drei Rollen beteiligt die nicht zwangsweise dem selben Besitzer zugeordnet werden müssen. Diese Rollen repräsentieren das Grundprinzip einer SOA und lassen sich in einem sogenannten SOA Dreieck darstellen (Abbildung 2.1)[WCL+05].

Der *Service Provider* bietet den Service an und publiziert diesen mit einer Beschreibung in der *Service Registry*. Die *Service Consumer* können nun in der *Service Registry* nach einem geeigneten Service suchen und erhalten die Adresse des Services zurückgeliefert. Mit dieser Adresse kann die konkrete

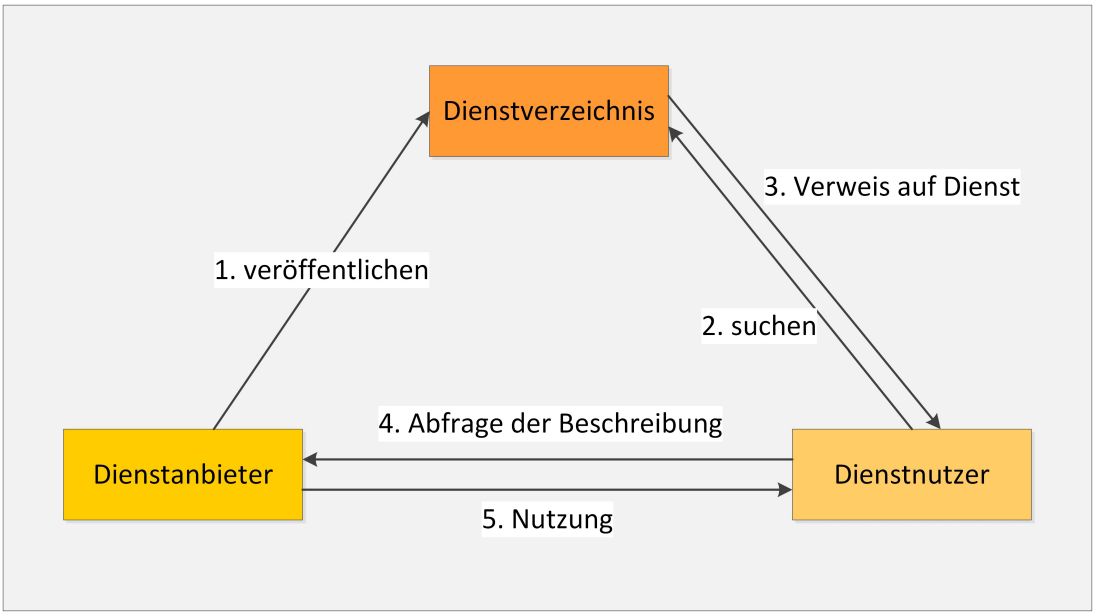


Abbildung 2.1.: SOA Dreieck nach [WCL+05]

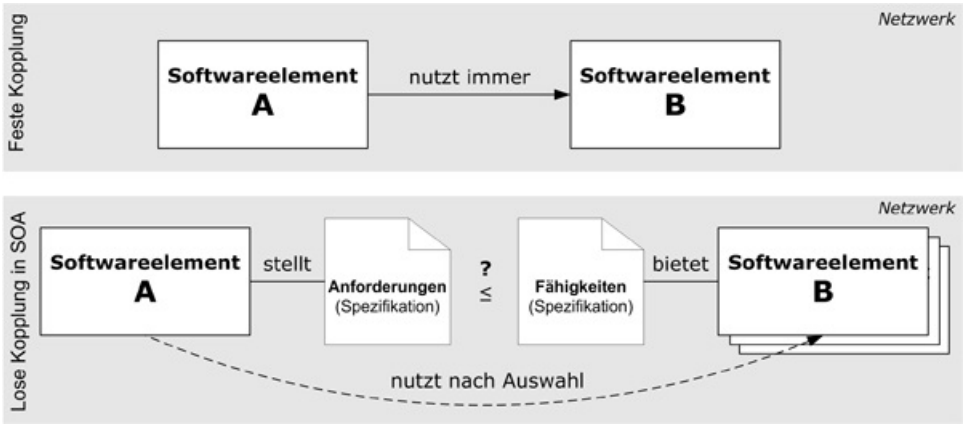


Abbildung 2.2.: Feste Kopplung vs. lose Kopplung nach [Gö12]

Schnittstellenbeschreibung vom Service beim *Service Provider* abgefragt und schließlich der Service, mit dessen Funktionalität, genutzt werden.

Eine weiteres Merkmal einer SOA ist die lose Kopplung von Diensten. Die Schritte des SOA Dreiecks um einen Service zu benutzen können jederzeit durchgeführt werden. Also auch während der Laufzeit und voll automatisch. Welcher Service dabei ausgewählt wird kann von den angeforderten Eigenschaften abhängen [Mel10]. Dies macht die Anwendung flexibel im Vergleich zu Architekturen mit fester Kopplung (Abbildung 2.2).

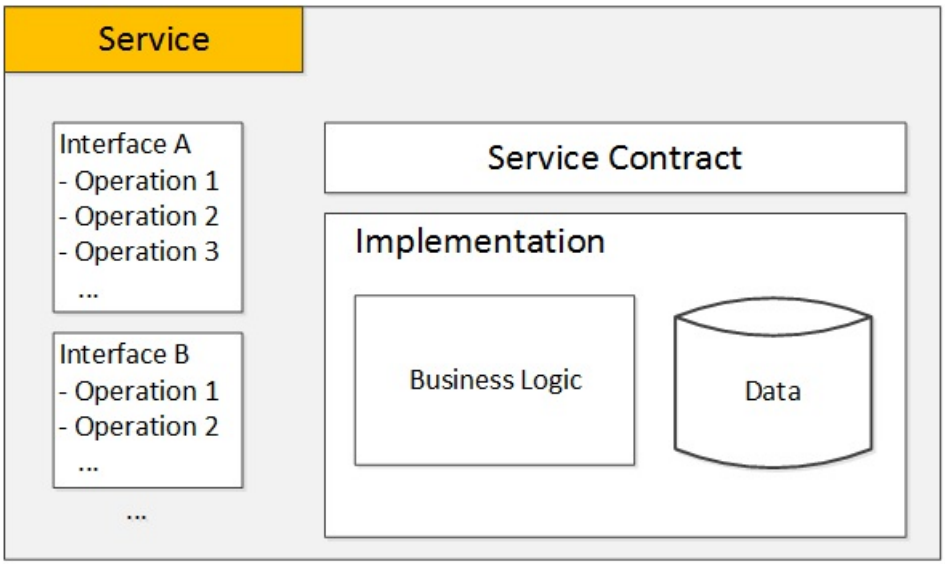


Abbildung 2.3.: Bestandteile eines Dienstes nach [KBS07]

Außer der Schnittstellenbeschreibung (Interface) besteht ein Service noch aus eine informelle Beschreibung (Service Contract) und einer Implementierung (Abbildung 2.3). Das Interface beschreibt die verfügbaren Operationen, inklusive der Eingabe und Ausgabewerte, die der Service ausführen kann. Dabei kann ein Service auch mehrere Interfaces anbieten. Weitere Beschreibungen wie die allgemeine Funktionalität, die Dokumentation und das Service-Level-Agreement (SLA) wird in dem Service Contract festgehalten. Die eigentliche Funktionalität wird mit einer Implementierung in einer beliebigen Programmiersprache umgesetzt. Dabei ist die eingesetzte Programmiersprache für den Service Consumer nicht relevant und nicht einsehbar.

Ein Service weißt folgende typische Eigenschaften auf [BKM07] [PVDH07]:

- Ein Service ist eine IT-Repräsentation von Funktionalität.
- Ein Service ist autonom.
- Ein Service ist plattformunabhängig.
- Ein Service ist in einem Netzwerk verfügbar.
- Ein Service ist transparent in der Lokalität.
- Ein Service hat eine wohldefinierte Schnittstelle.
- Ein Service unterstützt die lose Kopplung.
- Ein Service abstrahiert und versteckt interne Logik.
- Ein Service verfolgt den Zweck der Wiederverwendbarkeit.
- Ein Service steht anderen Anwendungen und Services zur Verfügung.

Eine SOA eignet sich sehr gut dazu Geschäftsprozesse abzubilden. Die einzelnen Arbeitsschritte werden dazu als Services modelliert. Durch zusammensetzen der Services kann man einen entsprechenden Workflow nachbilden, der jederzeit einfach durch ersetzten von Services modifizieren werden kann. Darüber hinaus lassen sich diese Schritte automatisieren [Mel10].

2.2. Webservice Technologie

Die technische Realisierung einer SOA (Abschnitt 2.1) kann mit unterschiedlicher Technologie erreicht werden. Die derzeit populärste Methode ist die Verwendung von Webservices. Es gilt zu beachten das die Webservice Technologie (WST) nicht automatisch eine SOA hervorbringt, da die WST auch ohne beispielsweise einen Verzeichnisdienst betrieben werden kann [WCL+05].

Ein wichtiger Bestandteil der WST ist die Extensible Markup Language (XML), welche vom World Wide Web Consortium (W3C) entwickelt wurde. Sie ist eine Metasprache für die Strukturierung von Daten. Mit ihr können neue Auszeichnungssprachen definiert werden. So bildet XML die Grundlage für weitere wichtige Standards des W3C die in der WST verwendet werden [BPSM+98].

2.2.1. Web Service Description Language

Mit der Web Service Description Language (WSDL) wird die Funktionalität des Webservice abstrahiert beschrieben. Dies erlaubt zwischen den Service Provider und Service Consumer eine Nachrichtenkommunikation die unabhängig von Nachrichtenformat und Netzwerkprotokoll ist [WCL+05].

Ein WSDL Dokument besteht aus dem XML Format und beinhaltet mindestens sechs Elemente um einen Webservice vollständig zu beschreiben. Die Elemente lassen sich dabei in abstrakte und konkrete Definitionen gruppieren. Abstrakte Elemente beschreiben die Funktionalität des Webservices wohingegen konkrete Elemente die Funktionalität an Adresse und Protokoll binden [CCM+01].

Abstrakte Elemente

Types
: Hier werden die Datentypen definiert, welche in den `message` Elementen verwendet werden. Es stehen die Standard XML Datentypen zur Verfügung, die sich zu komplexeren Typen zusammenbauen lassen.

Message
: Hier wird der Aufbau der Nachrichten beschrieben die vom Service gesendet und empfangen werden können. Eine Nachricht beinhaltet die Definition der übertragenden Daten.

Port Type
: Hier kann festgelegt werden welche Operation verfügbar sind und welches Paradigma sie folgen. Es gibt folgende Arten von Operationstypen: One-way, Request-response Solicit-response, Notification - die jeweils per Vorhandensein von Input und Output Elementen repräsentiert werden.

Konkrete Elemente

Binding
Hier wird ein konkretes Übertragungsprotokoll einen `Port Type` zugeordnet, sowie das Datenformat festgelegt. Als Protokoll wird für gewöhnlich SOAP verwendet. Des weiteren werden die verfügbaren Operationen gebunden.

Port
Hier wird die Adresse angegeben unter der die Operationen unter einen konkreten `binding` angesprochen werden können.

Service
Hier lassen sich mehrere Ports definieren, so dass der Service unter verschiedene Endpunkte erreichbar sein kann. Durch verschiedene Bindings und Ports kann der Service unterschiedliche Operationen je Endpunkt anbieten. Die außerdem andere Protokolle unterstützen könnten.

2.2.2. SOAP

Das Netzwerkprotokoll SOAP ist ein Standard des W3C und wird für die Übertragung der Nachrichten verwendet. Ursprünglich war SOAP die Abkürzung für Simple Object Access Protocol, jedoch ist seit Version 1.2 SOAP der Eigenname. Dies liegt daran das die Abkürzung nicht den eigentlichen Zweck widerspiegelt. Eine SOAP Nachricht besteht aus einem optionalen Header und einem Body, die von einen Envelope umschlossen sind. Der Envelope zeichnet die Nachricht als SOAP Nachricht aus. Im Header können Meta-Informationen mitgeschickt werden, während im Body die eigentlichen Nutzdaten stehen (Abbildung 2.4) [WCL+05].

SOAP kann als Nachrichten Framework aufgefasst werden, dass verschiedene Formate und Transportprotokolle unterstützt. Die Nachricht wird im allgemeinen im XML Format gesendet. Jedoch ist es auch möglich andere Formate wie CSV oder JSON zu verwenden. Als Transportprotokoll wird am häufigsten HTTP und TCP verwendet, aber auch hier kann beispielsweise alternativ SMTP oder JMS verwendet werden [WCL+05][ML+03].

2.2.3. Orchestration und Choreographie

Die Funktionalität von einzelnen Services lassen sich zu neuen komplexeren Service kombinieren. Dabei unterscheidet man zwei Arten von Kompositionen. Die Orchestration und eine Choreographie. Bei einer Orchestration steht eine Verwaltungseinheit im Zentrum über die einzelne Webservices kommunizieren. Das kann mit einem Dirigent verglichen werden. Hingegen bei einer Choreographie sprechen die Webservices untereinander ohne zentrale Einheit (Abbildung 2.5) [Pel03].

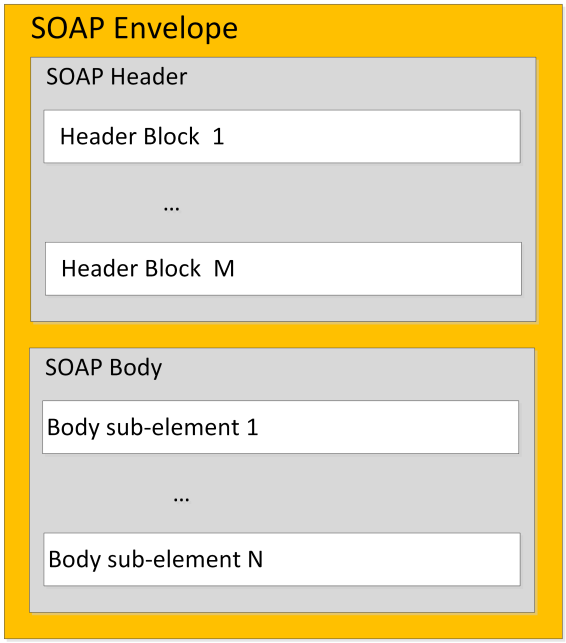


Abbildung 2.4.: SOAP Nachricht Struktur nach [WCL+05]

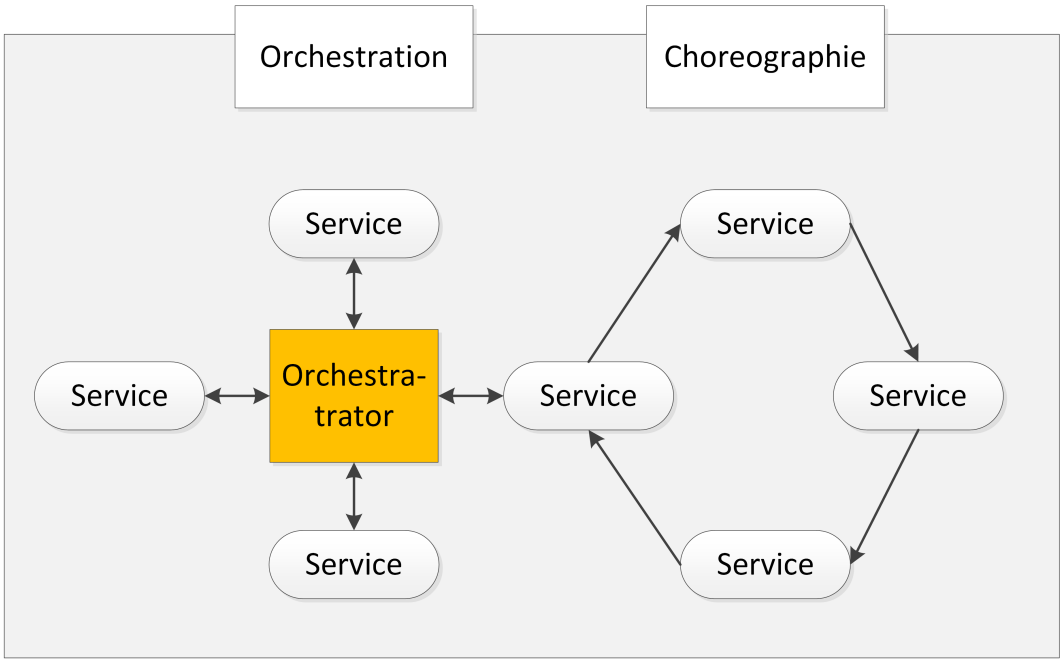


Abbildung 2.5.: Vergleich Orchestration und Choreographie nach [Mas07]

2.3. Verwandte Arbeiten

Es existieren eine Menge Veröffentlichungen darüber wie sich Legacy Systeme migrieren lassen. Viele nutzen ähnliche Ansätze, weshalb hier stellvertretend zwei genannt werden. Es steht bei diesen jedoch nicht die Integration in einen Scientific Workflow im Mittelpunkt. Auch im Rahmen des Projektes des Exzellenzclusters SimTech sind bereits zwei Diplomarbeiten entstanden die sich mit der Kapselung von bestehenden Simulationsanwendungen mit Hilfe von Webservices beschäftigten. Deren Intention war es die Webservice in einen Scientific Workflow verfügbar zu machen.

2.3.1. Migrating Interactive Legacy Systems To Web Services

In dieser Arbeit von Canfora et al. wird eine Anwendung mit einem Wrapper versehen um das Legacy System als Webservice bereitzustellen. Da es sich um eine interactive Anwendung mit Rückfragen handelt muss es möglich sein Ein- und Ausgaben der Anwendung zum Benutzer durchzureichen. Dazu wird im Webservice der aktuelle Zustand gespeichert. Dieser Zustandsübergang wird mittels Modellierung als endlichen Automaten erreicht[CFFT06]. Bei diesem Ansatz ist der Webservice nicht mehr Zustandslos, da er vorherige Anfragen speichert. Dies erlaubt keine sinnvolle Anbindung in einen Workflow und ist daher nicht geeignet.

2.3.2. Generic web service wrapper for efficient embedding of legacy codes in service-based workflows

In der Arbeit von Glatard et al. wird ein generischer Wrapper präsentiert der zur Laufzeit eine Anwendung adaptieren und bereitstellen kann. Dies wird über eine Legacy Code Descriptor Datei im XML Format realisiert, welche die zu adaptierende Anwendung beschreibt. Zur Laufzeit wird dann die entsprechende WSDL Datei erzeugt [GEM$^+$06]. Mit diesem Ansatz ist es sehr leicht möglich Programme bereitzustellen die mit Kommandozeilen Befehlen funktionieren.

2.3.3. Generisches Web Service Interface um Simulationsanwendungen in BPEL-Prozesse einzubinden

Die Diplomarbeit von Jens Rutschmann verfolgt einen generischen Interface Ansatz um Simulationsanwendungen in BEPL-Prozesse einzubinden. In seiner Arbeit erstellte er ein generisches Web Service Interface das für alle Simulationsanwendungen geeignet sein soll. Dafür wurden exemplarisch zwei verschiedene Simulationsanwendungen analysiert. Danach wurden Gemeinsamkeiten der Simulationen extrahiert um eine einheitliche Interface Beschreibung zu erhalten. Die unterschiedlichen Simulations-Merkmale wurden dabei in ein Plugin-System ausgelagert. Ein Basis Webservice verwaltet die verfügbaren Plugins, welche Simulationsanwendungen anbinden. Die Simulationen müssen dazu die Plugin Schnittstelle implementieren und sich beim Basis Webservice anmelden. Um eine Simulationsanwendung zu verwenden wird der Basis Webservice angesprochen, dieser leitet die Anfragen weiter [Rut09].

2.3.4. Ausführung von Festkörpersimulationen auf Basis der Workflow Technologie

In der Diplomarbeit von Sven Hotta wurde eine Festkörpersimulationsanwendung mittels einen Workflow modelliert. Dazu musste zuerst die Legacy Simulationsanwendung Opal (Ostwald - Ripening of Precipitates on an Atomic Lattice) in einen Webservice gekapselt werden. Hierbei wurden verschiedene Ansätze diskutiert und getestet. Der Webservice wurde danach mit der Business Process Execution Language (BPEL) in einen Workflow orchestriert. Aufgrund von Problemen beim parallelen Zugriff auf Methoden der Simulationsanwendung mittels Java Nativ Access konnte der Ansatz nicht zielführend umgesetzt werden. Daher wurde auf die Verwendung von Java Runtime Klassen zurückgegriffen, da dies die einzige Möglichkeit darstellt eine parallele Ausführung der Simulation zu gewährleisten [Hot10].

Alle existierenden Arbeiten befassen sich in erster Linie damit, wie eine Legacy Anwendung an neue Systeme angebunden werden, allerdings nicht unbedingt in Form von Webservices. Hierbei steht oftmals nur die Kapselung im Mittelpunkt, jedoch nicht die Anwendung in einen Workflow zu integrieren. Bei der Mehrheit der Anwendungen handelt es ich um Business Anwendungen und nicht um Simulationsanwendungen. Dadurch lässt sich Vorgehensweise von Canfora et al. und anderen Arbeiten nicht für diese Arbeit anwenden. Der generische Web Service Wrapper von Glatard et al. betrachte nur Kommandozeilen Anwendungen erlaubt dafür aber ein effizientes bereitstellen dieser als Webservices. Da Simulationsanwendungen andere Eigenschaften wie Business Anwendungen aufweisen, wie zum Beispiel eine erheblich längere Laufzeit, müssen zusätzliche Aspekte betrachtet werden. Hier stellen die beiden Diplomarbeiten eine gute Basis bereit auf der aufgebaut werden kann. Die Erfahrungen und Ergebnisse fließen hier direkt in die Entwicklung des neuen Webservice ein. Die Arbeit knüpft an die Diplomarbeiten an und erweitert das bestehende System. Der entstehende Webservice wird später in einer Orchestration mit den Opal Webservices zusammenarbeiten. In diesem Fall erhält der neue Webservice die berechneten Daten der Opal Simulation als Eingabeparameter, welche in der ITAP Molecular Dynamics Simulation (IMD) weiterverarbeitet werden.

3. Simulationsanwendung

Es existieren bereits viele Simulationsanwendungen welche auf die Bedürfnisse von Wissenschaftlern zugeschnitten sind. Durch das vorangeschrittene Alter der Anwendungen weisen diese einen hohen Funktionsumfang auf und sind entsprechend gut getestet. Diese Anwendungen sind deshalb wertvoll und es besteht hohes Interesse diese auch zukünftig verwenden zu können. Solche Legacy Anwendungen verwenden aber oftmals überholte Technologien und lassen sind durch Inkompatibilität nicht einfach in ein neues System integrieren. So auch die Anwendung IMD (ITAP Molecular Dynamics) des Institut für Theoretische und Angewandte Physik[1] (ITAP). In dieser Arbeit wird IMD in der Version vom 25.02.2009 verwendet, jedoch ist gegen Ende der Arbeit eine aktuellere Version erschienen.

Das IMD stellt eine große Softwarebibliothek für klassische Molekulardynamik Simulationen bereit. Sie erlaubt den Versuchsaufbau der Simulation über Parameter und Konfigurationsdateien zu konfigurieren. Des weiteren werden Hilfsmittel (Utilitys) für die Nachbereitung der Simulationen zur Verfügung gestellt. Die Bibliothek wird von unterschiedlichen Instituten weltweit benutzt[al.09].

Ein möglicher Simulationsaufbau mittels IMD wäre zum Beispiel zu untersuchen wie sich eine Eisen-Kupfer-Legierung unter thermischer Alterung verhält. Die damit verbundenen Veränderungen in der Materialeigenschaft ist unter anderem für die Sicherheit von relevanter Bedeutung. Eine solche Legierung kommt beispielsweise in Stahl-Rohrleitungen und Stahl-Druckbehälter in Kraftwerken zum Einsatz[BS03].

3.1. Eigenschaften

Die IMD Bibliothek ist in ANSI-C programmiert und erfordert das Build-Management-Tool GNU make. Diese lässt sich daher einfach auf Unix-artigen Betriebssystemen verwenden. Eine Makefile für das Betriebsystem Windows existiert derzeit nicht. Die Bibliothek wurde flexibel mit einem Modularen Aufbau gestaltet um performant und lauffähig auf unterschiedlichen Rechnerarchitekturen zu sein. Es werden sowohl Arbeitsrechner, als auch Mehrkern Hochleistungsrechner, sowie Parallelisierung unterstützt. Zur Parallelisierung wird der Message Passing Interface (MPI) Standard verwendet.

Eine grobe Übersicht der einzelnen Komponenten ist in Abbildung 3.1 zu sehen. Die IMD Bibliothek bietet einen enormer Umfang an Algorithmen und Hilfsmittel zur Simulationsdurchführung an. Bestandteil der Bibliothek sind nicht nur Algorithmen zur Berechnung, sondern auch Werkzeuge zur Umrechnungen in andere Formate.

[1] ITAP, http://www.itap.physik.uni-stuttgart.de/

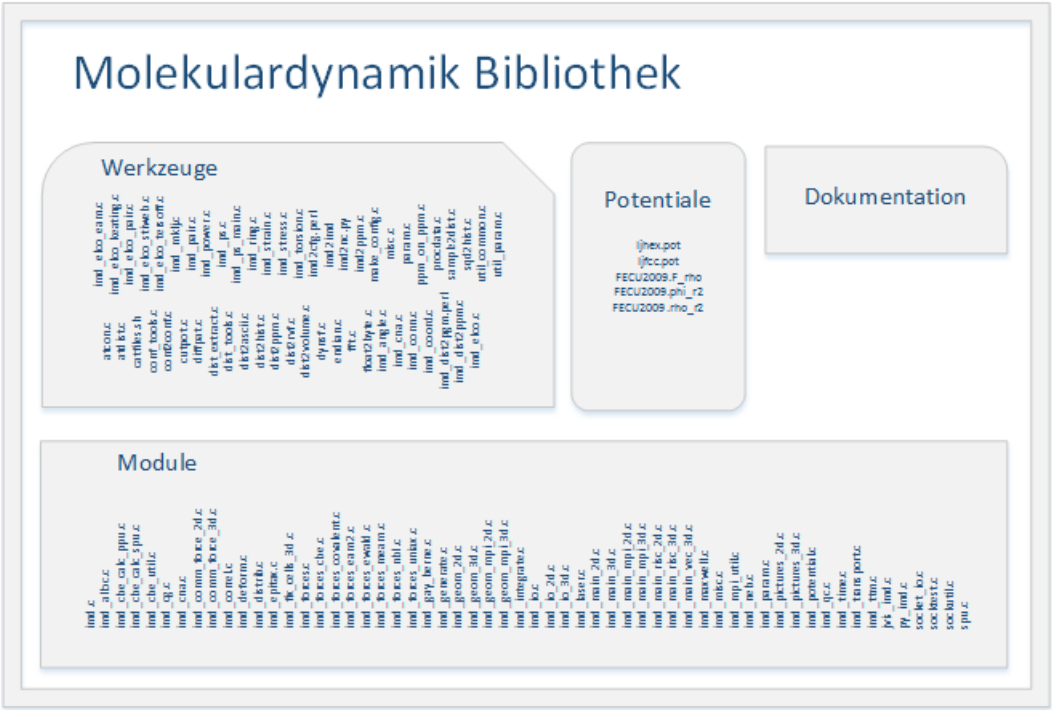


Abbildung 3.1.: Übersicht der Molekulardynamik Bibliothek

Um das kompilieren zu vereinfachen gibt es eine `IMDSYS` Variable, welche dem Programm make als Parameter übergeben werden muss. Diese Variable hält Voreinstellungen für verschiedene Rechnerarchitekturen in Zusammenstellung mit einem Compiler bereit. Soll ein Modul auf einer Maschine mit einem x86-Prozessor und dem Compiler gcc3 kompiliert werden muss dies in der Variable eingetragen sein. In diesem Fall müsste IMDSYS mit x86_64-gcc3 belegt werden. Die Werte haben immer die gleiche Form. Zuerst steht der Architekturtyp danach folgt ein Bindestrich und schließlich die Compilerversion. Stimmt der Parameter nicht mit der tatsächlichen Rechnerarchitektur überein lässt sich das Modul nicht kompilieren. Auf einer lokalen Umgebung empfiehlt es sich diese Variable als Umgebungsvariable zu setzen. Unter Umständen kann es erforderlich sein weitere Flags anzugeben. Dies muss über den herkömmlichen Weg, als Parameter für make, geschehen. Im Falle das keine passende Voreinstellung angeboten wird, müssen die entsprechenden Einstellungen selbst im Makefile vorgenommen werden. Es wird empfohlen gcc in Version 3 oder icc zu verwenden. Der Grund dafür ist, dass andere Compiler ein Programm bauen das langsam zur Laufzeit ist[al.09]. Im Programmcode von IMD finden sich Hinweise das eine Kommunikation über Sockets zu anderen Anwendungen realisiert werden kann. Das Handbuch umfasst dazu nur spärliche Notizen ohne brauchbare Details. Informationen wie diese Socketverbindung verwendet werden kann findet sich nicht. Zusätzlich weißt der Programmcode von IMD daraufhin, dass es möglich ist Python Module zu kompilieren. Möglicherweise sind diese äquivalent zu den C-Programmen. Leider findet sich im Handbuch weder ein Hinweis das diese Möglichkeit existiert noch wie diese zu verwenden ist.

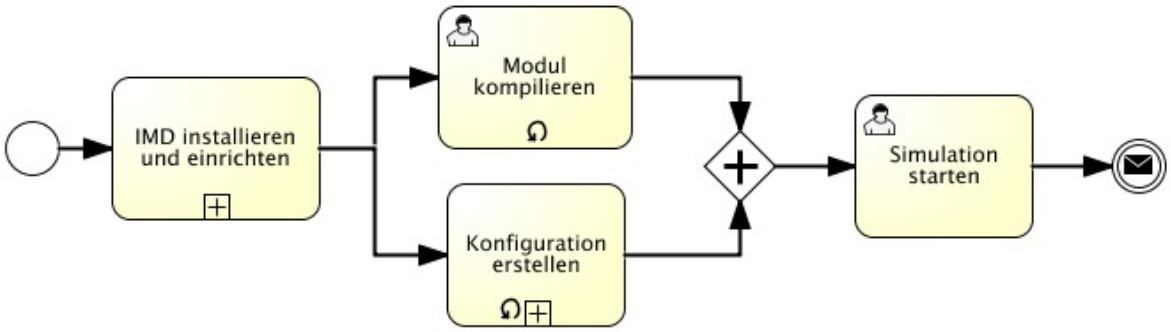


Abbildung 3.2.: Ablauf einer manuellen Simulation mit IMD

3.2. Verwendung

Im folgenden wird beschrieben wie eine Simulation mit IMD umgesetzt werden kann. Dabei wird im Verlauf der Simulation ein Zugversuch durchgeführt. Mit einem Zugversuch kann die Streckgrenze von Werkstoffen experimentell bestimmt werden. Die Simulation im folgenden stellt unter Umständen kein sinnvolles Experiment dar und dient nur der Veranschaulichung der Verwendung von IMD.

Um eine Simulation durchzuführen benötigt es mehrere Schritte die von Hand ausgeführt werden müssen (Abbildung 3.2). Zunächst muss die Bibliothek installiert und eingerichtet werden. Als erstes sollte der Verzeichnispfad in den die kompilierten Anwendungen abgelegt werden, in den Pfad der ausführbaren Dateien des Betriebssystems eintragen werden. Das erleichtert das aufrufen der Anwendungen, da nicht der gesamte Dateipfad angegeben werden muss. Die kompilierten Anwendungen werden standardmäßig unter `$HOME/bin/$HOSTTYPE` abgelegt. Das kann in der Makefile geändert werden.

Zunächst müssen für den Zugversuch passende Anwendungen kompiliert werden. Aus Modulen der IMD Bibliothek werden diese zusammengebaut. Die Befehle zum kompilieren werden im Quellcodeverzeichnis `/imd/src/` ausgeführt. Das Experiment besteht aus zwei Phasen, weshalb zwei Anwendungen benötigt werden. Für jede Anwendung wird ein make Prozess ausgeführt das als Parameter die IMDSYS Variable erhält. Die Funktionalität der Anwendung wird über einen weiteren Parameter bestimmt. Dieser Parameter besteht aus einer Zeichenkette von Schlüsselwörtern. Jedes Schlüsselwort repräsentiert ein Modul oder Option aus der IMD Bibliothek. Die Reihenfolge der Schlüsselwörter ist dabei unerheblich. Alle vorhandenen Schlüsselwörter für die Kompilieroptionen können aus dem Handbuch entnommen werden. Beide Anwendungen verwenden die Embedded Atom Method (eam) und die Optionen für Deformation (homdef) und Druck (stres). Zusätzlich enthält die erste Anwendung den Relaxator glok. Hierfür wird der Befehl 3.1 ausgeführt. Nach jedem make Befehl ist es erforderlich die erstellen temporären Dateien zu entfernen. Dies passiert über den Befehl 3.3 der das Verzeichnis von temporären Dateien säubert. Die zweite Anwendung enthält die gleichen Optionen und besteht aus dem Ensemble npt_axial, welche konstanten Druck und konstante Temperatur verwendet. Das wird mit Befehl 3.2 erreicht. Das Kompilieren dauert nur wenige Sekunden.

Listing 3.1 Befehl um ersten Simulationsschritt zu kompilieren

```
make IMDSYS=x86_64-gcc3 imd_eam_glok_homdef_stress
```

Listing 3.2 Befehl um zweiten Simulationsschritt zu kompilieren

```
make IMDSYS=x86_64-gcc3 imd_eam_npt_axial_homdef_stress
```

Als nächstes wird jeweils eine Konfigurationsdatei pro Phase angelegt. Diese entstehen von Hand und beschreiben was in der Simulation passiert. In jeder Zeile steht ein Parameter gefolgt von einem Wert oder mehrere Werte welche mit einem Komma separiert sind. Leere Zeilen oder Kommentare hinter einer Raute werden ignoriert. Welche Parameter optional sind und welche nicht hängt von der Anwendung ab. Jedoch muss jeder Parameterdatei das Simulationsensemble (zum Beispiel glok), die Startkonfiguration (coordname) und die Anzahl der durchzuführenden Schritte angeben. Vor dem Start der Simulation wird auf das vorhanden sein aller erforderlichen Parameter geprüft [al.09]. Die Konfiguration parameter_glok ist für den ersten Simulationsschritt (3.1) und parameter_npt für den zweiten Simulationsschritt (3.2). Die Einstellungen können den Listing 3.4 sowie Listing 3.5 entnommen werden.

Beide Konfigurationsdateien enthalten nur notwendige Parameter. Dazu zählt die Pfadangabe zu den Potentialdateien und der Startkonfiguration. Weiter wird der Name der Ausgabedateien, die Anzahl der zu berechnenden Checkpoints, sowie die Randbedingungen deklariert. Eine vollständige Angabe aller möglichen Parameter und deren Bedeutung ist dem Benutzerhandbuch [al.09] zu entnehmen. Die Pfade wo die entsprechenden Dateien hinterlegt sind sollten am besten mit absoluten Pfadangaben angegeben werden. Am einfachsten ist es alle Dateien in ein Verzeichnis zu kopieren, da dann auch mit relativen Pfaden gearbeitet werden kann. In diesem Beispiel befinden sich die Startkonfiguration und Potentialdateien im selben Verzeichnis wie die Anwendung, daher reicht es aus nur den Dateinamen anzugeben. Sind diese Vorbereitungen getroffen lässt sich anschließend der ersten Simulationsschritt mit dem Befehl 3.6 anstoßen. Es wird die Anwendung mit der Konfigurationsdatei als Parameter gestartet.

Sobald die Simulation gestartet ist erscheint iterativ eine Ausgabe. Die Ausgabe liefert Informationen über den Fortschritt der Simulation. Während die Simulation läuft werden in verschiedenen Stadien der Berechnung Checkpoint Dateien in das Dateisystem herausgeschrieben. Diese Checkpoint Dateien enthalten die Berechnungen welche bis dato durchgeführt worden sind. Außerdem repräsentieren sie eine Startkonfiguration mit der die Simulation an dieser Stelle aufgenommen und fortgesetzt werden kann. Die einzelnen Stadien der Simulation lassen sich so nachverfolgen. Die Ausgabedateien werden automatisch aufsteigend nummeriert. Als nächstes können die berechneten Checkpoint Dateien mit dem zweiten Simulationsschritt weiterverarbeitet werden. Der Befehl 3.7 stößt den zweiten Schritt an. Für jede dieser Checkpoint Dateien muss zuvor eine neue Konfigurationsdatei angelegt werden. In der

Listing 3.3 Befehl um Verzeichnis von temporären Dateien zu säubern

```
make clean
```

Listing 3.4 Konfigurationsparameter für ersten Simulationsschritt

```
coordname testdatei.dat  # Startkonfiguration für die zugrundeliegende Berechnung
outfiles glok # Dateiname für Ausgabedatei
core_potential_file FECU2009.phi_r2 #
embedding_energy_file FECU2009.F_rho # Potential Dateien
atomic_e-density_file FECU2009.rho_r2 #
ensemble glok # Aktiviertes Ensemble
maxsteps 1000 # Anzahl Schritte
timestep 0.2
checkpt_int 1000
eng_int 10
starttemp 0.05
relax_rate 0.1
relax_mode axial
ntypes 3
box_from_header 1
```

Listing 3.5 Konfigurationsparameter für zweiten Simulationsschritt

```
coordname glok.00001.chkpt # Startkonfiguration für die zugrundeliegende Berechnung
outfiles nptdef # Dateiname für Ausgabedatei
core_potential_file FECU2009.phi_r2 #
embedding_energy_file FECU2009.F_rho # Potential Dateien
atomic_e-density_file FECU2009.rho_r2 #
ensemble npt_axial # Aktiviertes Ensemble
maxsteps 140000 # Anzahl Schritte
timestep 0.2
checkpt_int 14000
eng_int 5
starttemp 0.0258520
endtemp 0.0258520
pressure_start 0
pressure_end 0
relax_dirs 1 1 0
lindef_interval 1
lindef_x 0 0 0
lindef_y 0 0 0
lindef_z 0 0 1e-6
inv_tau_eta 0.66
inv_tau_xi 0.15
ntypes 3
box_from_header 1
```

Listing 3.6 Befehl um ersten Simulationsschritt auszuführen

```
imd_eam_glok_homdef_stress -p parameter_glok
```

Listing 3.7 Befehl um zweiten Simulationsschritt auszuführen

```
imd_eam_npt_axial_homdef_stress -p parameter_npt
```

neuen Konfiguration muss mindestens der `coordname` Parameter erneuert werden. Der `coordname` verweist dabei auf die Checkpointdatei, welche als Startkonfiguration verwendet wird.

Der zweite Simulationsschritt erzeugt auch wieder Checkpoint Dateien. Im Verlauf der Simulation entstehen so viele Dateien die verarbeitet und analysiert werden können. Die Analyse kann dabei mit zu Hilfenahme des Kommandozeilen Programms `gnuplot` durchgeführt werden, welches Diagramme aus den Dateien erstellt. Um nicht unbeabsichtigt Dateien zu überschreiben empfiehlt es sich für jede Simulation ein separates Verzeichnis anzulegen.

3.3. Analyse

Um eine Simulation zu beschreiben und durchzuführen benötigt es viele Dateien welche verwaltet werden müssen. Im laufe der Simulation entstehen immer weitere Dateien was ein hohes Maß an Fleißarbeit benötigt diese zu organisieren. Es ist wünschenswert, dass sich Wissenschaftler voll und ganz der Durchführung ihrer Simulation widmen können, ohne diesen Overhead an Verwaltung bearbeiten zu müssen. Oftmals werden ähnliche Simulationen mit feinen Parameter Änderungen durchgeführt um die Auswirkungen zu analysieren. Diese Parameterstudien erfordern beim manuellen Ausführen der Simulation einen noch höheren Verwaltungsaufwand. Daraus resultiert das Ziel den Wissenschaftler eine automatisierte Lösung anbieten zu können, welche ihnen einerseits die technische Verwaltung aber auch das zeitintensive manuelle Ausführen der Simulation abnimmt. Eine Automatisierung führt zu einer Entlastung der Wissenschaftler, welche sich dadurch auf Ihre Kernkompetenzen konzentrieren können.

Aufgrund des enormen Umfangs der Bibliothek wird auf das bereitstellen der Werkzeuge vorerst verzichtet und nur die rechenintensiven Hauptkomponenten als Webservice angeboten. Zu den Hauptkomponenten zählt alles was benötigt wird um eine Simulation zusammenstellen und auszuführen. Weiter ist zu beachten das die Bibliothek nur Plattformabhängige Anwendungen kompilieren und ausführen können. Da eine Anwendung für unterschiedliche Prozessorarchitekturen gebaut werden muss.

4. Umsetzung

Im folgenden Kapitel werden die Anforderungen für den Webservice spezifiziert. Es werden unterschiedliche Varianten diskutiert und anschließend eine geeignete Architektur entworfen. Die gewählte Architektur wird mit einer Programmiersprache implementiert und später die Verwendung des Webservice demonstriert.

4.1. Anforderungen

Es haben zwei Institute Interesse an dieser Arbeit, welche jeweils spezifische Anforderungen an das Resultat stellen. Die Anforderungen wurden an Hand von Gesprächen mit den beteiligten Personen erfasst. Das IMWF erhofft sich nicht nur, dass eine vereinfachte Arbeitsweise mit der Bibliothek ermöglicht werden kann. Sondern auch das durch die folgende Automatisierung in kommenden Arbeiten eine Menge Zeit eingespart wird. Gerade durch die optionale Parallelisierung mittels MPI auf verteilten Systemen verspricht man sich eine verkürzte Rechendauer einer Simulation. Das IAAS hingegen möchte die Entwicklung ihres Scientific WfMs vorantreiben. Das Ziel ist es zu zeigen, dass mit Business Workflows auch erfolgreich Scientific Workflows umgesetzt werden können. Deshalb soll die Bibliothek in einzelne Teilaspekte zerlegt werden um später einen größtmöglichen Workflow zu orchestrieren der die Komplexität der Simulationen widerspiegelt. Für die Anbindung an einen Workflow der mit BPEL orchestriert wird, muss die Bibliothek in einen Webservice gekapselt werden. Aufgrund dessen muss der entstehende Webservice ein hohes Maß an Flexibilität erfüllen.

4.2. Entwurf

Der Webservice wird mittels WSDL beschrieben. Anschließend erfolgt die serverseitige Implementierung. Um den Webservice zu erstellen wurde die Entwicklung mit dem Top-Down Entwurf durchgeführt. Auf Basis der Analyse von Funktionsweise und gegebenen Anforderungen entsteht nun der Entwurf der Services. Als erstes werden die erforderlichen Arbeitsschritte (Abbildung 3.2) in eigenständige Teilprozesse zerlegt. Diese Teilprozesse werden dann als Service umgesetzt. Es lassen sich eine Vorbereitungsphase und eine Ausführungsphase erkennen. Eine Nachbereitungsphase existiert derzeit mit IMD nicht. Zur Vorbereitungsphase gehören die Prozesse des Kompilierens und das Parameterdateien entwerfen. Die Ausführungsphase umfasst das starten der Anwendung. Aus den Teilprozessen lässt sich jeweils ein Service erstellen. Es entstehen so vier Services, die zusammen mit der IMD Bibliothek arbeiten. Ein Kompilier-Service, welcher das Zusammenbauen der Anwendungen übernimmt. Ein Parameter-Service, welcher die Parameterdateien verwaltet und erstellt. Ein Potential-Service, der existierende Potentialdateien bereithält und ein Simulation-Service, welcher die

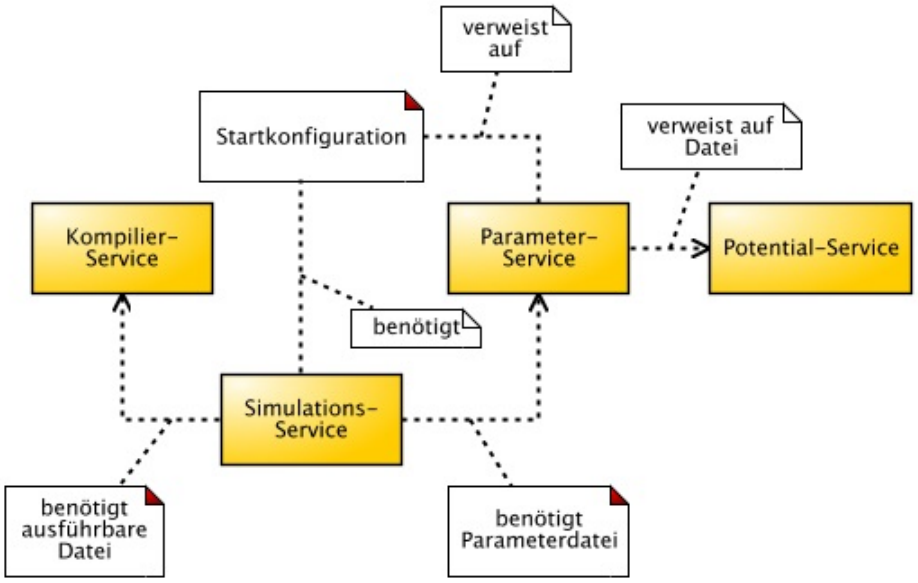


Abbildung 4.1.: Beziehungen zwischen den Services

Simulation durchführt. Alle Services bis auf der Simulations-Service sind eigenständig verwendbar. Die Beziehungen der einzelnen Services sind in Abbildung 4.1 dargestellt.

Eine besondere Abhängigkeiten besteht zwischen den Kompilier-Services und dem Simulations-Service. Dadurch das ein ausführbare IMD Anwendung für eine bestimmte Architektur übersetzt wird, müssen Kompilier und Simualations Service eine Prozessorarchitektur teilen. Eine unterschiedliche Prozessorarchitektur führt zu Inkompatibilität, so dass der Simulations-Service die Anwendung nicht ausführen kann und diese mit einem Fehler beenden muss. Alle anderen Dateien sind reine Textdateien und somit Plattformunabhängig. Es muss überlegt werden wie sichergestellt werden kann, dass jeweils ein Kompilier und Simulations Service Paar kompatibel zueinander sind.

Die einfachste Lösung wäre alle Services innerhalb eines großen Webservice abzuarbeiten und nur eine Schnittstelle nach außen anzubieten. Diese Fassade würde intern die einzelnen Service anstoßen und hat damit eine globale Sicht auf die Simulation. Das würde alle Abhängigkeiten auflösen, da alles auf einem Server laufen kann. Dieser Ansatz ist aber sehr unflexibel. Es können die Services die wenig Rechenleistung benötigen nicht auf andere Server platziert werden und verschwenden so Ressourcen auf teuren Supercomputern. Mehrere Instanzen dieses Webservice auf verschiedenen Servern zu verteilen widerspricht dem Ansatz alles in einem zu kapseln. Da der Vorteil der Unabhängigkeit verloren geht und die Webservice möglicherweise nicht kompatibel zueinander sind. Weiter widerspricht dies dem SOA Konzept und dem der Wiederverwendbarkeit. Somit ist dieser Ansatz keine zufriedenstellende Lösung.

Auch die Idee Potential, Parameter, sowie Kompilier und Simulations Service in drei Webservice Gruppen mit jeweils einer Schnittstelle aufzuteilen scheint nicht zufriedenstellend zu sein. Die Abhängigkeit zur Prozessorarchitektur wären aufgelöst. Jedoch wäre der Kompilier und Simulations

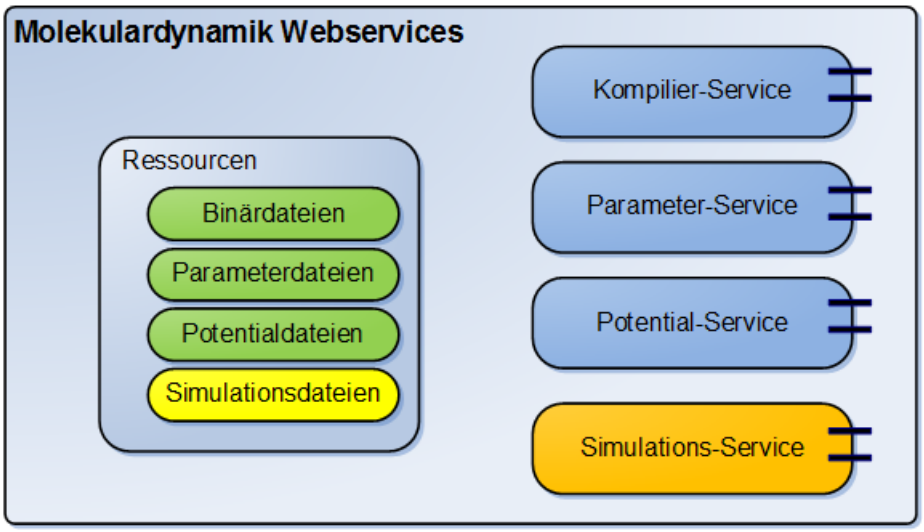


Abbildung 4.2.: Molekulardynamik Webservice Architektur mit Schnittstellen und Ressourcen

Service fest gekoppelt. Zwar lassen sich so die rechenintensiven Services ohne Overhead auf andere Lokationen platzieren trotzdem lassen sich die Services nicht einzeln nutzen.

Deshalb wurde ein Kompromiss eingegangen. Es werden alle Service als Webservice bereitgestellt (Abbildung 4.2). So erhält man die größtmögliche Flexibilität jedoch wird in kauf genommen das eine inkompatible Datei kommuniziert werden könnte. Dies bringt aber keine schwerwiegenden Konsequenzen mit sich, da dies vor dem starten der Simulation erkannt und mit einer Fehlermeldung quittiert wird. Das Wissen welche Webservices untereinander kompatibel sind kann die Service Registry übernehmen. Im WfMs wäre dies die Aufgabe der GUI oder der Verwaltungseinheit. Diese sollten nur kompatible Webservices zum orchestrieren anbieten. Die Ressourcen werden von den Webservices lokal vorgehalten. Es wird damit explizit auf einen zentralen Ressourcen Manager wie in [Hot10] verzichtet. Dies hat den Vorteil das kein Single Point of Failure (SPoF) entsteht. Sollte der Ressourcen Manager nicht verfügbar sein wären alle Webservices nicht mehr benutzbar welche diesen verwenden. Durch die lokale Ablage der Ressourcen sind diese nicht an einen Ressourcen Manager gebunden. Zwar sind dann nicht überall die selben Ressourcen verfügbar, dass ist aber auch nicht notwendig da die Simulationsanwendungen Plattformabhängig sind. Für die Integration in ein Workflow könnten die Ressourcen zusätzlich im Ressourcen Manager abgelegt werden. Dieser müsste dazu nur die Dateien von den Webservices anfordern. Durch die Aufteilung in einzelne Services können diese auf unterschiedlichen Servern verteilt werden. Die Services der Vorbereitungsphase benötigen relativ wenig Rechenleistung und können auf rechenschwachen Servern deployed werden. Wohingegen der Simulationsservice viel Rechenleistung und im besten Fall sogar MPI zur Verfügung stellt, auf einen Hochleistungsrechner platziert werden sollte.

Im folgenden ist eine genauere Beschreibung der einzelnen Service, welche Aufgaben diese übernehmen und welche Operationen angeboten werden.

Kompilier-Service Dieser Service baut eine konkrete Anwendung aus der Bibliothek zusammen. Die Anwendung enthält die Algorithmen Bibliothek zur Berechnung einer Simulation. Der Service benötigt Benutzerrechte für das Kompilieren der Anwendungen. Es müssen Anfragenachrichten mit einer Vorschrift wie kompiliert werden soll entgegengenommen und ausgeführt werden. Als Antwortnachricht erhält der Anfragesteller die Binärdaten oder den Pfad zu dieser ausgeliefert.

Operationen:
`compileBinary`, `listBinarys`, `getBinaryFileByName`, `getBinaryPathByName`, `deleteBinaryByName`
Formale Beschreibung: Anhang A.2.4

Potential-Service In diesem Service werden die Potentialdateien, welche als Parameter für die Anwendungen notwendig sind, verwaltet. Es können alle vorhandenen Potentiale aufgelistet werden. Einzelne Potentialdateien erhält man in Form von Binärdaten oder als Pfad zurück.

Operationen:
`writePotential`, `getPotentialList`, `getPotentialFileByName`, `getPotentialPathByName`, `deletePotential`
Formale Beschreibung: Anhang A.2.2

Parameter-Service Der Service erlaubt es Konfigurationen für die Simulation in Form der Parameterdatei zu erstellen. Es werden Tupel von Konfigurationelement und Wert entgegengenommen und als Parameterdatei geschrieben. Parameterdateien werden als Binärdaten oder als Pfad zurückgegeben.

Operationen:
`writeParameterFile`, `getParameterList`, `getParameterFileByName`, `getParameterPathByName`, `deleteParameterFile`
Formale Beschreibung: Anhang A.2.3

Simulations-Service Dieser Service führt die Simulation aus. Es werden Benutzerrechte zum ausführen benötigt. Als Eingabe wird die Simulationsanwendung, die Eingabedatei, Potentialdateien und die Parameterdatei benötigt. Es erfolgt eine Antwort mit der Simulationsnummer. Während der Simulation werden die Checkpoint-Dateien als Callback an den Anfragesteller gesendet.

Operationen:
`execute`, `abort`, `deleteSimulation`, `listFilesBySimulationId`, `getFileByNameAndSimulationId`
Formale Beschreibung: Anhang A.2.5

Es gilt noch die Frage zu klären, wie letztendlich die Bibliothek im Kompilier-Service als Webservice zur Verfügung gestellt werden kann. Um eine Kapselung durchzuführen gibt es unterschiedliche Methoden und Vorgehensweisen. Mögliche Varianten wären das verwenden von Sockets, einer neuen API, der direkter Dateiaustausch oder das Aufrufen nativer Methoden [Sne00]. Ausgehend von den Anforderungen bieten sich mehrere Varianten an die Bibliothek als Webservice bereitzustellen. Da bereits zwei Diplomarbeiten (Abschnitt 2.3) eine sehr ähnliche Anforderung gelöst haben bot es sich an die dort gewählten Varianten zuerst zu betrachten.

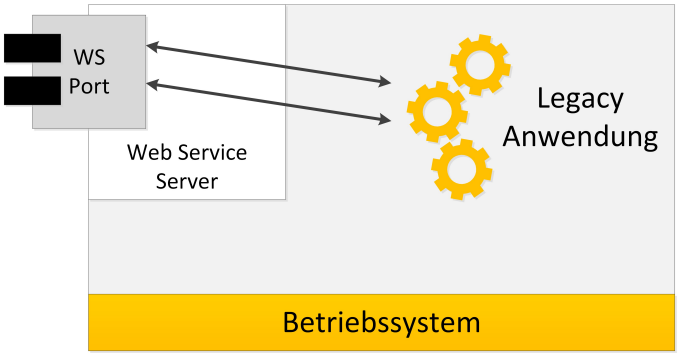


Abbildung 4.3.: Lösung mit integrierten Webservice nach [Rut09]

Die erste Möglichkeit ist es die Bibliothek selbst mit einem Webservice auszustatten. Damit erhält die Anwendungen die Anfragen direkt, ohne eine vermittelnde Komponente. In der Abbildung 4.3 ist dies dargestellt.

In beiden Arbeiten wurde das gSOAP Toolkit in Betracht gezogen um einen Webservice in die Anwendungen einzubauen. gSOAP verwendet die Programmiersprache C genau wie die IMD Bibliothek und unterstützt damit eine Implementierung in der selben Programmiersprache. Hier wäre zusätzlich der Vorteil das kein Wrapper benötigt wird um die Anwendungen auszuführen. Allerdings benötigt dies eine direkte Modifikation an IMD. Wobei ein weiteres Modul für IMD erstellt werden müsste. Dies wurde nicht getestet, da keine fortgeschrittenen C Kenntnisse vorhanden sind. Es ist jedoch auf jeden Fall ein Wrapper für das Kompilieren erforderlich um alle IMD Module zusammenstellen zu können, da die gesamte IMD Bibliothek bereitgestellt werden soll und nicht nur einzelne Anwendungen. Hier ist fraglich wie letztendlich die kompilierten IMD Anwendungen mit integrierten Webservice deployed werden. Der kompilierende Service müsste dies mitübernehmen. Ein Top-Down Entwurf lässt sich wegen der unterschiedlichen Module schwer realisieren. Die Operationen in der WSDL müssten dynamisch erzeugt werden. Jedoch ist auch ein Bottom-Up Entwurf aufgrund der Art und Weiße wie IMD kompiliert wird schwer umzusetzen. Eine Umsetzung mit gSOAP verspricht viele Fallstricke weswegen von diesem Ansatz abgesehen wurde. Um diese Komplikationen zu vermeiden bietet es sich an den folgenden Ansatz zu verwenden der einen Adapter benutzt.

Bei einem Adapter werden die Anfragen von einer anderen Anwendung entgegengenommen. Das erlaubt die neue Funktionalität getrennt von der alten Anwendung zu implementieren. Der Adapter konvertiert die Anfragen und vermittelt diese weiter (Abbildung 4.4). Eine Modifikation der alten Anwendung ist nicht notwendig. Das hat außerdem den Vorteil, dass die Schnittstelle bei einer Weiterentwicklung von IMD nicht angepasst werden muss.

Eine bestehende Adapter-Anwendung wird von Jens Rutschmann angeboten. Hier soll ein Plugin für die Anbindung an den Basis Webservice reichen [Rut09]. Beim genaueren betrachten stellt sich das Plugin allerdings als vollwertige Service Beschreibung in WSDL heraus, welche an den Basis Webservice gebunden wird. Der Basis Webservice fungiert dabei als Proxy, das die einzelnen Plugins anspricht. Somit bringt die Verwendung des generischen Interfaces immer einen Overhead mit sich. Die Realisierung mittels adaptieren und binden zur Laufzeit, wie es in [GEM+06] gemacht wird

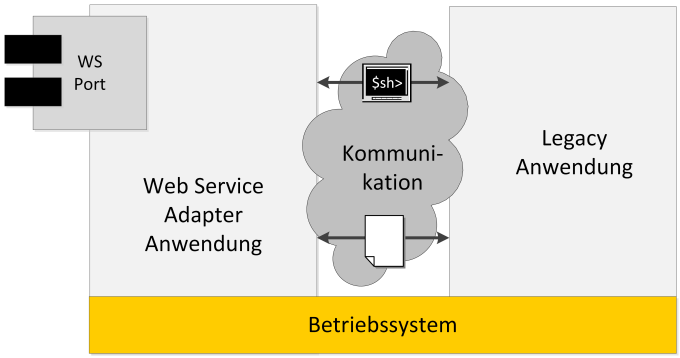


Abbildung 4.4.: Lösung mit Webservice Adapter nach [Rut09]

ist flexibler. Dort benötigt es keinen Proxy der zwischen Anfrage und Service geschaltet wird. Der Aufwand eine vollständige WSDL Beschreibung für jede Anwendung zu schreiben entfällt dort auch. Um den Basis Webservice trotzdem zu verwenden ist es erforderlich, dass entsprechende Plugin-Callback Interface zusätzlich vom Simulations Webservice implementieren zu lassen. Dies kann auch nachträglich ausgeführt werden. Der Vorteil ist dabei nicht klar ersichtlich. Aufgrund dessen und der damit ansteigenden Komplexität wurde entgegen das schreiben eines Plugins entschieden. Stattdessen entsteht ein neuer eigenständiger Webservice Adapter der die Funktionalität der Bibliothek umhüllt.

4.3. Implementierung

Die vorherigen Arbeiten haben für die Implementierung der Webservices den Webserver Apache Tomcat verwendet. Deshalb wird auch für diese Arbeit Tomcat verwendet, um eine einfache Integration in die Infrastruktur zu gewährleisten. Es kommt dabei Tomcat in der Version 7 zum Einsatz. Für die SOAP und WSDL Unterstützung in Tomcat wird Axis2 in der Version 1.5.6 verwendet, obwohl schon eine neuere Version angeboten wird. Die schon vorhandenen Webservices verwenden alle die alte Axis Version. Die neuere Version ist allerdings nicht abwärtskompatibel, deshalb wird die ältere Version verwenden. Die ältere Version wird wegen der Inkompatibilität immer noch mit Updates versorgt, so dass dies kein Problem darstellt. Die Webservices wurden mit der Programmiersprache Java implementiert.

Durch die Wahl des Top-Down Entwurfs wurden zuerst die WSDL Dateien für die Webservices erstellt. In einer XSD Datei wurden zuerst die Typen und Elemente definiert, welche von allen Webservice gemeinsam verwendet werden. Die XSD Definitionen können dann in die WSDL Dateien eingebunden und zur Erstellung der individuellen Beschreibung benutzt werden. Aus den fertigen WSDL Dateien wurde mit Hilfe des Axis2 Werkzeugs wsdl2java das Service-Grundgerüst erzeugt. Mit wsdl2java ist es möglich die Programmiersprachen unabhängige Beschreibung in eine konkrete Java Realisierung zu übersetzen. Bei diesem Schritt werden die abstrakten Definitionen in konkrete Java Klassen und Java Typen übersetzt und generiert. Anschließend kann die eigentliche Einbettung der Logik erfolgen.

Listing 4.1 Java Runtime Klasse zur Befehlsausführung verwenden

```
1 // Starte eine Shell (bash) und führe einen Befehl damit aus.
2 String[] cmd = new String[] {"/bin/bash", "-c", "echo Hallo Welt" };
3 // Für Windows "cmd.exe" und Schalter "/C" verwenden.
4 Runtime.getRuntime().exec(cmd);
```

Die Geschäftslogik ist dazu in einem eigenen Java Paket abgelegt worden, um automatisch generierten Code von Axis und selbst erstellte Logik zu trennen. Das hat den Vorteil das bei erneutem generieren des Service-Grundgerüstes der entwickelte Code nicht überschrieben wird.

Der Kompile-Service erhält eine Liste von Modulen welche von der Bibliothek zu einer Anwendung kompiliert werden soll. Die Liste wird zu einem Befehl für make transformiert. Mit Hilfe der Java Runtime Klasse wird dieser Befehl ausgeführt. Dies ist jedoch nicht ohne weiteres möglich da die Runtime Klasse keine echte Shell Umgebung mit Kommandozeilen Interpreter bereitstellt. Deswegen ist es erforderlich zuerst eine Shell zu starten. Mit der Runtime Klasse wird eine Bash (Bourne-again shell) gestartet (Listing 4.1). Mit dieser Shell kann der Befehl verarbeitet werden. Eine Bash ist hauptsächlich auf Unix-artigen Betriebssystemen verfügbar, weshalb die Betriebsystemunabhänigkeit auf der Serverseite verloren geht. Da allerdings das Makefile von IMD nur für Unix-artige Betriebssysteme vorhanden ist wurde sich entschieden dies dennoch so umzusetzen. Um die Betriebsystemunabhänigkeit nicht zu verlieren wäre es erforderlich ein weiteres Makefile bereitzustellen und das Richtige entsprechend der Plattform auszuwählen. Auf welcher Plattform der Webservice läuft könnte über ein Konfigurationsdatei geregelt werden. Anstelle der Bash müsste dann die Windows Kommandozeile gestartet werden.

Für die Verwaltung der Potential- und Parameterdateien genügt es diese im Dateisystem abzulegen. Dazu wurde innerhalb des Webservices Containers eine Verzeichnisstruktur angelegt. Im Gegensatz dazu wäre es auch möglich einen Pfad außerhalb der Webservice Umgebung zu bestimmen. Das hat aber den Nachteil das bei jedem Service deploy sichergestellt werden muss, dass dieser Pfad existiert und Lese- sowie Schreibrechte vorhanden sind. Dieser Konfigurationsaufwand wurde mit dem ablegen der Dateien innerhalb des Webservice Containers vermieden.

Webservice Ressourcen Verzeichnisse

/imd
: In diesem Verzeichnis wird die IMD Bibliothek abgelegt. Es können mehrere Versionen in Unterordner organisiert werden. Es empfiehlt sich jeweils das Realease Datum als Ordnernamen zu verwenden.

/binarys
: Dieses Verzeichnis ist auch für den Kompilier-Service und enthält die fertig übersetzten IMD Anwendungen.

/parameters
: Das Verzeichnis für den Parameter-Service. Es enthält alle bereits geschriebene Parameterdateien.

/potentials
Alle Potentialdateien des Potential-Services werden hier abgelegt.

/simulations
Jede Simulation erhält einen Unterordner in diesem Verzeichnis. Für die Simulation werden hier alle relevanten Dateien abgelegt. Zu den Dateien gehören die erzeugten Checkpoint, Iterations und Ergebnisdateien, sowie Logdateien der Fehler und Standardausgabe.

Der Simulations-Service verwendet die Java Runtime Klasse um die Simulationsanwendung auszuführen. Bevor die Anwendung gestartet wird werden die erforderlichen Dateien in einem neuen Unterordner abgelegt. Pro Simulationsinstanz entsteht so ein neuer Ordner. Dies ist notwendig, weil die berechneten Dateien im selben Ordner abgelegt werden in der die Simulationsanwendung gestartet wird. Andernfalls überschreiben sich die Dateien. Sobald während der Simulation eine neue Datei erstellt wird, ruft der Service ein Callback auf, dass den Klienten über die neue Datei benachrichtigt. Der Klient kann dann entscheiden ob diese Datei gleich in einer weiteren Simulationsinstanz weiterverarbeitet wird. Notwendigerweise muss dazu der Klient das Callback Interface implementiert haben um die asynchronen Rückmeldungen zu erhalten. Mit folgende Operationen kann dabei der Klient benachrichtigt werden.

Callback Interface Das Callback-Interface wird vom Klienten implementiert um eine asynchrone Rückmeldung vom Simulations-Service zu erhalten. Es können Meldungen über Statusänderungen der Simulation empfangen werden. Dazu zählen allgemeine Vorkommnisse wie auch das erstellen von neuen Checkpointdateien.

Operationen:
`checkpointCallback`, `outputlogCallback`, `errorlogCallback`, `newFileCallback`, `notifyCallback`
Formale Beschreibung: Anhang A.2.6

Um eine Rückmeldungen zu erhalten muss beim Simulationsstart die Rückmeldeadresse des Klienten als Parameter mit übergeben werden. Normalerweise bietet SOAP dafür ein entsprechenden ReplyTo Header an. Dieser Header wurde aber aufgrund von Kompatibilität mit den bereits existierenden Webservices nicht verwendet.

Um die Webservices möglichst einfach zu konfigurieren wurde eine Java-Properties-Datei angelegt. In dieser lassen sich die Standard Pfade zu den verwendeten Ordner festlegen. Des weiteren kann für den Kompilier-Service die IMDSYS Variable festgelegt und das darunterliegende Betriebssystem eingestellt werden. Für den Simulations-Service kann eingestellt werden ob MPI verfügbar ist und welches Betriebssystem verwendet wird. Entsprechend der Einstellungen wird die Java Runtime Klasse die Befehle ausführen.

5. Verwendung

Bevor die Webservices verwendet werden können müssen diese auf einem Server bereitgestellt werden. Dazu muss für Tomcat ein aar-Archiv deployed werden, dass den Webservice beinhaltet. Dieses Archiv entspricht einer normalen Archivdatei, wobei eine konkrete Ordnerstruktur und die Dateiendung eingehalten werden muss. Aus der Entwicklungsumgebung werden die gewünschten Webservices als jar-Archiv exportiert. Anschließend wird das jar-Archiv umbenannt so das es die Endung aar trägt. Dies funktioniert, weil ein jar-Archiv bereits die richtige Ordnerstruktur aufweist. Nachdem die Webservices erfolgreich auf den Server deployed worden sind können diese verwendet werden. Es werden nur die Service Operationen detailliert betrachtet, welche eine komplexe Datenstruktur erwarten oder speziellen Richtlinien folgen.

Die meisten Operationen erwarten keine oder nur primitive Parameter. Eine einfache Antwortnachricht Anfrage die eine Liste zurückliefert könnte wie folgt aussehen (Listing 5.1).

Um beispielsweise eine Parameterdatei zu erstellen muss dem Operation `writeParameterFile` vom Parameter-Service eine Liste von Tupeln übergeben werden. Ein Tupel ist einer Optionen und Wert aus dem IMD Handbuch zugeordnet. Eine SOAP Nachricht mit dem Inhalt von Listing 5.2 würde eine Parameterdatei mit dem Inhalt von Listing 3.4 erstellen.

Der Kompilier-Service erwartet eine Liste von Modulen, welche in eine Anwendung übersetzt werden soll. Die verfügbaren Kompilieroptionen können aus dem IMD Handbuch entnommen werden. Die einzelnen Optionen werden dann als Liste von Strings gesendet. Die Anfrage für den Kompilierbefehl aus Listing 3.1 kann mit folgender SOAP Nachricht realisiert werden (Listing 5.3).

Eine Anfrage für den Simulations-Service setzt sich aus mehreren Parametern zusammen. Wichtig dabei ist zu beachten das zwischen den Parametern Referenzen bestehen (Abbildung 5.1). Die Verweise bestehen aus den Pfadangaben. Da die Dateien der Parameter in das Simulationsverzeichnis kopiert

Listing 5.1 Beispiel SOAP Antwortnachricht mit Rückgabewert in Form einer Liste aus Strings

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Body>
    <ns2:getPotentialListResponse
        xmlns:ns2="http://imd.ws.uni-stuttgart.de/ImdPotentialsService/">
      <ns1:value xmlns:ns1="http://imd.ws.uni-stuttgart.de/ImdTypes">FECU2009.phi_r2</ns1:value>
      <ns1:value xmlns:ns1="http://imd.ws.uni-stuttgart.de/ImdTypes">FECU2009.F_rho</ns1:value>
      <ns1:value xmlns:ns1="http://imd.ws.uni-stuttgart.de/ImdTypes">FECU2009.rho_r2</ns1:value>
    </ns2:getPotentialListResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

Listing 5.2 Beispiel SOAP Nachricht für ein writeParameterFileRequest (gekürzt)

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
     xmlns:imd="http://imd.ws.uni-stuttgart.de/ImdParameterService/"
     xmlns:imd1="http://imd.ws.uni-stuttgart.de/ImdTypes">
   <soapenv:Header/>
   <soapenv:Body>
     <imd:writeParameterFileRequest>
       <imd1:key>coordname</imd1:key>
       <imd1:value>testbox1.dat</imd1:value>
       <imd1:key>outfiles</imd1:key>
       <imd1:value>glok</imd1:value>
       <imd1:key>ensemble</imd1:key>
       <imd1:value>glok</imd1:value>
       <imd1:key>maxsteps</imd1:key>
       <imd1:value>1000</imd1:value>
       <!-- [...] -->
       <imd1:key>timestep</imd1:key>
       <imd1:value>0.05</imd1:value>
       <imd1:key>relax_rate</imd1:key>
       <imd1:value>0.1</imd1:value>
       <imd1:key>core_potential_file</imd1:key>
       <imd1:value>FECU2009.phi_r2</imd1:value>
     </imd:writeParameterFileRequest>
   </soapenv:Body>
</soapenv:Envelope>
```

Listing 5.3 Beispiel SOAP Nachricht zum kompilieren eines imd_eam_glok_homdef_stress Moduls

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
     xmlns:imd="http://imd.ws.uni-stuttgart.de/ImdCompilerService/"
     xmlns:imd1="http://imd.ws.uni-stuttgart.de/ImdTypes">
   <soapenv:Header/>
   <soapenv:Body>
     <imd:writeBinaryRequest>
       <options>
         <imd1:value>eam</imd1:value>
         <imd1:value>glok</imd1:value>
         <imd1:value>homdef</imd1:value>
         <imd1:value>stress</imd1:value>
       </options>
     </imd:writeBinaryRequest>
   </soapenv:Body>
</soapenv:Envelope>
```

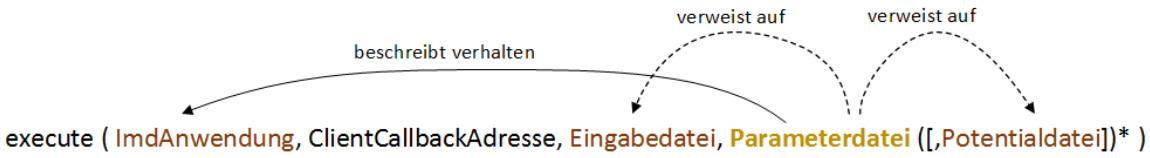


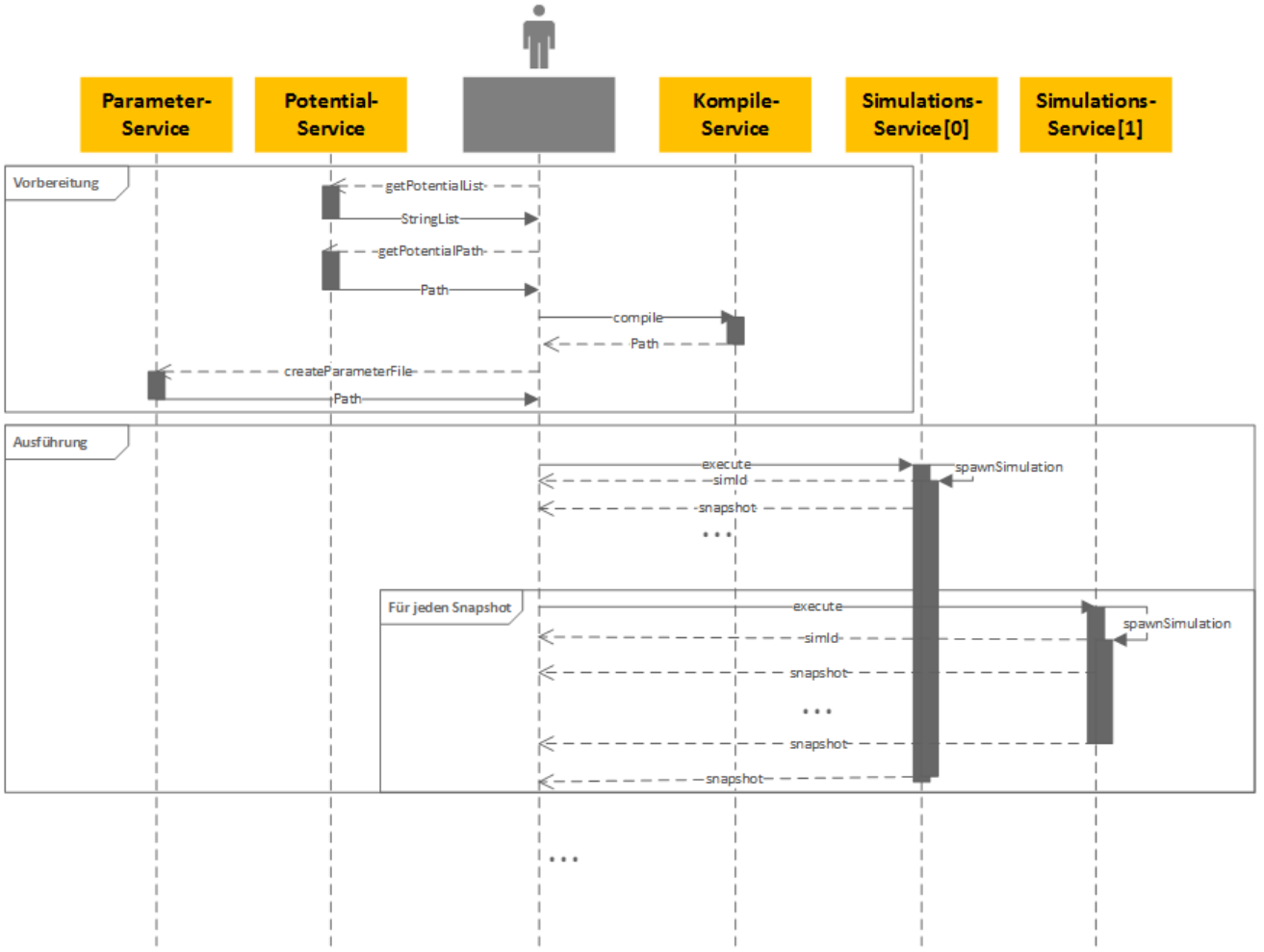
Abbildung 5.1.: Interne Verweise der execute Parameter vom Simulations-Service

Listing 5.4 Beispiel SOAP Antwortnachricht mit einer Datei als Rückgabewert (gekürzt)

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Body>
    <ns3:getParameterFileByNameResponse
        xmlns:ns3="http://imd.ws.uni-stuttgart.de/ImdParameterService/">
      <ns1:name xmlns:ns1="http://imd.ws.uni-stuttgart.de/ImdTypes">param_glok_mod</ns1:name>
      <ns1:description xmlns:ns1="http://imd.ws.uni-stuttgart.de/ImdTypes">parameter
          file</ns1:description>
      <ns1:file xmlns:ns1="http://imd.ws.uni-stuttgart.de/ImdTypes">Y29vcmRu<!-- [...]
          -->hZGVyIDEK</ns1:file>
    </ns3:getParameterFileByNameResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

werden reicht es relative Pfade anzugeben. In diesem Fall reicht es sogar aus nur die Dateinamen in der Parameterdatei anzugeben.

Es bieten sich zwei Möglichkeiten an eine Simulation mit Hilfe der Webservices durchzuführen. Sind die Webservice auf einem Server vorhanden können die einzelnen Ressourcen über die Pfade aufgelöst werden. Das hat den Vorteil das der Datenverkehr entfällt der beim Senden der Dateien anfallen würde. Ist dies nicht der Fall müssen die Dateien direkt übertragen werden. Das Sequenzdiagramm (Abbildung 5.2) zeigt einen möglichen Ablauf der Simulation aus Abschnitt 3.2 mit Hilfe der Webservices.



Die Webservices sind in diesem Diagramm alle auf dem selben Server deployed worden. Deshalb können die Dateien über Pfade aufgelöst werden.

Zur Besseren Übersicht existieren zwei Simulations-Service Instanzen.

Abbildung 5.2.: Beispiel einer Orchestrierung der Webservice Aufrufe

6. Zusammenfassung und Ausblick

Rechnergestützte Experimente in Form von Simulationen gehören oftmals zur wesentlichen Forschungsarbeit von Wissenschaftlern. Simulationen sind in den meisten Fällen aufwendig am Computer zu modellieren und auszuführen. Es erfordert entsprechendes Know-how um eine Simulation durchzuführen. Damit sich Wissenschaftler auf ihre Kernkompetenzen konzentrieren können, wird von dem Institut für Architektur und Anwendungssystemen ein Worfkflow Management System entwickelt. Das WfMS hat das Ziel, Wissenschaftlern das Durchführen von Simulationen ohne Programmierkenntnisse zu ermöglichen. Dafür werden existierende Simulationsanwendungen gekapselt und als Webservice bereitgestellt. Die Webservices können dann im WfMS zu einer Simulation kombiniert werden. Das WfMS übernimmt die automatische Ausführung und Verwaltung der Simulation, welche vorher manuell durchgeführt werden musste.

Für dieses WfMS wurde eine existierende Softwarebibliothek analysiert. Die Softwarebibliothek IMD wurde vom Institut für Materialprüfung, Werkstoffkunde und Festigkeitslehre bereitgestellt. Mit Hilfe der Bibliothek können verschiedene Simulationsanwendungen erstellt werden. Eine Simulationsanwendung besteht aus verschiedenen Modulen, welche aus der Simulationsbibliothek kompiliert werden kann. Unter anderem wird die Bibliothek für Simulationen im Bereich der Molekulardynamik verwendet. Die manuelle Benutzung erfordert, aufgrund der vielen Dateien die dabei erstellt werden, einen hohen Verwaltungs- und Konfigurationaufwand. Das soll sich mit der Bereitstellung der Bibliothek als Webservice ändern.

Ziel der Arbeit war es ein Webservice für die IMD Simulationsbibliothek zu entwerfen, der künftig in einen Workflow integriert werden kann. Aus den einzelnen manuellen Arbeitsschritten entstand jedoch nicht ein einzelner Webservice, sondern eine Gruppe von Webservices. Ein Webservice ist für das Zusammenbauen der Simulationsanwendung aus der Simulationsbibliothek zuständig. Zwei weitere Webservices verwaltet Konfigurations und Potentialdateien, welche für die Durchführung der Simulation benötigt werden. Der vierte Webservice führt die Simulationsanwendung abhängig von der eingegebenen Konfiguration aus. Dabei unterstützt der Webservice eine parallele Ausführung mittels MPI. Das erlaubt zukünftig schneller zu Berechnungsergebnissen zu kommen. Die Aufteilung in kleinere Teilservice ermöglicht eine flexiblere Verwendung. So dass der rechenintensive Teil auf Hochleistungsrechnern platziert werden kann.

Durch das Anbieten der Bibliothek als Webservice wird außerdem die Zusammenarbeit an Simulationen vereinfacht. So werden einmal erstellte Konfigurationen vom Webservice archiviert und können von anderen Wissenschaftlern jederzeit abgerufen werden. Auch die während der Simulation erstellten Dateien sind auf diesem Weg einfach abrufbar. Es ist lediglich die Simulations-Identifikationsnummer und die Adresse des Webservices weiterzugeben um entsprechende Dateien abrufen zu können.

Ausblick

Die Kapselung der Bibliothek als Webservice war der erste Schritt um diese in einen Workflow einbinden zu können. Eine weitere Arbeit die das Integrieren in einen Workflow übernimmt ist bereits am Institut für Architektur und Anwendungssysteme am Entstehen. In der folgenden Arbeit werden die unterschiedlichen IMD Webservices miteinander verbunden um eine automatische Durchführung der Simulation zu ermöglichen. Dabei kommen weitere Simulationsanwendungen zum Einsatz. Mit Hilfe der Modellierung als Workflow lassen sich die Simulationen künftig einfach und schnell durchführen.

Um die Möglichkeiten für verschiedene Simulationen weiter zu erhöhen, bietet es sich an weitere Simulationsanwendungen als Webservice bereitzustellen. Aber auch weitere Anwendungen für die Vor- und Nachbereitung der Simulationen sind wünschenswert. Die vorhandenen Werkzeuge von IMD werden derzeit nicht als Webservice angeboten. Diese Werkzeuge könnten in folgenden Arbeiten gekapselt und zusätzlich angeboten werden. Ein weiteres Ziel stellt die automatisierte Analyse der Ergebnisse dar. An Hand der Ergebnisse könnte die Simulation lernen und eine neue Simulationsinstanz mit veränderten Parametern anstoßen um ein besseres Ergebnis zu erhalten.

A. Anhang

A.1. Abkürzungsverzeichnis

aar	Axis Archiv
API	Application Programming Interface
Bash	Bourne-again shell
BPEL	Business Process Execution Language
CSV	Comma-separated values
EAM	Embedded Atom Method
HTTP	Hypertext Transfer Protocol
IAAS	Institut für Architektur von Anwendungssystemen
IMWF	Institut für Materialprüfung,Werkstoffkunde und Festigkeitslehre
IMD	ITAP Molecular Dynamics
ITAP	Institut für Theoretische und Angewandte Physik
jar	Java Archiv
JSON	JavaScript Object Notation
JMS	Java Message Service
SimTech	Simulation Technologie
SLA	Service-Level-Agreement
SMTP	Simple Mail Transfer Protocol
SOA	Service orientierte Architektur
SPoF	Single Point of Failure
TCP	Transmission Control Protocol
MPI	Message Passing Interface
OASIS	Organization for the Advancement of Structured Information Standards
Opal	Ostwald - Ripening of Precipitates on an Atomic Lattice

W3C World Wide Web Consortium

WfMs Workflow-Management-System

WS Webservice

WSDL Web Services Description Language

WST Webservice Technologie

XML Extensible Markup Language

XSD XML Schema Definition

A.2. Formale Webservice Beschreibungen

Es folgen die Definitionen der Webservices mit einer kurzen Beschreibung. Des weiteren werden die verfügbaren Operationen aufgelistet und die verwendeten Typen aufgeführt.

A.2.1. Webservice Elemente und Typen XSD

Die folgenden selbst definierten Elemente und Typen werden in den Webservices importiert und verwendet.

XSD

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <schema targetNamespace="http://imd.ws.uni-stuttgart.de/ImdTypes"
3        elementFormDefault="qualified" xmlns="http://www.w3.org/2001/XMLSchema"
4        xmlns:tns="http://imd.ws.uni-stuttgart.de/ImdTypes">
5
6        <annotation>
7              <documentation>Enthält alle gemeinsame Elemente und Typen für die IMD
                   Webservices.</documentation>
8        </annotation>
9
10       <simpleType name="path">
11       <annotation>
12             <documentation>Ein Dateipfad.</documentation>
13       </annotation>
14       <restriction base="string"></restriction>
15       </simpleType>
16
17       <simpleType name="statusCode">
18       <annotation>
19             <documentation>Statuscode für Antworten.</documentation>
20       </annotation>
21       <restriction base="int"></restriction>
22       </simpleType>
23
24       <complexType name="stringListType">
25       <annotation>
26             <documentation>Eine Liste von Strings.</documentation>
27       </annotation>
28       <sequence minOccurs="0" maxOccurs="unbounded">
29                   <element name="value" type="string"></element>
30             </sequence>
31       </complexType>
32
33       <complexType name="exceptionType">
34       <annotation>
35             <documentation>Eine Ausnahme.</documentation>
36       </annotation>
37       <sequence>
38                   <element name="errno" type="int"></element>
```

```
39                <element name="error" type="string"></element>
40          </sequence>
41    </complexType>
42
43    <complexType name="stringKeyValueArrayType">
44    <annotation>
45          <documentation>Ein Hashmap von Strings.</documentation>
46    </annotation>
47    <sequence minOccurs="0" maxOccurs="unbounded">
48                <element name="key" type="string"></element>
49                <element name="value" type="string"></element>
50          </sequence>
51    </complexType>
52
53    <complexType name="binaryType">
54    <annotation>
55          <documentation>Eine Binärdatei.</documentation>
56    </annotation>
57    <sequence>
58                <element name="name" type="string" maxOccurs="1" minOccurs="0"></element>
59                <element name="description" type="string" maxOccurs="1" minOccurs="0"></element>
60                <element name="file" type="base64Binary"></element>
61          </sequence>
62    </complexType>
63
64    <complexType name="binaryListType">
65    <annotation>
66          <documentation>Eine Liste von Binärdateien.</documentation>
67    </annotation>
68    <sequence minOccurs="0" maxOccurs="unbounded">
69                <element name="binary" type="tns:binaryType"></element>
70          </sequence>
71    </complexType>
72 </schema>
```

Listing A.1: Webservice Elemente und Typen XSD

A.2.2. Potential-Service WSDL

Der Potential-Service bietet Operationen zum Verwalten der Potentialdateien an. Dazu gehört das Anlegen, Löschen sowie das Empfangen von Dateien.

WSDL

```
1  <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2  <definitions name="ImdPotentials" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
3        xmlns:tns="http://imd.ws.uni-stuttgart.de/ImdPotentialsService/"
             xmlns="http://schemas.xmlsoap.org/wsdl/"
4        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
             targetNamespace="http://imd.ws.uni-stuttgart.de/ImdPotentialsService/">
5        <documentation>Verwaltet die Potentialdateien für die IMD Webservices.</documentation>
6        <types>
7              <xsd:schema targetNamespace="http://imd.ws.uni-stuttgart.de/ImdPotentialsService/"
8                    xmlns:imd="http://imd.ws.uni-stuttgart.de/ImdTypes">
```

```
9                 <xsd:import schemaLocation="imd_types.xsd"
10                      namespace="http://imd.ws.uni-stuttgart.de/ImdTypes">
11                </xsd:import>
12
13                <xsd:element name="getPotentialListResponse" type="imd:stringListType" />
14                <xsd:element name="getPotentialFileByNameRequest" type="xsd:string" />
15                <xsd:element name="getPotentialFileByNameResponse" type="imd:binaryType" />
16                <xsd:element name="getPotentialFileByNameFault" type="tns:potentialException" />
17                <xsd:element name="getPotentialPathByNameRequest" type="xsd:string" />
18                <xsd:element name="getPotentialPathByNameResponse" type="imd:path" />
19                <xsd:element name="getPotentialPathByNameFault" type="tns:potentialException" />
20                <xsd:element name="writePotentialRequest" type="imd:binaryType" />
21                <xsd:element name="writePotentialResponse" type="xsd:string" />
22                <xsd:element name="writePotentialFault" type="tns:potentialException" />
23                <xsd:element name="deletePotentialRequest" type="xsd:string" />
24                <xsd:element name="deletePotentialResponse" type="xsd:string" />
25                <xsd:element name="deletePotentialFault" type="tns:potentialException" />
26
27                <xsd:complexType name="potentialException">
28                      <xsd:complexContent>
29                            <xsd:extension base="imd:exceptionType"></xsd:extension>
30                      </xsd:complexContent>
31                </xsd:complexType>
32          </xsd:schema>
33    </types>
34
35    <message name="getPotentialListResponse">
36          <part name="parameters" element="tns:getPotentialListResponse" />
37    </message>
38    <message name="getPotentialFileByNameRequest">
39          <part name="parameters" element="tns:getPotentialFileByNameRequest" />
40    </message>
41    <message name="getPotentialFileByNameResponse">
42          <part name="parameters" element="tns:getPotentialFileByNameResponse" />
43    </message>
44    <message name="getPotentialFileByNameFault">
45          <part name="fault" element="tns:getPotentialFileFault" />
46    </message>
47    <message name="getPotentialPathByNameRequest">
48          <part name="parameters" element="tns:getPotentialPathByNameRequest" />
49    </message>
50    <message name="getPotentialPathByNameResponse">
51          <part name="parameters" element="tns:getPotentialPathByNameResponse" />
52    </message>
53    <message name="getPotentialPathByNameFault">
54          <part name="fault" element="tns:getPotentialPathByNameFault" />
55    </message>
56    <message name="getPotentialListRequest"></message>
57    <message name="writePotentialRequest">
58          <part name="parameters" element="tns:writePotentialRequest" />
59    </message>
60    <message name="writePotentialResponse">
61          <part name="parameters" element="tns:writePotentialResponse" />
62    </message>
63    <message name="writePotentialFault">
```

```
64            <part name="fault" element="tns:writePotentialFault" />
65      </message>
66      <message name="deletePotentialRequest">
67            <part name="parameters" element="tns:deletePotentialRequest" />
68      </message>
69      <message name="deletePotentialResponse">
70            <part name="parameters" element="tns:deletePotentialResponse" />
71      </message>
72      <message name="deletePotentialFault">
73            <part name="fault" element="tns:deletePotentialFault" />
74      </message>
75
76      <portType name="ImdPotentialPortType">
77            <operation name="getPotentialList">
78                  <documentation>Liefert eine Liste von verfügbaren
                        Potentialdateien</documentation>
79                  <input message="tns:getPotentialListRequest" />
80                  <output message="tns:getPotentialListResponse" />
81            </operation>
82            <operation name="getPotentialFileByName">
83                  <documentation>Erwartet als Eingabe einen Namen der verfügbaren
                        Potentialdateien.
84                        Gibt als Antwort die Potentialdatei zurück.
85                        Wirft eine Exception, wenn keine Datei gefunden wurde.
86                  </documentation>
87                  <input message="tns:getPotentialFileByNameRequest" />
88                  <output message="tns:getPotentialFileByNameResponse" />
89                  <fault message="tns:getPotentialFileByNameFault" name="fault" />
90            </operation>
91            <operation name="getPotentialPathByName">
92                  <documentation>Erwartet als Eingabe einen Namen der verfügbaren
                        Potentialdateien.
93                        Gibt als Antwort den Pfad zu dieser Potentialdatei zurück.
94                        Wirft eine Exception, wenn keine Datei gefunden wurde.
95                  </documentation>
96                  <input message="tns:getPotentialPathByNameRequest" />
97                  <output message="tns:getPotentialPathByNameResponse" />
98                  <fault message="tns:getPotentialPathByNameFault" name="fault" />
99            </operation>
100           <operation name="writePotential">
101                 <documentation>Speichert eine Potentialdatei.
102                       Wirft eine Exception falls das Speicher nicht erfolgreich war.
103                 </documentation>
104                 <input message="tns:writePotentialRequest" />
105                 <output message="tns:writePotentialResponse" />
106                 <fault name="fault" message="tns:writePotentialFault" />
107           </operation>
108           <operation name="deletePotential">
109                 <documentation>Löscht eine Potentaldatei.
110                       Wirft eine Exception falls die Datei nicht gelöscht werden kann.
111                 </documentation>
112                 <input message="tns:deletePotentialRequest" />
113                 <output message="tns:deletePotentialResponse" />
114                 <fault name="fault" message="tns:deletePotentialFault" />
115           </operation>
```

```
116       </portType>
117
118       <binding name="ImdPotentialSOAP" type="tns:ImdPotentialPortType">
119             <soap:binding style="document"
120                   transport="http://schemas.xmlsoap.org/soap/http" />
121             <operation name="getPotentialList">
122                   <soap:operation
123                         soapAction="urn:getPotentialList" />
124                   <input>
125                         <soap:body use="literal" />
126                   </input>
127                   <output>
128                         <soap:body use="literal" />
129                   </output>
130             </operation>
131             <operation name="getPotentialFileByName">
132                   <soap:operation
133                         soapAction="urn:getPotentialFileByName" />
134                   <input>
135                         <soap:body use="literal" />
136                   </input>
137                   <output>
138                         <soap:body use="literal" />
139                   </output>
140                   <fault name="fault">
141                         <soap:fault use="literal" name="fault" />
142                   </fault>
143             </operation>
144             <operation name="getPotentialPathByName">
145                   <soap:operation
146                         soapAction="urn:getPotentialPathByName" />
147                   <input>
148                         <soap:body use="literal" />
149                   </input>
150                   <output>
151                         <soap:body use="literal" />
152                   </output>
153                   <fault name="fault">
154                         <soap:fault use="literal" name="fault" />
155                   </fault>
156             </operation>
157             <operation name="writePotential">
158                   <soap:operation
159                         soapAction="urn:writePotential" />
160                   <input>
161                         <soap:body use="literal" />
162                   </input>
163                   <output>
164                         <soap:body use="literal" />
165                   </output>
166                   <fault name="fault">
167                         <soap:fault use="literal" name="fault" />
168                   </fault>
169             </operation>
170             <operation name="deletePotential">
```

```
171                   <soap:operation
172                         soapAction="urn:deletePotential" />
173                   <input>
174                         <soap:body use="literal" />
175                   </input>
176                   <output>
177                         <soap:body use="literal" />
178                   </output>
179                   <fault name="fault">
180                         <soap:fault use="literal" name="fault" />
181                   </fault>
182             </operation>
183       </binding>
184
185       <service name="ImdPotentialService">
186             <port binding="tns:ImdPotentialSOAP" name="ImdPotentialSOAP">
187                   <soap:address location="http://localhost:8080/ImdPotentialsService" />
188             </port>
189       </service>
190 </definitions>
```

Listing A.2: Potential-Service WSDL

A.2.3. Parameter-Service WSDL

Der Parameter-Service verwaltet die Parameterdateien für die Simulationen. Es können vorhandene Dateien angezeigt oder neue erzeugt werden. Die bestehenden Dateien können gelöscht oder empfangen werden.

WSDL

```
1  <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2  <definitions name="ImdParameter" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
3        xmlns:tns="http://imd.ws.uni-stuttgart.de/ImdParameterService/"
           xmlns="http://schemas.xmlsoap.org/wsdl/"
4        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
           targetNamespace="http://imd.ws.uni-stuttgart.de/ImdParameterService/">
5        <documentation>Verwaltet die Parameterdateien für die IMD Webservices.</documentation>
6        <types>
7              <xsd:schema targetNamespace="http://imd.ws.uni-stuttgart.de/ImdParameterService/"
8                    xmlns:imd="http://imd.ws.uni-stuttgart.de/ImdTypes">
9                    <xsd:import schemaLocation="imd_types.xsd"
10                         namespace="http://imd.ws.uni-stuttgart.de/ImdTypes">
11                   </xsd:import>
12
13                   <xsd:element name="writeParameterFileRequest"
                         type="imd:stringKeyValueArrayType" />
14                   <xsd:element name="writeParameterFileResponse" type="imd:path" />
15                   <xsd:element name="writeParameterFileFault" type="tns:parameterException" />
16
17                   <xsd:element name="getParameterFileByNameRequest" type="xsd:string" />
18                   <xsd:element name="getParameterFileByNameResponse" type="imd:binaryType" />
19                   <xsd:element name="getParameterFileByNameFault" type="tns:parameterException" />
```

```
20
21                <xsd:element name="getParameterPathByNameRequest" type="xsd:string" />
22                <xsd:element name="getParameterPathByNameResponse" type="imd:path" />
23                <xsd:element name="getParameterPathByNameFault" type="tns:parameterException" />
24
25                <xsd:element name="getParameterListResponse" type="imd:stringListType" />
26
27                <xsd:element name="deleteParameterFileRequest" type="xsd:string" />
28                <xsd:element name="deleteParameterFileResponse" type="xsd:string" />
29                <xsd:element name="deleteParameterFileFault" type="xsd:string" />
30
31                <xsd:complexType name="parameterException">
32                      <xsd:complexContent>
33                            <xsd:extension base="imd:exceptionType"></xsd:extension>
34                      </xsd:complexContent>
35                </xsd:complexType>
36          </xsd:schema>
37    </types>
38
39    <message name="writeParameterFileRequest">
40          <part element="tns:writeParameterFileRequest" name="parameters" />
41    </message>
42    <message name="writeParameterFileResponse">
43          <part element="tns:writeParameterFileResponse" name="parameters" />
44    </message>
45    <message name="writeParameterFileFault">
46          <part name="parameters" element="tns:writeParameterFileFault" />
47    </message>
48    <message name="getParameterFileByNameRequest">
49          <part name="parameters" element="tns:getParameterFileByNameRequest" />
50    </message>
51    <message name="getParameterFileByNameResponse">
52          <part name="parameters" element="tns:getParameterFileByNameResponse" />
53    </message>
54    <message name="getParameterFileByNameFault">
55          <part name="fault" element="tns:getParameterFileByNameFault" />
56    </message>
57    <message name="getParameterListRequest"></message>
58    <message name="getParameterListResponse">
59          <part name="parameters" element="tns:getParameterListResponse" />
60    </message>
61    <message name="getParameterPathByNameRequest">
62          <part name="parameters" element="tns:getParameterPathByNameRequest" />
63    </message>
64    <message name="getParameterPathByNameResponse">
65          <part name="parameters" element="tns:getParameterPathByNameResponse" />
66    </message>
67    <message name="getParameterPathByNameFault">
68          <part name="fault" element="tns:getParameterPathByNameFault" />
69    </message>
70    <message name="deleteParameterFileRequest">
71          <part name="parameters" element="tns:deleteParameterFileRequest" />
72    </message>
73    <message name="deleteParameterFileResponse">
74          <part name="parameters" element="tns:deleteParameterFileResponse" />
```

```
75        </message>
76        <message name="deleteParameterFileFault">
77              <part name="fault" element="tns:deleteParameterFileFault" />
78        </message>
79
80        <portType name="ImdParameterPortType">
81              <operation name="writeParameterFile">
82                    <documentation>Speichert eine Parameterdatei um sie später abzurufen.
83                          Wirft eine Exception falls das Schreiben unmöglich ist.
84                    </documentation>
85                    <input message="tns:writeParameterFileRequest" />
86                    <output message="tns:writeParameterFileResponse" />
87                    <fault name="fault" message="tns:writeParameterFileFault" />
88              </operation>
89              <operation name="getParameterFileByName">
90                    <documentation>Erwartet einen Dateinamen. Liefert als Antwort die
                          Parameterdatei.
91                          Wirft eine Exception, wenn die Datei nicht vorhanden ist.
92                    </documentation>
93                    <input message="tns:getParameterFileByNameRequest" />
94                    <output message="tns:getParameterFileByNameResponse" />
95                    <fault name="fault" message="tns:getParameterFileByNameFault" />
96              </operation>
97              <operation name="getParameterList">
98                    <documentation>Liefert eine Liste von Namen aller verfügbaren Parameterdateien.
99                    </documentation>
100                   <input message="tns:getParameterListRequest" />
101                   <output message="tns:getParameterListResponse" />
102             </operation>
103             <operation name="getParameterPathByName">
104                   <documentation>Erwartet einen Dateinamen.
105                         Liefert als Antwort den Pfad zu dieser Datei.
106                         Wirft eine Exception, wenn Datei nicht vorhanden ist.
107                   </documentation>
108                   <input message="tns:getParameterPathByNameRequest" />
109                   <output message="tns:getParameterPathByNameResponse" />
110                   <fault name="fault" message="tns:getParameterPathByNameFault" />
111             </operation>
112             <operation name="deleteParameterFile">
113                   <documentation>Löscht eine Parameterdatei.
114                         Wirft eine Exception falls das Löschen fehlschlägt.
115                   </documentation>
116                   <input message="tns:deleteParameterFileRequest" />
117                   <output message="tns:deleteParameterFileResponse" />
118                   <fault name="fault" message="tns:deleteParameterFileFault" />
119             </operation>
120       </portType>
121
122       <binding name="ImdParameterSOAP" type="tns:ImdParameterPortType">
123             <soap:binding style="document"
124                   transport="http://schemas.xmlsoap.org/soap/http" />
125             <operation name="writeParameterFile">
126                   <soap:operation
127                         soapAction="urn:writeParameterFile" />
128                   <input>
```

```
129                     <soap:body use="literal" />
130               </input>
131               <output>
132                     <soap:body use="literal" />
133               </output>
134               <fault name="fault">
135                     <soap:fault use="literal" name="fault" />
136               </fault>
137         </operation>
138         <operation name="getParameterFileByName">
139               <soap:operation
140                     soapAction="urn:getParameterFileByName" />
141               <input>
142                     <soap:body use="literal" />
143               </input>
144               <output>
145                     <soap:body use="literal" />
146               </output>
147               <fault name="fault">
148                     <soap:fault use="literal" name="fault" />
149               </fault>
150         </operation>
151         <operation name="getParameterList">
152               <soap:operation
153                     soapAction="urn:getParameterList" />
154               <input>
155                     <soap:body use="literal" />
156               </input>
157               <output>
158                     <soap:body use="literal" />
159               </output>
160         </operation>
161         <operation name="getParameterPathByName">
162               <soap:operation
163                     soapAction="urn:getParameterPathByName" />
164               <input>
165                     <soap:body use="literal" />
166               </input>
167               <output>
168                     <soap:body use="literal" />
169               </output>
170               <fault name="fault">
171                     <soap:fault use="literal" name="fault" />
172               </fault>
173         </operation>
174         <operation name="deleteParameterFile">
175               <soap:operation
176                     soapAction="urn:deleteParameterFile" />
177               <input>
178                     <soap:body use="literal" />
179               </input>
180               <output>
181                     <soap:body use="literal" />
182               </output>
183               <fault name="fault">
```

```
184                         <soap:fault use="literal" name="fault" />
185                   </fault>
186             </operation>
187       </binding>
188
189       <service name="ImdParameterService">
190             <port binding="tns:ImdParameterSOAP" name="ImdParameterSOAP">
191                   <soap:address location="http://localhost:8080/ImdParameterService" />
192             </port>
193       </service>
194 </definitions>
```

Listing A.3: Parameter-Service WSDL

A.2.4. Kompilier-Service WSDL

Der Kompilier-Service baut die angeforderten Module der Bibliothek zu einer ausführbaren Datei zusammen. Bereits erstellte Anwendungen können ausgelesen oder gelöscht werden.

WSDL

```
1  <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2  <definitions name="ImdCompiler" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
3        xmlns:tns="http://imd.ws.uni-stuttgart.de/ImdCompilerService/"
           xmlns="http://schemas.xmlsoap.org/wsdl/"
4        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
           targetNamespace="http://imd.ws.uni-stuttgart.de/ImdCompilerService/">
5        <documentation>Kompiliert und verwaltet die IMD Binärdateien.</documentation>
6        <types>
7              <xsd:schema targetNamespace="http://imd.ws.uni-stuttgart.de/ImdCompilerService/"
8                    xmlns:imd="http://imd.ws.uni-stuttgart.de/ImdTypes">
9                    <xsd:import schemaLocation="imd_types.xsd"
10                         namespace="http://imd.ws.uni-stuttgart.de/ImdTypes">
11                   </xsd:import>
12
13                   <xsd:element name="compileBinaryRequest" type="imd:stringListType" />
14                   <xsd:element name="compileBinaryResponse" type="xsd:string" />
15                   <xsd:element name="compileBinaryFault" type="tns:compileException" />
16                   <xsd:element name="getBinaryFileByNameRequest" type="xsd:string" />
17                   <xsd:element name="getBinaryFileByNameResponse" type="imd:binaryType" />
18                   <xsd:element name="getBinaryFileByNameFault" type="tns:compileException" />
19                   <xsd:element name="deleteBinaryByNameRequest" type="xsd:string" />
20                   <xsd:element name="deleteBinaryByNameResponse" type="xsd:string" />
21                   <xsd:element name="deleteBinaryByNameFault" type="tns:compileException" />
22                   <xsd:element name="listBinarysResponse" type="imd:stringListType" />
23                   <xsd:element name="getBinaryPathByNameRequest" type="xsd:string" />
24                   <xsd:element name="getBinaryPathByNameResponse" type="imd:path" />
25                   <xsd:element name="getBinaryPathByNameFault" type="tns:compileException" />
26
27                   <xsd:complexType name="compileException">
28                         <xsd:complexContent>
29                               <xsd:extension base="imd:exceptionType"></xsd:extension>
30                         </xsd:complexContent>
31                   </xsd:complexType>
```

```
32            </xsd:schema>
33      </types>
34
35      <message name="compileBinaryRequest">
36            <part name="parameters" element="tns:compileBinaryRequest" />
37      </message>
38      <message name="compileBinaryResponse">
39            <part name="parameters" element="tns:compileBinaryResponse" />
40      </message>
41      <message name="compileBinaryFault">
42            <part name="fault" element="tns:compileBinaryFault" />
43      </message>
44      <message name="getBinaryFileByNameRequest">
45            <part name="parameters" element="tns:getBinaryFileByNameRequest" />
46      </message>
47      <message name="getBinaryFileByNameResponse">
48            <part name="parameters" element="tns:getBinaryFileByNameResponse" />
49      </message>
50      <message name="getBinaryFileByNameFault">
51            <part name="fault" element="tns:getBinaryFileByNameFault" />
52      </message>
53      <message name="deleteBinaryByNameRequest">
54            <part name="parameters" element="tns:deleteBinaryByNameRequest" />
55      </message>
56      <message name="deleteBinaryByNameResponse">
57            <part name="deleteBinaryByNameResponse" element="tns:deleteBinaryByNameResponse" />
58      </message>
59      <message name="listBinarysRequest"></message>
60      <message name="listBinarysResponse">
61            <part name="parameters" element="tns:listBinarysResponse" />
62      </message>
63      <message name="getBinaryPathByNameRequest">
64            <part name="parameters" element="tns:getBinaryPathByNameRequest" />
65      </message>
66      <message name="getBinaryPathByNameResponse">
67            <part name="parameters" element="tns:getBinaryPathByNameResponse" />
68      </message>
69      <message name="deleteBinaryByNameFault">
70            <part name="fault" element="tns:deleteBinaryByNameFault" />
71      </message>
72      <message name="getBinaryPathByNameFault">
73            <part name="fault" element="tns:getBinaryPathByNameFault" />
74      </message>
75
76      <portType name="ImdCompilerPortType">
77            <operation name="compileBinary">
78                  <documentation>Erwartet eine IMD Modulliste.
79                        Liefert nach Fertigstellung den Namen der Binärdatei.
80                        Wirft eine Exception bei einem Kompilierfehler.
81                  </documentation>
82                  <input message="tns:compileBinaryRequest" />
83                  <output message="tns:compileBinaryResponse" />
84                  <fault name="fault" message="tns:compileBinaryFault" />
85            </operation>
86            <operation name="getBinaryFileByName">
```

```
87                <documentation>Erwartet einen Dateinamen.
88                      Liefert die Datei.
89                      Wirft eine Exception falls die Datei nicht vorhanden ist.
90                </documentation>
91                <input message="tns:getBinaryFileByNameRequest" />
92                <output message="tns:getBinaryFileByNameResponse" />
93                <fault name="fault" message="tns:getBinaryFileByNameFault" />
94          </operation>
95          <operation name="deleteBinaryByName">
96                <documentation>Erwartet einen Dateinamen.
97                      Löscht eine Binärdatei.
98                      Wirft eine Exception falls die Datei nicht gelöscht werden kann.
99                </documentation>
100               <input message="tns:deleteBinaryByNameRequest" />
101               <output message="tns:deleteBinaryByNameResponse" />
102               <fault name="fault" message="tns:deleteBinaryByNameFault" />
103         </operation>
104         <operation name="listBinarys">
105               <documentation>Liefert eine Liste mit allen vorhandenen
                      Binärdateinamen.</documentation>
106               <input message="tns:listBinarysRequest" />
107               <output message="tns:listBinarysResponse" />
108         </operation>
109         <operation name="getBinaryPathByName">
110               <documentation>Erwatet einen Dateiname.
111                     Liefert den Pfad zur Binärdatei.
112                     Wirft eine Exception falls die Datei nicht vorhanden ist.
113               </documentation>
114               <input message="tns:getBinaryPathByNameRequest" />
115               <output message="tns:getBinaryPathByNameResponse" />
116               <fault name="fault" message="tns:getBinaryPathByNameFault" />
117         </operation>
118   </portType>
119
120   <binding name="ImdCompilerSOAP" type="tns:ImdCompilerPortType">
121         <soap:binding style="document"
122               transport="http://schemas.xmlsoap.org/soap/http" />
123         <operation name="compileBinary">
124               <soap:operation soapAction="urn:compileBinary" />
125               <input>
126                     <soap:body use="literal" />
127               </input>
128               <output>
129                     <soap:body use="literal" />
130               </output>
131               <fault name="fault">
132                     <soap:fault use="literal" name="fault" />
133               </fault>
134         </operation>
135         <operation name="getBinaryFileByName">
136               <soap:operation soapAction="urn:getBinaryFileByName" />
137               <input>
138                     <soap:body use="literal" />
139               </input>
140               <output>
```

```
141                         <soap:body use="literal" />
142                   </output>
143                   <fault name="fault">
144                         <soap:fault use="literal" name="fault" />
145                   </fault>
146             </operation>
147             <operation name="deleteBinaryByName">
148                   <soap:operation soapAction="urn:deleteBinaryByName" />
149                   <input>
150                         <soap:body use="literal" />
151                   </input>
152                   <output>
153                         <soap:body use="literal" />
154                   </output>
155                   <fault name="fault">
156                         <soap:fault use="literal" name="fault" />
157                   </fault>
158             </operation>
159             <operation name="listBinarys">
160                   <soap:operation soapAction="urn:listBinarys" />
161                   <input>
162                         <soap:body use="literal" />
163                   </input>
164                   <output>
165                         <soap:body use="literal" />
166                   </output>
167             </operation>
168             <operation name="getBinaryPathByName">
169                   <soap:operation soapAction="urn:getBinaryPathByName" />
170                   <input>
171                         <soap:body use="literal" />
172                   </input>
173                   <output>
174                         <soap:body use="literal" />
175                   </output>
176                   <fault name="fault">
177                         <soap:fault use="literal" name="fault" />
178                   </fault>
179             </operation>
180       </binding>
181
182       <service name="ImdCompilerService">
183             <port binding="tns:ImdCompilerSOAP" name="ImdCompilerSOAP">
184                   <soap:address location="http://localhost:8080/ImdCompilerService" />
185             </port>
186       </service>
187 </definitions>
```

Listing A.4: Kompilier-Service WSDL

A.2.5. Simulations-Service WSDL

Der Simulations-Service ist der Kern der IMD-Webservices. Er erlaubt das Ausführen von Simulationen mit Hilfe der zuvor erzeugten IMD Anwendungen und Parameterdateien. Während die Simulation ausgeführt wird werden asynchron Nachrichten an den Klienten geschickt.

WSDL

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<definitions name="ImdExecuter" xmlns="http://schemas.xmlsoap.org/wsdl/"
      xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
          xmlns:tns="http://imd.ws.uni-stuttgart.de/ImdExecuterService/"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
          targetNamespace="http://imd.ws.uni-stuttgart.de/ImdExecuterService/">
      <documentation>Verwaltet die IMD Simulationen und führt diese aus.</documentation>
      <types>
            <xsd:schema targetNamespace="http://imd.ws.uni-stuttgart.de/ImdExecuterService/"
                  xmlns:imd="http://imd.ws.uni-stuttgart.de/ImdTypes">
                  <xsd:import schemaLocation="imd_types.xsd"
                        namespace="http://imd.ws.uni-stuttgart.de/ImdTypes"></xsd:import>

                  <xsd:element name="executeRequest" type="tns:executeRequest" />
                  <xsd:element name="executeResponse" type="tns:simulationId" />
                  <xsd:element name="executeFault" type="tns:executeException" />
                  <xsd:element name="abortRequest" type="tns:simulationId" />
                  <xsd:element name="abortResponse" type="imd:statusCode" />
                  <xsd:element name="listFilesBySimulationIdRequest" type="tns:simulationId" />
                  <xsd:element name="listFilesBySimulationIdResponse" type="imd:stringListType" />
                  <xsd:element name="getFileByNameAndSimulationIdRequest"
                        type="tns:fileSimulationTupel" />
                  <xsd:element name="getFileByNameAndSimulationIdResponse"
                        type="imd:binaryType" />
                  <xsd:element name="getFileByNameAndSimulationIdFault"
                        type="tns:executeException" />
                  <xsd:element name="deleteSimulationRequest" type="tns:simulationId" />
                  <xsd:element name="deleteSimulationResponse" type="imd:statusCode" />
                  <xsd:element name="deleteSimulationFault" type="tns:executeException" />

                  <xsd:element name="notifyCallback" type="xsd:string" />
                  <xsd:element name="newFileCallback" type="xsd:string" />
                  <xsd:element name="checkpointCallback" type="xsd:string" />
                  <xsd:element name="outputlogCallback" type="xsd:string" />
                  <xsd:element name="errorlogCallback" type="xsd:string" />

                  <xsd:complexType name="executeException">
                        <xsd:complexContent>
                              <xsd:extension base="imd:exceptionType"></xsd:extension>
                        </xsd:complexContent>
                  </xsd:complexType>

                  <xsd:simpleType name="simulationId">
                        <xsd:restriction base="xsd:string"></xsd:restriction>
                  </xsd:simpleType>
```

```
44
45                <!-- Identifikation einer Datei und Simulation -->
46                <xsd:complexType name="fileSimulationTupel">
47                      <xsd:sequence>
48                            <xsd:element name="name" type="xsd:string" />
49                            <xsd:element name="simulationId" type="tns:simulationId" />
50                      </xsd:sequence>
51                </xsd:complexType>
52
53                <!-- Polymorphie um xs:choise Konstrukt zu vermeiden -->
54                <complexType abstract="true" name="absctractExecuteRequest">
55                      <sequence>
56                            <xsd:element name="simulationfile" type="imd:binaryType" />
57                            <xsd:element name="callbackEpr" type="xsd:string" />
58                            <xsd:element name="potentialFile" type="imd:binaryType"
59                                  maxOccurs="unbounded" minOccurs="0" />
60                            <xsd:element name="checkpoint" type="xsd:int"
61                                  maxOccurs="1" minOccurs="0" />
62                      </sequence>
63                </complexType>
64
65                <!-- Ein Execute Request mit Pfadangaben -->
66                <complexType name="executeRequestPath">
67                      <complexContent>
68                            <extension base="tns:absctractExecuteRequest">
69                                  <sequence>
70                                        <xsd:element name="binaryPath" type="imd:path" />
71                                        <xsd:element name="parameterPath" type="imd:path" />
72                                  </sequence>
73                            </extension>
74                      </complexContent>
75                </complexType>
76
77                <!-- Ein Execute Request mit Binärdateien -->
78                <complexType name="executeRequestBinary">
79                      <complexContent>
80                            <extension base="tns:absctractExecuteRequest">
81                                  <sequence>
82                                        <xsd:element name="binaryFile" type="imd:binaryType" />
83                                        <xsd:element name="parameterFile" type="imd:binaryType"
                                              />
84                                  </sequence>
85                            </extension>
86                      </complexContent>
87                </complexType>
88          </xsd:schema>
89      </types>
90
91      <!-- Asynchrone Nachrichten für den Klienten -->
92      <message name="checkpointCallback">
93            <part name="parameters" element="tns:checkpointCallback" />
94      </message>
95      <message name="newFileCallback">
96            <part name="parameters" element="tns:newFileCallback" />
97      </message>
```

```
98        <message name="notifyCallback">
99              <part name="parameters" element="tns:notifyCallback" />
100       </message>
101       <message name="outputlogCallback">
102             <part name="parameters" element="tns:outputlogCallback" />
103       </message>
104       <message name="errorlogCallback">
105             <part name="parameters" element="tns:errorlogCallback" />
106       </message>
107       <!-- -->
108       <message name="executeRequest">
109             <part name="parameters" element="tns:executeRequest" />
110       </message>
111       <message name="executeResponse">
112             <part name="parameters" element="tns:executeResponse" />
113       </message>
114       <message name="executeFault">
115             <part name="fault" element="tns:executeFault" />
116       </message>
117       <message name="abortRequest">
118             <part name="parameters" element="tns:abortRequest" />
119       </message>
120       <message name="abortResponse">
121             <part name="parameters" element="tns:abortResponse" />
122       </message>
123       <message name="listFilesBySimulationIdRequest">
124             <part name="parameters" element="tns:listFilesBySimulationIdRequest" />
125       </message>
126       <message name="listFilesBySimulationIdResponse">
127             <part name="parameters" element="tns:listFilesBySimulationIdResponse" />
128       </message>
129       <message name="getFileByNameAndSimulationIdRequest">
130             <part name="parameters" element="tns:getFileByNameAndSimulationIdRequest" />
131       </message>
132       <message name="getFileByNameAndSimulationIdResponse">
133             <part name="parameters" element="tns:getFileByNameAndSimulationIdResponse" />
134       </message>
135       <message name="getFileByNameAndSimulationIdFault">
136             <part name="fault" element="tns:getFileByNameAndSimulationIdFault"></part>
137       </message>
138       <message name="deleteSimulationRequest">
139             <part name="parameters" element="tns:deleteSimulationRequest" />
140       </message>
141       <message name="deleteSimulationResponse">
142             <part name="parameters" element="tns:deleteSimulationResponse" />
143       </message>
144       <message name="deleteSimulationFault">
145             <part name="fault" element="tns:deleteSimulationFault" />
146       </message>
147
148       <portType name="ImdExecuterPortType">
149             <operation name="execute">
150                   <documentation>Startet eine Simulation. Liefert als Antwort die SimulationsId.
151                         Wirft eine Exception bei ungültigen Parametern.
152                   </documentation>
```

```
153                 <input message="tns:executeRequest" />
154                 <output message="tns:executeResponse" />
155                 <fault name="fault" message="tns:executeFault" />
156           </operation>
157           <operation name="abort">
158                 <documentation>Bricht eine laufende Simulation per SimulationsId
                        ab.</documentation>
159                 <input message="tns:abortRequest" />
160                 <output message="tns:abortResponse" />
161           </operation>
162           <operation name="listFilesBySimulationId">
163                 <documentation>Erwartet eine SimulationsId.
164                       Liefert eine Liste aller Dateien der Simulationsinstanz.
165                 </documentation>
166                 <input message="tns:listFilesBySimulationIdRequest" />
167                 <output message="tns:listFilesBySimulationIdResponse" />
168           </operation>
169           <operation name="getFileByNameAndSimulationId">
170                 <documentation>Erwartet einen Dateinamen einer Simulationsinstanz.
171                       Wirft eine Exception falls die Datei nicht vorhanden ist.
172                 </documentation>
173                 <input message="tns:getFileByNameAndSimulationIdRequest" />
174                 <output message="tns:getFileByNameAndSimulationIdResponse" />
175                 <fault name="fault" message="tns:getFileByNameAndSimulationIdFault" />
176           </operation>
177           <operation name="deleteSimulation">
178                 <documentation>Löscht eine Simulationsinstanz und alle Dateien.
179                       Wirft eine Exception falls die Simulation nicht abgebrochen wurde.
180                 </documentation>
181                 <input message="tns:deleteSimulationRequest" />
182                 <output message="tns:deleteSimulationResponse" />
183                 <fault name="fault" message="tns:deleteSimulationFault" />
184           </operation>
185     </portType>
186
187     <binding name="ImdExecuterSOAP" type="tns:ImdExecuterPortType">
188           <soap:binding style="document"
189                 transport="http://schemas.xmlsoap.org/soap/http" />
190           <operation name="execute">
191                 <soap:operation soapAction="urn:execute" />
192                 <input>
193                       <soap:body use="literal" />
194                 </input>
195                 <output>
196                       <soap:body use="literal" />
197                 </output>
198                 <fault name="fault">
199                       <soap:fault use="literal" name="fault" />
200                 </fault>
201           </operation>
202           <operation name="abort">
203                 <soap:operation soapAction="urn:abort" />
204                 <input>
205                       <soap:body use="literal" />
206                 </input>
```

```
207                   <output>
208                         <soap:body use="literal" />
209                   </output>
210             </operation>
211             <operation name="listFilesBySimulationId">
212                   <soap:operation soapAction="urn:listFilesBySimulationId" />
213                   <input>
214                         <soap:body use="literal" />
215                   </input>
216                   <output>
217                         <soap:body use="literal" />
218                   </output>
219             </operation>
220             <operation name="getFileByNameAndSimulationId">
221                   <soap:operation soapAction="urn:getFileByNameAndSimulationId" />
222                   <input>
223                         <soap:body use="literal" />
224                   </input>
225                   <output>
226                         <soap:body use="literal" />
227                   </output>
228                   <fault name="fault">
229                         <soap:fault use="literal" name="fault" />
230                   </fault>
231             </operation>
232             <operation name="deleteSimulation">
233                   <soap:operation soapAction="urn:deleteSimulation" />
234                   <input>
235                         <soap:body use="literal" />
236                   </input>
237                   <output>
238                         <soap:body use="literal" />
239                   </output>
240                   <fault name="fault">
241                         <soap:fault use="literal" name="fault" />
242                   </fault>
243             </operation>
244       </binding>
245
246       <service name="ImdExecuterService">
247             <port binding="tns:ImdExecuterSOAP" name="ImdExecuterSOAP">
248                   <soap:address location="http://localhost:8080/ImdExecuterService" />
249             </port>
250       </service>
251 </definitions>
```

Listing A.5: Simulations-Service WSDL

A.2.6. Callback-Klient WSDL

Der Klient muss diese Schnittstelle implementieren um die Callback-Nachrichten vom Simulations-Service zu erhalten. Die Nachrichten enthalten Informationen über Zustandswechsel der Simulation oder ob neue Dateien erstellt worden sind.

WSDL

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
      xmlns:tns="http://imd.ws.uni-stuttgart.de/ImdCallbackClient/"
          xmlns="http://schemas.xmlsoap.org/wsdl/"
      name="ImdCallbackClient" xmlns:imd="http://imd.ws.uni-stuttgart.de/ImdExecuterService/"
      targetNamespace="http://imd.ws.uni-stuttgart.de/ImdCallbackClient/">
      <documentation>Async Callbackinterface für die Klienten.</documentation>

      <import namespace="http://imd.ws.uni-stuttgart.de/ImdExecuterService/"
            location="imd_executer.wsdl"></import>

      <portType name="ImdCallbackClientPortType">
            <operation name="checkpointCallback">
                  <documentation>Ein Checkpoint wurde von der Simulation erstellt.</documentation>
                  <input message="imd:checkpointCallback" />
            </operation>
            <operation name="outputlogCallback">
                  <documentation>Die Ausgaben der Simulation.</documentation>
                  <input message="imd:outputlogCallback" />
            </operation>
            <operation name="errorlogCallback">
                  <documentation>Die Fehlermeldungen der Simulation.</documentation>
                  <input message="imd:errorlogCallback" />
            </operation>
            <operation name="newFileCallback">
                  <documentation>Eine neue Datei wurde von der Simulation
                      erstellt.</documentation>
                  <input message="imd:newFileCallback" />
            </operation>
            <operation name="notifyCallback">
                  <documentation>Benachrichtigung an den Clienten. Zum Beispiel Status der
                      Simulation.</documentation>
                  <input message="imd:notifyCallback" />
            </operation>
      </portType>

      <binding name="ImdCallbackClientSOAP" type="tns:ImdCallbackClientPortType">
            <soap:binding style="document"
                  transport="http://schemas.xmlsoap.org/soap/http" />
            <operation name="checkpointCallback">
                  <soap:operation soapAction="urn:checkpointCallback" />
                  <input>
                        <soap:body use="literal" />
                  </input>
            </operation>
            <operation name="outputlogCallback">
                  <soap:operation soapAction="urn:outputlogCallback" />
                  <input>
                        <soap:body use="literal" />
                  </input>
            </operation>
            <operation name="errorlogCallback">
                  <soap:operation soapAction="urn:errorlogCallback" />
```

```
51                  <input>
52                        <soap:body use="literal" />
53                  </input>
54            </operation>
55            <operation name="newFileCallback">
56                  <soap:operation soapAction="urn:newFileCallback" />
57                  <input>
58                        <soap:body use="literal" />
59                  </input>
60            </operation>
61            <operation name="notifyCallback">
62                  <soap:operation soapAction="urn:notifyCallback" />
63                  <input>
64                        <soap:body use="literal" />
65                  </input>
66            </operation>
67      </binding>
68 </definitions>
```

Listing A.6: Callback-Klient WSDL

Literaturverzeichnis

[al.09] J. S. et al. IMD Benutzerhandbuch, 2009. URL http://imd.itap.physik.uni-stuttgart.de/userguide/imd.html. (Zitiert auf den Seiten 21, 22 und 24)

[BKM07] P. Bianco, R. Kotermanski, P. F. Merson. Evaluating a service-oriented architecture. 2007. (Zitiert auf Seite 15)

[BPSM+98] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, F. Yergeau. Extensible markup language (XML). *World Wide Web Consortium Recommendation REC-xml-19980210. http://www. w3. org/TR/1998/REC-xml-19980210*, 1998. (Zitiert auf Seite 16)

[BS03] P. Binkele, S. Schmauder. An atomistic Monte Carlo simulation of precipitation in a binary system. *Zeitschrift für Metallkunde*, 94(8):858–863, 2003. (Zitiert auf Seite 21)

[CCM+01] E. Christensen, F. Curbera, G. Meredith, S. Weerawarana, et al. Web services description language (WSDL) 1.1, 2001. (Zitiert auf den Seiten 11 und 16)

[CFFT06] G. Canfora, A. R. Fasolino, G. Frattolillo, P. Tramontana. Migrating interactive legacy systems to web services. In *Software Maintenance and Reengineering, 2006. CSMR 2006. Proceedings of the 10th European Conference on*, S. 10–pp. IEEE, 2006. (Zitiert auf Seite 19)

[FS12] O. K. Ferstl, E. J. Sinz. *Grundlagen der Wirtschaftsinformatik*. Oldenbourg Verlag, 2012. (Zitiert auf Seite 13)

[Gó12] P. D. J. C. M. Gómez. Enzyklopaedie der Wirtschaftsinformatik, 2012. URL http://www.enzyklopaedie-der-wirtschaftsinformatik.de/wi-enzyklopaedie/lexikon/is-management/Systementwicklung/Softwarearchitektur/Architekturparadigmen/Serviceorientierte-Architektur. (Zitiert auf den Seiten 6 und 14)

[GEM+06] T. Glatard, D. Emsellem, J. Montagnat, et al. Generic web service wrapper for efficient embedding of legacy codes in service-based workflows. In *Proceedings of the Grid-Enabling Legacy Applications and Supporting End Users Workshop*, S. 1–10. 2006. (Zitiert auf den Seiten 19 und 31)

[GKC+09] S. C. Glotzer, S. Kim, P. T. Cummings, A. Deshmukh, M. Head-Gordon, G. Karniadakis, L. Petzold, C. Sagui, M. Shinozuka. International Assessment of Research and Development in Simulation-Based Engineering and Science. Panel Report. Technischer Bericht, DTIC Document, 2009. (Zitiert auf Seite 9)

[Hot10] S. Hotta. Ausführung von Festkörpersimulationen auf Basis der Workflow Technologie. 2010. (Zitiert auf den Seiten 20 und 29)

[JEA+07] D. Jordan, J. Evdemon, A. Alves, A. Arkin, S. Askary, C. Barreto, B. Bloch, F. Curbera, M. Ford, Y. Goland, et al. Web services business process execution language version 2.0. *OASIS standard*, 11:11, 2007. (Zitiert auf Seite 11)

[KBS07] D. Krafzig, K. Banke, D. Slama. Enterprise SOA. Best Practices für Serviceorientierte Architekturen–Einführung, Umsetzung. *Praxis, Heidelberg (mitp-Verlag)*, S. 347, 2007. (Zitiert auf den Seiten 6 und 15)

[M+65] G. E. Moore, et al. Cramming more components onto integrated circuits, 1965. (Zitiert auf Seite 9)

[Mas07] D. Masak. *SOA?: Serviceorientierung in Business und Software*. Springer-Verlag New York Inc, 2007. (Zitiert auf den Seiten 6 und 18)

[Mel10] I. Melzer. *Service-orientierte Architekturen mit Web Services: Konzepte-Standards-Praxis*. Springer DE, 2010. (Zitiert auf den Seiten 13, 14 und 16)

[ML+03] N. Mitra, Y. Lafon, et al. Soap version 1.2 part 0: Primer. *W3C recommendation*, 24:12, 2003. (Zitiert auf Seite 17)

[MLM+06] C. M. MacKenzie, K. Laskey, F. McCabe, P. F. Brown, R. Metz, B. A. Hamilton. Reference model for service oriented architecture 1.0, 2006. (Zitiert auf Seite 13)

[Pel03] C. Peltz. Web services orchestration and choreography. *Computer*, 36(10):46–52, 2003. (Zitiert auf Seite 17)

[PVDH07] M. P. Papazoglou, W.-J. Van Den Heuvel. Service oriented architectures: approaches, technologies and research issues. *The VLDB journal*, 16(3):389–415, 2007. (Zitiert auf Seite 15)

[Rut09] J. Rutschmann. Generisches web service interface um simulationsanwendungen in bpel-prozesse einzubinden. 2009. (Zitiert auf den Seiten 6, 19, 31 und 32)

[S+06] H. M. Sneed, et al. Wrapping legacy software for reuse in a SOA. *Multikonerenz Wirtschaftsinformatik*, 2006, 2006. (Zitiert auf Seite 10)

[Sne00] H. M. Sneed. Encapsulation of legacy software: A technique for reusing legacy software components. *Annals of Software Engineering*, 9(1-2):293–313, 2000. (Zitiert auf Seite 30)

[TDGS07] I. J. Taylor, E. Deelman, D. Gannon, M. Shields. *Workflows for e-Science*. Springer-Verlag London Limited, 2007. (Zitiert auf Seite 9)

[VAC+05] O. Vogel, I. Arnold, A. Chughtai, E. Ihler, U. Mehling, T. Neumann, M. Völter, U. Zdun. *Software-Architektur*. Elsevier, Spektrum Akad. Verlag, 2005. (Zitiert auf Seite 13)

[WCL+05] S. Weerawarana, F. Curbera, F. Leymann, T. Storey, D. F. Ferguson. *Web services platform architecture: SOAP, WSDL, WS-policy, WS-addressing, WS-BPEL, WS-reliable messaging and more*. Prentice Hall PTR, 2005. (Zitiert auf den Seiten 6, 9, 10, 13, 14, 16, 17 und 18)

Alle URLs wurden zuletzt am 01. 04. 2014 geprüft.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift